
E-CAM Documentation

Release 0.2

Alan O'Cais

Mar 26, 2022

Contents

| | | |
|----------|---|------------|
| 1 | Classical MD Modules | 3 |
| 2 | Electronic Structure Modules | 139 |
| 3 | Quantum Dynamics Modules | 243 |
| 4 | Meso- and Multi-scale Modules | 331 |
| 5 | What is a module? | 561 |
| 6 | E-CAM Activities | 579 |
| 7 | Contributing to this documentation | 581 |
| | Bibliography | 589 |

Links**Scientific Areas**

- *Classical MD Modules*
- *Electronic Structure Modules*
- *Quantum Dynamics Modules*
- *Meso- and Multi-scale Modules*

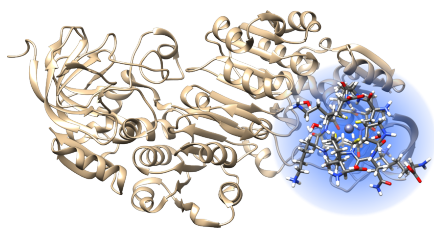
Best Practices

- *Scientific Software Best Practices*

On this page

- *The E-CAM Software Library*
 - *What is a module?*
 - * *Scientific Software Development Best Practices*
 - *E-CAM Activities*
 - * *Pilot Projects*
 - * *Extended Software Development Workshops*
 - *Contributing to this documentation*

- [search](#)



Formally, E-CAM is a European HPC Centre of Excellence supporting HPC simulations in industry and academia through software development, training and discussion in simulation and modeling. Collected in this website we have a compilation of the *software modules* that have been documented by the **E-CAM** community within the four initial target areas of **E-CAM**. These four areas represent the relative diversity of the broad domain of interest relevant to **E-CAM**, and indeed the wider **CECAM** community, and try to perform a rough categorisation of the areas of interest:

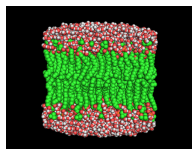
General Information**Contents**

- *Classical MD Modules*
 - *Introduction*
 - *Rare events and path sampling*
 - *OpenPathSampling*
 - *Machine Learning Potentials*
 - *n2p2*
 - *Pilot Projects*
 - *Extended Software Development Workshops (ESDWs)*
 - *European Environment for Scientific Software Installations*

- *How to contribute?*

- [search](#)

1.1 Introduction



This is a collection of the modules that have been created by **E-CAM** community within the area of Classical MD. This documentation is created using ReStructured Text and the git repository for the documentation source files can be found at <https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library> which are open to contributions from E-CAM members.

In the context of E-CAM, the definition of a software module is any piece of software that could be of use to the E-CAM community and that encapsulates some additional functionality, enhanced performance or improved usability for people performing computational simulations in the domain areas of interest to us.

This definition is deliberately broader than the traditional concept of a module as defined in the semantics of most high-level programming languages and is intended to capture inter alia workflow scripts, analysis tools and test suites as well as traditional subroutines and functions. Because such E-CAM modules will form a heterogeneous collection we prefer to refer to this as an E-CAM software repository rather than a library (since the word library carries a particular meaning in the programming world). The modules do however share with the traditional computer science definition the concept of hiding the internal workings of a module behind simple and well-defined interfaces. It is probable that in many cases the modules will result from the abstraction and refactoring of useful ideas from existing codes rather than being written entirely de novo.

Perhaps more important than exactly what a module is, is how it is written and used. A final E-CAM module adheres to current best-practice programming style conventions, is well documented and comes with either regression or unit tests (and any necessary associated data). E-CAM modules should be written in such a way that they can potentially take advantage of anticipated hardware developments in the near future (and this is one of the training objectives of E-CAM).

1.2 Rare events and path sampling

In many simulations, we come across the challenge of bridging timescales. The desire for high resolution in space (and therefore time) is inherently in conflict with the desire to study long-time dynamics. To study molecular dynamics with atomistic detail, we must use timesteps on the order of a femtosecond. However, many problems in biological chemistry, materials science, and other fields involve events that only spontaneously occur after a millisecond or longer (for example, biomolecular conformational changes, or nucleation processes). That means that we would need around 10^{12} time steps to see a single millisecond-scale event. This is the problem of “rare events” in theoretical and computational chemistry.

While modern supercomputers are beginning to make it possible to obtain trajectories long enough to observe some of these processes (such as [millisecond dynamics of a protein](#)), even then, we may only find one example of a given transition. To fully characterize a transition (with proper statistics), we need many examples. This is where path sampling comes in. Path sampling approaches obtain many trajectories using a Markov chain Monte Carlo approach: An existing trajectory is perturbed (usually using a variant of the “shooting” move), and the resulting trial trajectory is accepted or rejected according to conditions that preserve the distribution of the path ensemble. As such, path sampling is Monte Carlo in the space of paths (trajectories). Conceptually, this enhances the sampling of transitions by focusing on the transition region instead of the stable states. In direct MD, trajectories spend much more time in stable states than in the transition region (exponential population differences for linear free energy differences); path sampling skips over that time in the stable states.

The main path sampling approaches used in the modules below are [transition path sampling](#) (TPS) and [transition interface sampling](#) (TIS). In practice, TPS is mainly used to characterize the mechanism of a transition, while TIS (which is more expensive than TPS) is used to calculate rates and free energy landscapes. Overviews of these methods, as well as other rare events methods, can be found in the following review articles:

- [2010 review by Bolhuis and Dellago in Reviews in Computational Chemistry](#)
- [2008 review by Dellago and Bolhuis in Advances in Polymer Science](#)

In addition, several other resources are available on the web to teach path sampling, including:

- [Wikipedia entry on path sampling](#)
- [Aaron Dinner’s tutorial on path sampling](#)

Since the problem of bridging timescales, which path sampling addresses, is a generic one, path sampling can be used in many fields. Indeed, there’s nothing in the methodology that even restricts it to molecular simulation. However, it is best known in the field of classical MD simulations, where path sampling methods have shown many successes, including:

- [Mechanisms of complex chemical reactions, such as autoionization of water](#)
- [Mechanism of hydrophobic assembly](#)
- [Evidence that the glass transition is a first-order phase transition](#)
- [Mechanism of crystal nucleation](#)
- [Mechanism of cavitation](#)
- [Identifying new mechanisms in catalytic systems](#)
- [Characterization of the conformational dynamics networks in proteins](#)

As computational resources become more powerful, path sampling has the promise to provide insight into rare events in larger systems, and into events with even longer timescales. For example:

- [Drug/protein binding and unbinding](#) (timescales of minutes), which is essential for predicting the efficacy of drugs

- Association processes of proteins (large systems), which is at the core of communication in biochemical pathways
- Self assembly processes for complex systems (many intermediates), which can be important for the design of new materials

Further, applying the known successes of path sampling methods to larger systems can also be quite valuable. Path sampling can shed light on the networks of conformational dynamics for large proteins and protein complexes, and on the mechanisms and rates of complex reactions and phase transitions. The range of possibilities is so broad that it is impossible to enumerate – both academics and industry will benefit greatly from having software for these methods.

The modules listed here deal with software to perform path sampling methods, as well as other approaches to rare events.

1.3 OpenPathSampling

Several modules were developed based on [OpenPathSampling \(OPS\)](#). These include modules that have been incorporated into the core of OPS, as well as some that remain separate projects. The modules that were incorporated into the core are:

1.3.1 Path Density for OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module implements path density calculations for OpenPathSampling, including a generic multidimensional sparse histogram, and plotting functions for the two-dimensional case.

Purpose of Module

Path density is one of the useful ways to visualize a path ensemble generated by path sampling. At first glance, a typical path density plot may appear similar to a two-dimensional free energy landscape plot. They are both “heatmap”-type plots, plotting a two-dimensional histogram in some pair of collective variables. However, path density differs from free energy in several important respects:

- A path density plot is histogrammed according to the number of paths, not the number of configurations. So if a cell is visited more than once during a path, it still only gets counted once.
- A path density plot may interpolate across cells that the path jumps over. This is because it is assumed that the input must actually be continuous.

These differences can prevent metastable regions from overwhelming the transition regions in the plot. When looking at mechanisms, the path density is a more useful tool than the raw configurational probability.

The implementation in this module includes:

- A `SparseHistogram` class for histogramming in arbitrary dimensions (allowing path densities in more than the traditional two dimensional plot). This class can also be used for other purposes (such as free energy histograms).
- A `PathHistogram` subclass of `SparseHistogram`, which includes the options to normalize either per-path or per-configuration, and the tools for interpolating across skipped cells.
- A `PathDensityHistogram` subclass of `PathHistogram`, which includes the ability to convert trajectories to representations of the trajectories in some list of collective variables, and sets reasonable default behavior for normalization and interpolation.
- A `HistogramPlotter2D` that is specialized for plotting a 2D histogram, especially a `PathDensityHistogram`.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `nosetests` from the root directory of the repository.

Examples

The examples below link to both a notebook (in the `OpenPathSampling` GitHub repository) and that same notebook in the context of the `OpenPathSampling` official documentation. Note that these examples represent the most recent version of code, and may not be identical to what was included in the module. The original examples are included in the source code, below.

- Options for `PathHistogram` [[PathHistogram on GitHub](#) | [PathHistogram Docs](#)]

- Use of `PathDensityHistogram` within analysis of alanine dipeptide TPS: [[PathDensityHistogram on Github](#) | [PathDensityHistogram Docs](#)]

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/504>
- <https://github.com/openpathsampling/openpathsampling/pull/506>
- <https://github.com/openpathsampling/openpathsampling/pull/511>

1.3.2 Direct MD (on-the-fly) flux/rate in `OpenPathSampling`

Software Technical Information

The information in this section describes `OpenPathSampling` as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module contains code to implement the direct (on-the-fly) calculation of flux and rate in `OpenPathSampling`.

Purpose of Module

This calculates the flux out of a state and through an interface, or the rate of the transition between two states, while running a trajectory. This means that you don't have to save all the frames of the trajectory. This is especially useful for small (toy) systems, where you can easily run very long trajectories to get very accurate results, and would rather re-run than save the full trajectory. A separate module calculates the rate and flux from an existing trajectory.

Calculating the flux out of a state and through an innermost interface is one of the steps required in transition interface sampling (TIS). This module enables that. Although there are other ways to calculate the flux (for example, replica exchange TIS can calculate the flux as a byproduct of the "minus move"), frequently a direct calculation gives the best

statistics. In addition, an approximate flux should be calculated before running the simulation in order to ensure that the state and innermost interface definitions have been chosen well.

In principle, this module also makes it possible to calculate rates directly by running long molecular dynamics simulations. That won't be useful for very rare events, but it can be useful for validation.

The primary object implemented in this module is the `DirectSimulation` subclass of `PathSimulator`, which performs on-the-fly analysis of flux and rate.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `nosetests` from the root directory of the repository.

Examples

Flux for MISTIS: [\[GitHub | Docs\]](#)

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/495>

1.3.3 Improved input for OPS networks

Software Technical Information

The information in this section describes `OpenPathSampling` as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module includes modifications to `OpenPathSampling` that simplify the setup of transition networks, including providing a method so that extra information about interfaces can (optionally) be provided on setup in order to simplify analysis.

Purpose of Module

In `OpenPathSampling`, “reactions” (which are often conformational changes) are represented by objects called `Transitions`, and the set of all reactions of interest is represented by a `TransitionNetwork`. The `TransitionNetwork` knows about all the reactions being sampled, as well as the path ensembles used to sample them. In general, a full reaction network might involve hundreds of path ensembles, so the `TransitionNetwork` is a factory that creates the ensembles so the user doesn’t have to, and also provides conveniences for analysis, such as grouping the path ensembles according to the reaction they sample.

This module deals with two aspects of transition interface sampling methods. The first is the interface set, which is the group of interfaces associated with a given transition. These interfaces are associated with volumes in phase space. Those volumes are typically defined by a maximum value of some order parameter, λ . Knowing this edge value is essential for calculating rates.

The second aspect is the multiple state outer (MS-outer) interface ensemble. This is an ensemble used in some variants of multiple state transition interface sampling to facilitate replica exchange between paths with different initial states. In practice, this approach is likely to become less frequently used (there are more efficient approaches to achieve the same goals), so requiring that the MS-outer interface exist is not very forward-thinking, although removing entirely is also not the best approach.

Prior to the improvements made in this module, `OpenPathSampling` suffered from the following problems:

- An interface had no way of knowing what its “edge” value was, only whether a given snapshot was inside it or outside it. This made it difficult to automatically determined the value at the outermost interface for calculating the rate. The previous code relied on a hack that assumed that trajectories in the interface had a relatively low probability of crossing to another state.
- All networks required a multiple state outer interface, even if it wasn’t used. This also meant that the outermost interface a user defined was converted to an MS-outer interface, which could lead to confusion. This module makes usage much easier to understand.

This module changes the setup of interface sets, such that they can identify their edge values (if it is uniquely identified; the code still works if it is not unique). This obviates the need for an ugly hack to guess where the boundary was.

This module also changes the way that the multiple state outer interface is set up. Now the user must explicitly make a multiple state outer interface object, which will then make the appropriate MS-outer ensemble. We wanted to keep the ability to have an MS-outer ensemble. However, we did not want to require it, because there are, in general, better approaches to accomplish the same things.

The primary new objects in this module are:

- `InterfaceSet`: replace the previous list of `Volume` objects with a proper set of interfaces, which can be associated with a list of `lambdas`.

- `VolumeInterfaceSet`: subclass of `InterfaceSet`, intended to directly replace the old functions to create several volumes at once. Also has the ability to automatically create a new interface based on the new value of the maximum lambda. This also makes it potentially useful in methods where the interfaces should be treated parametrically, such as adaptive multilevel splitting.
- `MSOuterTISInterface`: object to create the multiple state outer interface for transition interface sampling. Mainly allows us to smoothly transition away from using this sort of object, since there are better approaches to solve the same problems.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `nosetests` from the root directory of the repository.

Examples

The first example demonstrates how to use the new classes that have been incorporated in `OpenPathSampling` through this module. This includes some of the new features of interface sets, such as identifying values of lambda and creating new interfaces based on a desired “edge” value, as well as a couple approaches to building an MS-outer interface, and how to build a `TransitionNetwork` with or without MS-outer interfaces.

- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/network_input.ipynb [HTML]

The example below links to the official `OpenPathSampling` documentation. The notebooks that make up that example can also be found in the `OpenPathSampling` GitHub repository. Note that this example represents the most recent version of code, and may not be identical to what was included in the module. The original example is included in the source code, below. In this second example, usage of this module is illustrated in the context of a larger example of MSTIS.

- [Multiple State TIS on a Toy Model](#)

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/528>
- <https://github.com/openpathsampling/openpathsampling/pull/530>
- <https://github.com/openpathsampling/openpathsampling/pull/538>
- <https://github.com/openpathsampling/openpathsampling/pull/553>

1.3.4 New WHAM code

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module includes a re-write of the OpenPathSampling reweighted histogram analysis code. This fixes limitations and is more readable than the previous version.

Purpose of Module

Methods like transition interface sampling (TIS) sample multiple ensembles, and then combine the results from the individual restricted ensembles to obtain results for the unrestricted (natural) ensemble, such as the rates of a reaction or projections of the free energy surface. One approach to combining these ensembles is the weighted histogram analysis method (WHAM). This module provides an implementation of WHAM that is specialized for path sampling. Details about the WHAM method (as used for calculating free energies) can be found in [Frenkel and Smit](#), section 7.3.

The module is a rewrite of previous code in OPS. The previous code had several limitations, most notably the assumption that all ensembles had the same number of sampling. Practical cases required that the number of samples in each ensemble be allowed to vary. In addition, the previous code was poorly documented and untested. This module fixes all of that, and includes detailed comments connecting the code to the equations in Frenkel and Smit.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling use the `nose` package.

This module has been included in the OpenPathSampling core. Its tests can be run by setting up a developer install of OpenPathSampling and running the command `nosetests` from the root directory of the repository.

Examples

An example of how to use this code can be found at:

- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/wham.ipynb

Further cases where this has been used are implicit in the analysis notebooks in OpenPathSampling.

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/541>

1.3.5 Flux/Rate Analysis in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module adds the ability to use existing trajectories to calculate the flux through an interface or the rate of a transition in OpenPathSampling.

Purpose of Module

Calculating the flux out of a state and through a given interface is an important step in transition interface sampling (TIS). It is required for the rate calculation, and can be used to determine the best location for the innermost interface before performing the main TIS sampling.

In many cases, the main TIS sampling is only performed after a significant amount of direct MD has been performed. For example, MD trajectories may have already been used to identify and test the stability of stable states. This module calculates the flux (as well as the rate, a related quantity) from existing trajectories. These can be trajectories generated with OPS or loaded into OPS from other simulation packages, such as Gromacs.

The flux calculation is used as part of obtaining the rate in TIS. While some variants of TIS calculate the flux as part of their primary sampling, others need an auxiliary calculation. In addition, the flux calculation is always useful for confirming a good definition of the innermost interface in TIS, and using an existing trajectory to select the location of the innermost interface does not require re-running the dynamics.

The rate calculation will probably not be used very much, because rates require extremely long trajectories. Both flux and rate are included as a single “module” because the code is very closely related.

This module analyses previously existing trajectories. Another module will calculate these things on-the-fly with an OPS engine.

The primary new objects in this module are:

- `TrajectoryTransitionAnalysis`: Contains all the methods to perform the analysis. The overall approach is based on identifying subtrajectory segments that correspond to various conditions, e.g., time in one volume before entering another volume (lifetimes, which are then related to the rate). By keeping these subtrajectories, this module also enables visualization on time traces of the trajectory.
- `TrajectorySegmentContainer`: Container class for a list of trajectory segments, created as results of the `TrajectoryTransitionAnalysis` object. Also includes conveniences for reporting by either number of frames or time: each is reported as a numpy array, so that numpy’s built in statistics can be used for, e.g., mean or standard deviation.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `nosetests` from the root directory of the repository.

Examples

Examples for this have been provided in the `ops_additional_examples` repository. In particular, the Jupyter notebooks:

- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/transition_analysis_Abl.ipynb
- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/DNA_flux_example.ipynb

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/435>
- <https://github.com/openpathsampling/openpathsampling/pull/448>
- <https://github.com/openpathsampling/openpathsampling/pull/451>
- <https://github.com/openpathsampling/openpathsampling/pull/654>

1.3.6 OpenPathSampling Snapshot Features

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module includes several new OpenPathSampling snapshot “features,” which make attributes directly accessible from the snapshot object. In particular, this includes support for `masses`, `n_degrees_of_freedom`, and `instantaneous_temperature` in both the toy and OpenMM engines.

Purpose of Module

In OpenPathSampling, certain quantities can be accessed directly from each snapshot. The standard examples of such “features” are things like coordinates and velocities, which are stored for each snapshot. However, additional features can also be added, which may not require per-snapshot storage. This approach makes them accessible from the snapshot as `snapshot.feature`, just like a stored quantity, even if they are not stored. For example, the `snapshot.masses` can actually be a pointer to a single array of masses for all snapshots from a given engine, and `snapshot.instantaneous_temperature` can actually be a quantity that is computed on demand, rather than stored. This module makes several new features available for snapshots from the OpenMM engine and from the toy engine.

By adding a standard interface for snapshots to provide similar information, this provides several advantages:

- Many analysis tools require the masses. This module makes it easier to apply analysis tools for one MD engine to the results from another MD engine.
- Other modules will require a standardized interface. In particular, the two-way shooting module will require both the masses and the number of degrees of freedom, and future modules for collective variables are also likely to need to masses.
- This provides examples of how to implement snapshot “features,” which are necessary for the implementation of new engine modules.
- The instantaneous temperature, in particular, is an important check that a simulation has been well-behaved. Drift of the instantaneous temperature is a sign of a problem in the simulation.

Included in this implementation are:

- `masses`: was already available in toy, but now also available in OpenMM. Only stored once (in the `engine`), but accessible from any snapshot.
- `n_degrees_of_freedom`: added for both OpenMM and toy engines. Calculates the number of degrees of freedom the fly.
- `instantaneous_temperature`: added for both OpenMM and toy engines. Calculated on the fly (requires calculation of `n_degrees_of_freedom` and of kinetic energy).

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling use the `nose` package.

This module has been included in the OpenPathSampling core. Its tests can be run by setting up a developer install of OpenPathSampling and running the command `nose` from the root directory of the repository.

Examples

An example of these features in use can be found at:

- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/snapshot_features_1.ipynb [HTML]

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/579>
- <https://github.com/openpathsampling/openpathsampling/pull/589>
- <https://github.com/openpathsampling/openpathsampling/pull/649>

1.3.7 Two-Way Shooting in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module adds the ability to do two-way shooting to OpenPathSampling.

Purpose of Module

Different types of dynamics are better suited for different kinds of Monte Carlo moves in path sampling. “One-way shooting,” which was already implemented in OpenPathSampling, is an efficient approach for sampling paths when the dynamics are chaotic or diffusive. However, it requires stochastic dynamics, and therefore isn’t appropriate for deterministic dynamics, as should be used with ballistic processes. For ballistic processes and deterministic dynamics, the “two-way shooting” move, which is implemented in this module, should be used. These moves differ in that one-way shooting selects a frame as a shooting point and evolves the trajectory *either* forward or backward, keeping part of the original trajectory, whereas two-way shooting selects a shooting point, modified it (usually by changing the velocities) and evolves *both* forward and backward. In one-way shooting, the stochastic dynamics obviates the need to modify the shooting point.

The shooting point selection methods used by one-way shooting can be re-used for two-way shooting. However, two-way shooting requires a `SnapshotModifier`, which one-way shooting does not. The basics of the `SnapshotModifier`, as well as an implementation which completely randomizes the velocities according to the Boltzmann distribution, were included in a module to do committor simulations.

This module implements the movers and move strategies necessary to support two-way shooting:

- `AbstractTwoWayShootingMover`, subclass of `EngineMover`. There’s a possibility that generalized ensembles may behave differently if you shoot the forward part of the path first, versus the backward part of the path. This isn’t the case for common TIS and TPS ensembles, but in case it is important for other ensembles, we create separate forward-first and backward-first two way shooting movers. This class is a common abstract base class for both.

- `ForwardFirstTwoWayShootingMover`, subclass of `AbstractTwoWayShootingMover`, which implements two-way shooting where the forward half-trajectory is generated first.
- `BackwardFirstTwoWayShootingMOver`, subclass of `AbstractTwoWayShootingMover`, which implements two-way shooting where the backward half-trajectory is generated first.
- `TwoWayShootingMover`, subclass of `RandomChoiceMover`, which randomly selects to either shoot forward-first or backward-first.
- `TwoWayShootingStrategy`, subclass of `MoveStrategy`, which generates a `TwoWayShootingMover` for every ensemble of interest.

In addition, this module implements new snapshot modifiers to change the velocities of the shooting point:

- `GeneralizedDirectionModifier`, subclass of `SnapshotModifier`, which acts as an abstract base class for the specific velocity direction modifiers.
- `VelocityDirectionModifier`, subclass of `GeneralizedDirectionModifier`, which modifies all the velocities of the selected subset of atoms.
- `SingleAtomVelocityDirectionModifier`, subclass of `GeneralizedDirectionModifier`, which modifies the velocities on one of the selected subset of atoms.

Beyond these, the previously existing `RandomVelocities` snapshot modifier is also likely to be used with two-way shooting.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `nosetests` from the root directory of the repository.

Examples

An example of this can be found at:

- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/two_way_shooting.ipynb

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/600>
- <https://github.com/openpathsampling/openpathsampling/pull/650>
- <https://github.com/openpathsampling/openpathsampling/pull/652>

1.3.8 Committor Analysis in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module adds a simulator to perform committor analysis in OpenPathSampling, given a set of initial points to shoot from.

Purpose of Module

The committor for a given configuration (in the context of some transition $A \rightarrow B$) is defined as $p_B(x)$, the probability that a trajectory beginning at configuration x will reach state B before state A . The isosurfaces of the committor are a good definition of the reaction coordinate (the probability of ending in the product state is certainly a measure of the progress of the reaction). The transition state will have an equal chance of going to either state, so configurations with a committor of approximately 50% are said to make up the “transition state ensemble.” As a result, a committor simulation is essential both for the definition of the reaction coordinate and for the identification of a proposed transition state. This module provides a straightforward way of calculating the committor for a given set of initial conditions.

In addition to calculating the committor, this module can be used to generate more physical transition trajectories from unphysical ones. A trajectory that connects the two states and has the same initial configuration could be a good candidate for an initial path sampling trajectory. The only unphysical aspect of such a trajectory is the sudden kink in velocities, which usually be removed after a short equilibration with path sampling.

The implementation in this module includes:

- `SnapshotModifier` abstract class to change a snapshot, along with concrete subclasses `NoModification` (used in testing) and `RandomVelocities` (used for committor analysis). This same class of object will be reused for two-way shooting.
- A `CommittorSimulation` subclass of `PathSimulator` to run the committor simulation.
- A generic `TransformedDict` object which acts as a dictionary, but applies an arbitrary key-altering function before accessing the keys.

- `SnapshotByCoordinateDict`, a subclass of `TransformedDict`, which uses the coordinates of a snapshot as the internal keys. Thus multiple snapshots with the same coordinates can map to the same values, regardless of their velocities.
- `ShootingPointAnalysis`, a subclass of `SnapshotByCoordinateDict`, which performs the analysis of shooting points. This includes calculating the committor and making 1D and 2D histograms of the committor (mapped with arbitrary axes).

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `nosetests` from the root directory of the repository.

Examples

- OPS docs committor example [[Committor GitHub](#) | [Committor Docs](#)]
- Alanine dipeptide committor example [[Alanine dipeptide committor GitHub](#) | [Alanine dipeptide committor Docs](#)]

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/450>
- <https://github.com/openpathsampling/openpathsampling/pull/454>
- <https://github.com/openpathsampling/openpathsampling/pull/466>
- <https://github.com/openpathsampling/openpathsampling/pull/601>
- <https://github.com/openpathsampling/openpathsampling/pull/618>
- <https://github.com/openpathsampling/openpathsampling/pull/647>

1.3.9 OPS Channel Analysis

Software Technical Information

The information in this section describes `OpenPathSampling` as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

In many cases, more than one channel is available to a system – either because there are multiple channels between two states, or because there are multiple states, and transitions between each pair is a different channel. This module provides tools to identify which paths are in each channel, and to study the statistical behavior of the switching between channels.

Purpose of Module

In practical examples, more than one channel (i.e., mechanism) can occur during a path sampling simulation. This is inherently the case if you have multiple stable states, since the transition between each pair of states will be of separate interest. It can also be the case when you have a single pair of states, but multiple channels that connect the states.

This module uses the `OPS Ensemble.split` function to study how a simulation samples these channels. The user must provide a list of possible channels, described as `OPS Ensemble` objects. From this, each path is analyzed, and various statistical behavior about the sampling process can be determined.

The main object added in this module is the `ChannelAnalysis` object, which performs this analysis and stores the results. Once the analysis has been performed, several properties can be extracted, including:

- `switching_matrix`: how many times a switch from one channel to another occurred
- `residence_times`: the number of MC steps spent with the path in each channel (returns the entire list so the user can calculate distribution properties with, e.g., `numpy`)
- `total_time`: total number of MC steps spent in each channel
- `status(step_num)`: the channel the simulation was in for a given step number

In principle, a path might satisfy the requirement for more than one channel at a time. This analysis class allows for that, and gives the user the option of setting its `treat_multiples` attribute:

- `newest`: use the most recent channel entered
- `oldest`: use the least recent channel entered
- `multiple`: treat multiple channels as a new type of channel, e.g., ‘a’ and ‘b’ because ‘a,b’
- `all`: treat each channel individually, despite overlaps. For `status` this is the same as ‘multiple’

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling use the `nose` package.

This module has been included in the OpenPathSampling core. Its tests can be run by setting up a developer install of OpenPathSampling and running the command `nose tests` from the root directory of the repository.

Examples

- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/channel_analysis.ipynb

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull request:

- <https://github.com/openpathsampling/openpathsampling/pull/658>

1.3.10 OPS New TIS Analysis

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

This module provides a new framework for analyzing transition interface sampling simulations using OpenPathSampling. The previous analysis tools gave no flexibility to the user, were not easily extendable, and had no unit tests. This module fixes all of that.

Purpose of Module

Transition interface sampling (TIS) is a powerful rare events method with a particular focus on calculating the rates of reactions. The core idea starts by splitting the rate k_{AB} into a product:

$$k_{AB} = \phi_{A_0} P_A(B|\lambda_0)$$

where k_{AB} is the rate from state A to state B , ϕ_{A_0} is the flux out of state A and through an interface λ_0 , and $P_A(B|\lambda_0)$ is the transition probability of that a trajectory enters B before any other state given that has exited the interface λ_0 , starting in state A .

TIS further splits the transition probability into several conditional probabilities, by adding a set of m interfaces (surfaces in phase space) $\{\lambda_i\}$, with λ_0 as the innermost. Mathematically, this gives us:

$$P_A(B|\lambda_0) = P_A(B|\lambda_m) \prod_{i=0}^{m-1} P_A(\lambda_{i+1}|\lambda_i)$$

By sampling trajectories that necessarily cross each given interface λ_i , TIS provides the information that can be used to determine $P_A(\lambda_{i+1}|\lambda_i)$. However, there are several approaches have been developed/proposed to efficiently turn the sampling data into a best estimate of the transition probability.

The previous analysis in OPS took one of those method, and provided very little room to customize the procedure. This module makes it so that it is easier to customize the analysis or to use different approaches to calculate the various terms that make up the TIS rate expression.

A much more detailed description of the TIS analysis as implemented here is given in the core OPS documentation, which was also contributed as part of this module. That section of the documentation is online at http://openpathsampling.org/latest/topics/tis_analysis.html

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling use the `nose` package.

This module has been included in the OpenPathSampling core. Its tests can be run by setting up a developer install of OpenPathSampling and running the command `nosetests` from the root directory of the repository.

Examples

Example in the OPS repository on using this:

- https://github.com/openpathsampling/openpathsampling/blob/master/examples/toy_model_mstis/toy_mstis_A3_new_analysis.ipynb

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull request:

- <https://github.com/openpathsampling/openpathsampling/pull/686>

1.3.11 Resampling Statistics

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

The module provides tools for resampling (e.g., statistical bootstrapping) in the context of `pandas` `DataFrames` in general and specifically OpenPathSampling. This provides tools for estimating statistical error on multiple quantities simultaneously.

Purpose of Module

Providing an estimate of uncertainty is essential when presenting scientific results. This module provides the ability to perform statistical analysis of a large simulation from OpenPathSampling by using the sampled trajectories to create subsamples, which are then assumed to be independent. The subsamples are analyzed separately, and this module makes it easy to obtain mean, standard deviation, or percentile values. In particular, this module provides the tools to do such an analysis on functions that return a table of data using a `pandas.DataFrame` object, as the OpenPathSampling rate matrix calculation does.

Most of the code is generic, and could be used for any function that produces a `pandas.DataFrame` as its output. Therefore this module may be useful for many projects other than OpenPathSampling. Within OpenPathSampling, this can be used to obtain statistics on rates, fluxes, and other such quantities.

These tools are implemented in two main classes. The first is `BlockResampling`, which organizes the input (MC steps in OPS) into blocks to be passed to a function that does the analysis. This allows us to obtain several results for the analysis. The second is `ResamplingStatistics`, which takes those blocks and a function (that returns a `pandas.DataFrame`) as input. It then applies that function to each of those blocks, and then makes it easy to access properties such as the mean, standard deviation, or percentile values for each frame element.

While `BlockResampling` is the only resampling method implemented in the module (as it is the one needed for TIS rate calculations), it would be straightforward to extend this framework with other resampling methods, such as variants of bootstrapping.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `nosetests` from the root directory of the repository.

Examples

- https://gitlab.e-cam2020.eu/dwhswenson/ops_additional_examples/blob/master/resampling_statistics.ipynb

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/684>

1.3.12 Gromacs engine in OpenPathSampling

Software Technical Information

The information in this section describes `OpenPathSampling` as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7, 3.6, 3.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

Authors David W.H. Swenson

- *Purpose of Module*
- *Background Information*

- *Testing*
- *Examples*
- *Source Code*

This module adds support for Gromacs as an engine for OpenPathSampling.

Purpose of Module

Different molecular dynamics (MD) codes have developed to serve different communities. Gromacs is one of the major MD codes for the biomolecular community, and even though much of its functionality can be reproduced by other MD codes, such as OpenMM, there are still some extensions that are built on top of Gromacs that haven't been ported to other codes. For example, the MARTINI coarse-grained model is not available other codes such as OpenMM.

Additionally, people who are familiar with a given MD package will prefer to continue to work with that. Therefore codes that wrap around MD packages, as OpenPathSampling does, can expand their reach by adding ways to interface with other MD packages.

This module adds the Gromacs engine for OpenPathSampling. It is the first practical test of the external engine API of OPS.

Specific functionality in this module includes:

- `GromacsEngine`: the OPS dynamics engine, based on the `ExternalEngine`, that runs Gromacs as an external tool. Option on initialization allow the user to customize the path to the Gromacs executable.
- `ExternalMDSnapshot`: an OPS snapshot for external MD engines, which contains coordinates, velocities, and box vectors. Requires that the engine implement a `read_frame_data` method to load from a specific MD trajectory.
- `snapshot_from_gro`: a function that creates an OPS snapshot from a Gromacs `.gro` file.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

This module is in a development branch of OpenPathSampling. If you have conda installed, this branch of OPS can be installed by downloading the `conda_ops_dev_install.sh` script and running it with the command:

```
source conda_ops_dev_install.sh dwhswenson gromacs_engine
```

This will download a new copy of the OPS git repository, select the `gromacs_engine` branch from the `dwhswenson` fork, install the requirements, and create an editable install of OPS. If you would like to do this in a new conda environment, set the environment variable `OPS_ENV`, and it will install in a new environment with the name `$OPS_ENV`.

To run tests, you may need `pytest`, which can be installed with `conda install pytest`.

The entire OPS test suite can be run with `run with py.test --pyargs openpathsampling`. Tests specific to the Gromacs engine can be run with `py.test --pyargs openpathsampling.tests.test_gromacs_engine`.

Examples

- An example can be found here: https://github.com/dwhswenson/openpathsampling/tree/gromacs_engine/examples/gromacs

Source Code

This module is contained in the following pull request:

- <https://github.com/openpathsampling/openpathsampling/pull/819>

1.3.13 OPS Visit All States Ensemble

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7, 3.6, 3.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

Software module developed by David W.H. Swenson

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

This module adds a convenient new OpenPathSampling ensemble that allows trajectories to continue until they have visited all the states in the system. In addition, it provides real-time reporting about the progress.

Purpose of Module

One of the ways to get initial trajectories for path sampling is to use dynamics that aren't physical for the ensemble of interest, such as using an increased temperature. If a trajectory has a frame in every state, then it must have

subtrajectories that connect from each state to another one, and therefore it has all the information to start a MSTIS simulation. The ensemble definition tools in OPS make it easy to create such custom sequential ensembles¹.

However, users often want to do simulation setup in an interactive mode, such as in a Jupyter notebook, or at a minimum want to have a sense of the progress made on a long trajectory such as this. The default OPS ensemble gives no output and therefore no sense of how much progress has been made.

This module provides a custom OPS ensemble that gives such output during its simulation. It outputs the length of the trajectory so far, as well as the states that have and have not already been visited. This gives a much better sense of how long the simulation will take to run.

This module includes:

- `default_state_progress_report`: A function to create the progress report string. This can be replaced by another function to customize the output.
- `VisitAllStatesEnsemble`: A class that wraps an OPS `SequentialEnsemble`. The `can_append` method, called while generating the dynamics, can output information about the progress. The behavior of this output can be set with the initialization variable `progress`, which can take the values of `default` for the default output, `silent` for no output, or can take a 2-tuple of callables where the first determines what to write, and the second determined how to emit the information.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use `pytest`.

This module has been included in the `OpenPathSampling` core. Its tests can be run by installing `pytest` and OPS (with commit 8e767872, which will be part of release 1.0 and later), and running the command `py.test --pyargs openpathsampling`.

Examples

This module is used in the OPS [alanine dipeptide MSTIS example](#), during the creation of initial trajectories.

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/826>

¹ D.W.H. Swenson, J.-H. Prinz, F. Noe, J.D. Chodera, and P.G. Bolhuis. J. Chem. Theory Comput. **15**, 837 (2019); <http://doi.org/10.1021/acs.jctc.8b00627>

1.3.14 Interface-Constrained Shooting in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

Software module developed by Peter G. Bolhuis and David W.H. Swenson

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

This module adds interface-constrained shooting to OpenPathSampling. Interface-constrained shooting is a technique that can improve the efficiency of transition interface sampling.

Purpose of Module

In transition interface sampling (TIS), one defines stable states (volumes in phase space) and interfaces (surfaces in phase space). For a trajectory to be accepted in TIS, it must begin with exactly one frame in a given initial state, cross the interface, and end with exactly one frame in any state volume (including the initial state).

New trajectories are generated with the shooting move, which selects a point along an initial trajectory from which new frames can be made. In one-way shooting, the dynamics only needs to run in one direction (with the stochastic nature of the dynamics ensuring that a new trajectory is generated).

However, if the interfaces are far from the initial state and if all frames are equally likely to be used for shooting, it can be very likely for the shooting point to come before the trajectory has crossed the interface. This can then lead to shooting moves that usually generate trajectories that don't cross the interface, and therefore must be rejected. This uses a lot of simulation effort without generating useful new trajectories.

Interface-constrained shooting (also called “constrained interface shooting”)¹ is an approach to solve this problem. Instead of selecting from anywhere along the trajectory, only the first point after crossing the interface is allowed as a shooting point. This ensures that every trajectory that is generated will be valid (will cross the interface). In addition, because the first crossing is still the first crossing in the new trajectory, this leads to the Metropolis acceptance probability also being 1. Therefore, every trial trajectory is accepted.

In practice, this must be combined with the path reversal move in order to sample all of trajectory space. The result is an approach with very high acceptance, although decorrelation of the trajectory is a little slower.

¹ Bolhuis, P. G. (2008). Rare events via multiple reaction channels sampled by path replica exchange. The Journal of Chemical Physics, 129(11), 114108. <https://doi.org/10.1063/1.2976011>

This module implements interface-constrained shooting using:

- `ForwardShootingStrategy`: An `OPS MoveStrategy` to do forward-only shooting. The interface-constrained shooting approach uses forward-only stochastic dynamics, counting on path reversal to handle the backward-time dynamics.
- `InterfaceConstrainedSelector`: A `ShootingPointSelector` that selects the first point outside the given volume (the boundary of which defines the interface).

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use `pytest`.

This module has been included in the `OpenPathSampling` core. Its tests can be run by installing `pytest` and `OPS` (with commit `c340818`, which will be part of release 0.9.6 and later), and running the command `py.test --pyargs openpathsampling`.

Examples

An example of this is in the following notebook:

- https://github.com/openpathsampling/openpathsampling/blob/7e157661dd8633690ebfcae4a8265fc14e31c5b9/examples/toy_model_mstis/toy_mstis_A4_constrained_shooting.ipynb

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/788>
- <https://github.com/openpathsampling/openpathsampling/pull/790>
- <https://github.com/openpathsampling/openpathsampling/pull/800>

1.3.15 Progress meters in OPS analysis

Software Technical Information

The information in this section describes `OpenPathSampling` as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7, 3.6, 3.7, 3.8)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence MIT

Software module developed by David W.H. Swenson

- *Purpose of Module*
 - *The general and reusable approach*
 - *Specific implementations*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

This module adds a general framework for progress meters within the analysis tools of OpenPathSampling, as well as adding progress meters for several specific analysis functions.

Purpose of Module

Some analysis in OpenPathSampling can take a significant amount of time to perform. Previously, users had no feedback on how long such an analysis would take, which can be a frustrating experience.

This module adds a standardized and straightforward approach to progress meters in OPS analysis functions. In addition, it includes these progress meters in several OPS analysis tools.

The general and reusable approach

The primary goals of the OPS progress meter approach are to make it easy for contributors to add progress meters to their analysis tools and to make it flexible with regards to the progress meter used, especially making it easy to silence the output if desired.

Therefore, the approach used here is to create a mix-in class that provides a `progress` property, which wraps around an iterable to provide a progress bar on that iterable. The progress meter was designed to wrap around the widely-used package `tqdm`, and much of the API mimics the `tqdm` API.

The `progress` property can be set with the string `'tqdm'` to use `tqdm` or `'silent'` to not output anything. In addition, it can be further customized by the user by setting it to a function that takes keyword `desc` (a string description) and `leave` (a boolean indicating whether the progress bar should remain after completion) and returns a closure that takes an iterable and yields the elements of that iterable.

In order to use the progress meter in an analysis tool, a developer must simply inherit from the mix-in `openpathsampling.progress.SimpleProgress` and wrap the appropriate iterable with `self.progress`, i.e., a loop for `step` in `steps` becomes for `step` in `self.progress(steps)`.

If `tqdm` is present in the user's environment, the default behavior is to use `tqdm`. If it is not present, the progress meter is silent.

Specific implementations

This has been implemented for several OPS analysis classes. Straightforward implementation have been performed for:

- `ShootingPointAnalysis`
- `PathHistogram`
- `MoveAcceptanceAnalysis`

It has also been implemented as part of the OPS TIS analysis subpackage. This implementation was less straightforward, as progress in this analysis contains nested loops. However, it has been implemented for:

- `MultiEnsembleSamplingAnalyzer`, the base class for many TIS analysis classes
- `StandardTISAnalysis` has significant customized work to use the new progress bars

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

`OpenPathSampling` can be installed with either `pip` or `conda`:

```
pip install openpathsampling
# or
conda install -c conda-forge openpathsampling
```

Tests in `OpenPathSampling` use `pytest`. The requirements for testing are `pytest` and `nose`, both of which can be installed with either `pip` or `conda`.

With the package and its testing tools installed, tests can be run with:

```
py.test --pyargs openpathsampling
```

Examples

This will affect many existing analysis examples. OPS examples can be found:

- In the documentation: <http://openpathsampling.org/latest/examples/index.html>
- On GitHub: <https://github.com/openpathsampling/openpathsampling/tree/master/examples>

Source Code

This module has been merged into `OpenPathSampling`. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/882>
- <https://github.com/openpathsampling/openpathsampling/pull/895>

- <https://github.com/openpathsampling/openpathsampling/pull/902>
- <https://github.com/openpathsampling/openpathsampling/pull/906>

All of the functionality in this module will be included in OpenPathSampling 1.3.

1.3.16 Faster Path Density Analysis in OPS

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7, 3.7, 3.8, 3.9)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence MIT

Software module developed by David W.H. Swenson

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

A previous module introduced a path density analysis in OpenPathSampling. However, the interpolation scheme used in that was rather slow. This module makes it so that users can change interpolation schemes, and introduces two faster options.

Purpose of Module

As discussed in the module *Path Density for OpenPathSampling*, the path density is a useful tool for analyzing mechanisms in transition path sampling. One of the features of the path density, which distinguishes it from a frame-based density, is that it uses interpolation over the trajectory. That is, histogram bins that are traversed are included, even if no snapshot falls in them.

The interpolation approach introduced in the previous module worked by subdividing intervals to find all bins that are crossed. This approach is exact, but slow. This module refactors the path density so that the interpolation algorithm can be provided by the user, and also provides two new (and faster) interpolation approaches:

- **BresenhamInterpolation**: Interpolation using the [Bresenham line drawing algorithm](#)
- **BresenhamLikeInterpolation**: An interpolation scheme similar to Bresenham, but using floats instead of integers.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling use `pytest`.

This module has been included in the OpenPathSampling core as of version 1.1. Its tests can be run by installing `pytest` and OPS, and running the command `py.test --pyargs openpathsampling`.

Examples

An example for path densities can be found at:

- https://github.com/openpathsampling/openpathsampling/blob/master/examples/misc/tutorial_path_histogram.ipynb

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull request:

- <https://github.com/openpathsampling/openpathsampling/pull/875>

1.3.17 SimStore: OPS New Storage Subsystem (part 1)

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (3.6+)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence MIT

- *Purpose of Module*
- *Background Information*
- *Installation and Testing*
- *Examples*

• *Source Code*

Authors: David W.H. Swenson

This module provides the core of SimStore, a new storage subsystem for OpenPathSampling, which is more flexible and has better performance than the older storage system. This module is also necessary to serialize data for transfer over a network, as is needed for parallelization across multiple nodes.

Purpose of Module

OpenPathSampling has the following needs from a storage subsystem:

1. In addition to data objects (data created by the simulation), the simulation objects (i.e., the details of how the simulation was run) should be stored. This helps track provenance and enhances reproducibility.
2. All objects should have universally unique identifiers (UUIDs). References in data objects to the UUIDs of the simulation objects that generated them are important for provenance and reproducibility.
3. Because users may add new simulation objects, storing simulation objects should be very general, and should place minimal burden on the users who create new simulations objects.
4. The results from functions that are calculated during a simulation should be stored in such a way that they can be retrieved again, instead of recalculating them.
5. Path sampling generates large quantities of data. Because of this, and because analysis is frequently done in layers (e.g., one can perform the TIS rate analysis without reloading any coordinate information), it should be possible to reload some information without reloading the entire object.
6. Storage of simulation objects should be (nearly) human readable. While there are some exceptions to human readability, it is important that, for the most part, simulation parameters can be read and interpreted if a user wishes to load OPS data in, e.g., another programming language.

This set of requirements was met by the previous storage subsystem, `netcdfplus`. However, `netcdfplus` was written in a recursive style, which means that every load from disk was a separate request. This made `netcdfplus` very slow. Additionally, the storable function results (#4 above) were written in a way that was not compatible with parallelization. Finally, the base storage class of `netcdfplus` inherits from `netcdf4`, meaning that it was tied to a single backend.

With this module, we introduce SimStore, which is being added as an experimental module in OpenPathSampling, with the intent of replacing `netcdfplus` in OPS 2.0. SimStore will have all the same features, with better performance, more flexibility for users and developers, and a design that is prepared for parallelization. Until version 2.0, both storage subsystems will coexist in the OPS library.

This module, in particular, provides the core storage capabilities (#1, #3, and #6 above) and the proxy-based lazy loading (#5). The UUIDs (#2) and the are still provided by `netcdfplus`. A future module will address the problem of storing function results (#4). Importantly, the API for flexible storage of general simulation objects (#3) remains the same as in `netcdfplus`, facilitating the transition to SimStore.

Some of the specific functionality covered includes:

- **SQL backend:** The first backend for SimStore is SQL, defaulting to `sqlite3`. However, in principle, it should be nearly trivial to use a MySQL or PostgreSQL instance instead, which would be suitable for parallel usage.
- **Schema-based storage:** The description of data objects, which does not vary much for a given application, is provided by a human-readable schema, including specification of what objects should be loaded as lazy proxies. This makes it easy for users or developers to see and understand the overall data model.
- **Dynamic registration of new tables:** Some aspects of the data model do not vary for a certain application (e.g., in OpenPathSampling, trajectories are always lists of snapshots). However, some aspects do depend on the specific use case (e.g., in OpenPathSampling, the size of the coordinates array depends on the specific molecular

system being studied). SimStore allows dynamic registration of tables in order to create new tables of the correct size for, e.g., snapshots coordinates.

- **Extensible JSON-based simulation object serialization:** In order to create a (mostly) human readable description of simulation objects, SimStore (like `netcdfplus`) uses JSON. However, some simulation objects from outside packages (e.g., instances of `simtk.Quantity`, which pair a value with a unit, and are used in OpenMM) require custom serialization. In `netcdfplus`, that custom serialization was inside a `netcdfplus` function, and extension by the user required editing the `netcdfplus` code. SimStore uses a simple registration protocol so that new custom JSON serialization can be provided by the user without digging into the internals.

This module only deals with the generic and reusable aspects of SimStore. Integration of SimStore and OpenPathSampling will be the subject of a future module.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Installation and Testing

This was included in the version 1.4 release of OpenPathSampling. It can be installed via the `conda` package manager with:

```
conda install -c conda-forge openpathsampling
```

In addition to previous OPS requirements, this module requires SQLAlchemy, and other parts of the new storage require Dill. These can be installed with, e.g., `conda install -c conda-forge sqlalchemy dill`.

The tests for this module are split between unit tests included in the OpenPathSampling repository and integration tests in a separate repository. The easiest way to run both sets of tests is to download or clone the integration test repository at <https://github.com/dwhswenson/ops-storage-notebooks>. Install the required testing software, e.g., with:

```
conda install -c conda-forge pytest pytest-cov nbval
```

Then just run the `test-storage.sh` script in that repository. Note: although the module will work with Python 3.6+, some of the notebook tests are not compatible with more recent versions of Python, so the tests should be run with Python 3.7.

Examples

An example for this module can be found at:

- https://github.com/dwhswenson/ops-storage-notebooks/blob/master/examples/02_load_old_cvs.ipynb

Source Code

This module includes the general SimStore components of the pull request at: <https://github.com/openpathsampling/openpathsampling/pull/928>. In particular, this module is for the files in the `openpathsampling.experimental.simstore` subpackage within that pull request.

1.3.18 SimStore: Storable Functions

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (3.6+)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence MIT

- *Purpose of Module*
- *Background Information*
- *Installation and Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module adds “storable functions” to SimStore, the new storage subsystem for OpenPathSampling. Storable functions cache the results of previous calculations to disk. This new implementation will support future parallelization approaches.

Purpose of Module

Trajectory-based methods to study rare events, such as transition path sampling (TPS), frequently require calculation of some collective variables during the simulation. In some cases, these collective variables can be relatively expensive to calculate, and may be calculated hundreds of thousands of times during simulation.

For some types of simulations, such as the one-way shooting variable in TPS, parts of trajectories can be reused, making it advantageous to store the results of collective variables in memory. Furthermore, those same collective variables are frequently used in analysis, making it advantageous to store the results to disk.

This module introduces the parts of SimStore that manage that storage. This includes the `StorableFunction` class itself, which wraps around a user-defined function and handles caching results in memory, and looking up results cached to disk. The user-defined function must take a data object (such as a snapshot or a trajectory), which has a unique universal identifier (UUID), and must return the same value every time it operates on the same input (i.e., it must be a “pure” function).

A `StorableFunction` can be used in different modes: in 'analysis' mode, it first searches the memory cache, then the disk storage, then finally evaluates the internal function. In 'production' mode, it first searches the memory cache, then evaluates the function. Finally, in 'no-caching' mode, it always evaluates the internal function.

One of the challenges in designing the new storable function infrastructure was to ensure that it would be compatible with parallelization. This module includes functionality so that the memory caches from different remote workers can

be returned with the other results, and combined into a master memory cache of the process that also stores results to disk.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Installation and Testing

This was included in the version 1.4 release of OpenPathSampling. It can be installed via the conda package manager with:

```
conda install -c conda-forge openpathsampling
```

In addition to previous OPS requirements, this module requires SQLAlchemy, and other parts of the new storage require Dill. These can be installed with, e.g., `conda install -c conda-forge sqlalchemy dill`.

The tests for this module are split between unit tests included in the OpenPathSampling repository and integration tests in a separate repository. The easiest way to run both sets of tests is to download or clone the integration test repository at <https://github.com/dwhswenson/ops-storage-notebooks>. Install the required testing software, e.g., with:

```
conda install -c conda-forge pytest pytest-cov nbval
```

Then just run the `test-storage.sh` script in that repository. Note: although the module will work with Python 3.6+, some of the notebook tests are not compatible with more recent versions of Python, so the tests should be run with Python 3.7.

Examples

An example for this module can be found at:

- https://github.com/dwhswenson/ops-storage-notebooks/blob/master/examples/02_load_old_cvs.ipynb

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull requests:

- <https://github.com/openpathsampling/openpathsampling/pull/929>
- <https://github.com/openpathsampling/openpathsampling/pull/985>

1.3.19 OPS New Storage Subsystem

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (3.6+)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Installation and Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module interfaces OpenPathSampling with the underlying storage subsystem designed for SimStore.

Purpose of Module

The motivation for this is described in depth in the module *SimStore: OPS New Storage Subsystem (part 1)*. That module focused on the underlying, reusable library SimStore that was developed to meet the storage needs of OpenPathSampling. This module represents the software that integrates the generic code with OPS.

In particular, this includes:

- Design of the OPS serialization schema
- Custom handling of some objects, such as snapshots, for which the dimensionality is only known at runtime.
- Workarounds so that some objects could be stored.
- Custom subclass of the storable functions introduced in *SimStore: Storable Functions* that meet the requirements of OpenPathSampling. For example, when a collective variable is calculated on a trajectory, it should return a list with the value for each snapshot within the trajectory.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Installation and Testing

This was included in the version 1.4 release of OpenPathSampling. It can be installed via the conda package manager with:

```
conda install -c conda-forge openpathsampling
```

In addition to previous OPS requirements, this module requires SQLAlchemy, and other parts of the new storage require Dill. These can be installed with, e.g., `conda install -c conda-forge sqlalchemy dill`.

The tests for this module are split between unit tests included in the OpenPathSampling repository and integration tests in a separate repository. The easiest way to run both sets of tests is to download or clone the integration test repository at <https://github.com/dwhswenson/ops-storage-notebooks>. Install the required testing software, e.g., with:

```
conda install -c conda-forge pytest pytest-cov nbval
```

Then just run the `test-storage.sh` script in that repository. Note: although the module will work with Python 3.6+, some of the notebook tests are not compatible with more recent versions of Python, so the tests should be run with Python 3.7.

Examples

An example for this module can be found at:

- https://github.com/dwhswenson/ops-storage-notebooks/blob/master/examples/02_load_old_cvs.ipynb

Source Code

This module includes the general SimStore components of the pull request at: <https://github.com/openpathsampling/openpathsampling/pull/928>. In particular, this module is for the files in the `openpathsampling.experimental.storage` subpackage within that pull request.

1.3.20 SimStore: Support for OpenMM Snapshots

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (3.7, 3.8, 3.9)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence MIT

Software module developed by David W.H. Swenson

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*

This module adds support for OpenMM snapshots in SimStore, the new storage subsystem used by OpenPathSampling.

Purpose of Module

Previous modules have provided the core of the SimStore storage interface for OPS, as well as integration for OPS simulations using either the internal toy engine or using the Gromacs engine. However, one of the most commonly used engines for OPS is OpenMM. Because OpenMM data carries explicit units, it requires special techniques for storing. Additionally, OpenMM snapshots in OPS are split such that the configurational components can be reused for multiple initial velocities, which also requires special treatment. This module adds those techniques, thus adding support for OpenMM simulations in the new SimStore storage subsystem in OPS. SimStore is faster than the current OPS storage, and is essential for the parallelization of OPS.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling use `pytest`.

This was included in the version 1.4 release of OpenPathSampling. It can be installed via the `conda` package manager with:

```
conda install -c conda-forge openpathsampling
```

In addition to previous OPS requirements, this module requires SQLAlchemy, and other parts of the new storage require Dill. These can be installed with, e.g., `conda install -c conda-forge sqlalchemy dill`.

The tests for this module are split between unit tests included in the OpenPathSampling repository and integration tests in a separate repository. The easiest way to run both sets of tests is to download or clone the integration test repository at <https://github.com/dwhswenson/ops-storage-notebooks>. Install the required testing software, e.g., with:

```
conda install -c conda-forge pytest pytest-cov nbval
```

Then just run the `test-storage.sh` script in that repository. Note: although the module will work with Python 3.6+, some of the notebook tests are not compatible with more recent versions of Python, so the tests should be run with Python 3.7.

Source Code

This module has been merged into OpenPathSampling. It is composed of the following pull request:

- <https://github.com/openpathsampling/openpathsampling/pull/949>

The modules that are based on OPS, but remain separate, are:

1.3.21 Annotated Trajectories

Software Technical Information

This module is based on OpenPathSampling. This section includes information both for the specific module and for OpenPathSampling as a whole.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material http://openpathsampling.org/latest/examples/annotated_trajectories/tree/master/examples <https://github.com/dwhswenson/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module provides a data structure that adds annotations to frames of a trajectory, intended to label those frames as being, for example, in a given metastable state. It also provides some tools to analysis whether a proposed state definition is compatible with those annotations.

Purpose of Module

When dealing with biomolecular systems, one of the common challenges is to define the (meta)stable states. The existence of metastable states is easily determined by visually inspecting the trajectory. However, identifying geometric criteria to characterize the states remains difficult.

This module provides a data structure that allows the user to easily annotate a trajectory with the visually identified states, and to compare those annotations to proposed state definitions. It also includes tools to visualize where the proposed state definition matches the annotations.

This implementation includes:

- An `Annotation` class, which is essentially a structure to connect the state label and a range of frames (marked with their beginning and ending frames) that the user identifies as in the given state.
- Another data structure, `ValidationResults`, which contains the correctly identified frames, as well as false positives and false negatives, for a given proposed state definition.
- The `AnnotatedTrajectory` class, which associates the annotations with an `OpenPathSampling Trajectory` object and performs the analysis to compare the proposed states to those annotations.
- A method `plot_annotated`, which plots the annotations and the proposed state definition in order to visually inspect the quality of the proposed state.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling use the `nose` package.

The tests for this module can be run by downloading its source code, installing its requirements (namely, OpenPathSampling), and running the command `nosetests` from the root directory of the repository.

Once the requirements are installed, a standard installation of this package can be done with `python setup.py install`.

Examples

The features in this code, including the ability to save the annotations associated with a trajectory, are highlighted in a Jupyter notebook in its `examples/` directory. It can be viewed [here](#).

Source Code

This module is for the 0.1 release of `annotated_trajectories`. The source code for this module can be found in: https://github.com/dwhswenson/annotated_trajectories/releases/tag/v0.1.0

1.3.22 OPS Piggybacker (legacy file converter)

Software Technical Information

This module is based on OpenPathSampling. This section includes information both for the specific module and for OpenPathSampling as a whole.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation

- <http://openpathsampling.org>
- <http://opsiggybacker.readthedocs.io> (in progress)

Installation instructions

- OpenPathSampling: <http://openpathsampling.org/latest/install.html>
- OPSiggybacker: <https://github.com/dwhswenson/OPSPiggybacker>

Relevant Training Material

- <http://openpathsampling.org/latest/examples/>
- <https://github.com/dwhswenson/OPSPiggybacker/tree/master/examples>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: David W.H. Swenson

This module provides a library for converting path sampling simulations from legacy codes into a format that can be analyzed by OpenPathSampling. The implementation in this module works with flexible-length transition path sampling (TPS) with one-way shooting and uniform shooting point selection.

Purpose of Module

OpenPathSampling contains excellent tools for analyzing simulations, as well as excellent tools for sampling. However, since OpenPathSampling is a new software package, users may already have simulations that they have run with other packages. The purpose of this module is to provide tools that allow the user to easily convert legacy script output into a format that can be analyzed by OpenPathSampling. It is a library of tools that can be used with data from any existing path sampling simulation output, and specifically includes tools to simplify the use of flexible-length TPS with one-way shooting and uniform shooting point selection. Extending to the library to work with other simulation types will be part of future work.

The OPSPiggybacker essentially fakes a simulation, based on data from another source. In this module, it has the ability to read in data from one-way TPS. The user must create the appropriate OPS `TransitionNetwork` object (including defining the correct collective variables and state volumes). Then the code creates a `MoveScheme`, but instead of actually running the simulation, it reads in the results of an existing simulation and provides the same output that the `MoveScheme` *would* have provided. We call this a “pseudo-simulation.”

Classes implemented in this module include:

- `ShootingPseudoSimulator`, subclass of `openpathsampling.PathSimulator`. This acts like the OPS simulator, and runs the pseudo-simulation. Instead of taking an integer saying how many steps to run, it takes a list of data that describes each shooting move.
- `ShootingStub`, subclass of `openpathsampling.pathmovers.PathMover`. This acts like the `openpathsampling.OneWayShootingMover`. It reads in the data and creates the appropriate output that can be analyzed by OPS.
- `OneWayTPSConverter`, subclass of `ShootingPseudoSimulator`. This pseudo-simulator is designed to read a specific type of input file, which can be prepared based on the output from legacy simulation tools. Depending on the nature of the input trajectory files, several options can be set to ensure that the resulting OPS trajectories are correct. This is an abstract superclass, subclasses must define how to read trajectory files of the appropriate format.
- `GromacsOneWayTPSConverter`, subclass of `OneWayTPSConverter`. Specialized for reading in GRO-MACS files (using `MDTraj`).

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests use the `nose` package.

To test this module, first install its requirements (namely OpenPathSampling). Next, download OPSPiggybacker code, either by git or by downloading the `.tar` or `.zip` of this release and decompressing it. Change into the root directory of the OPSPiggybacker, and run `python ops_piggybacker/tests/common_test_data.py` to prepare some test data. After that, use the `nosetests` command to run the actual tests (this can be done from the same directory).

Installation of this package can be performed with `python setup.py install`. Installation can be done before or after testing.

Examples

Several examples are in the `examples/` directory of the code:

- Using the `ShootingPseudoSimulator` with partial (one-way) trajectories
- Using the `ShootingPseudoSimulator` with full (pre-joined) trajectories

Source Code

The module is for the 0.1 release of the OPSPiggybacker project. This includes all the work on that project through pull request #15 (merged on 28 December, 2016).

The source code for this module can be found in: <https://github.com/dwhswenson/OPSPiggybacker/releases/tag/v0.1.0>

1.3.23 Contact Map

Software Technical Information

Language Python (2.7, 3.4, 3.5, 3.6)

Licence LGPL 2.1 or later

Documentation Tool Sphinx/RST

Application Documentation <http://contact-map.readthedocs.io/>

Relevant Training Material <http://contact-map.readthedocs.io/en/latest/examples.html>

- *Purpose of Module*

- *Background Information*
- *Testing*
- *Source Code*

Frequently, we characterize states (especially in biomolecular systems) in terms of the contacts between specific residues or atoms. When trying to identify the specific contacts of interest, it can be useful to look at all the contacts. This module provides tools for mapping and identifying contacts in trajectories.

Purpose of Module

Contacts can be an important tool for defining (meta)stable states in processes involving biomolecules. For example, an analysis of contacts can be particularly useful when defining bound states during a binding processes between proteins, DNA, and small molecules (such as potential drugs).

The contacts analyzed by `contact_map` can be either intermolecular or intramolecular, and can be analyzed on a residue-residue basis or an atom-atom basis.

This package makes it very easy to answer questions like:

- What contacts are present in a trajectory?
- Which contacts are most common in a trajectory?
- What is the difference between the frequency of contacts in one trajectory and another? (Or with a specific frame, such as a PDB entry.)
- For a particular residue-residue contact pair of interest, which atoms are most frequently in contact?

It also facilitates visualization of the contact matrix, with colors representing the fraction of trajectory time that the contact was present.

Background Information

This is an independent module, but it builds on tools developed in [MDTraj](#).

Testing

This module can be installed with conda, using `conda install -c conda-forge contact_map`. To install the specific version associated with this module, use `conda install -c conda-forge contact_map==0.2.0`

Tests for this module can be run with pytest. Install pytest with `pip install pytest` and then run the command `py.test` from within the directory with the source code, or `py.test --pyargs contact_map` from anywhere after installation.

Source Code

The source code for this module, and modules that build on it, is hosted at https://github.com/dwhswenson/contact_map. This module specifically includes everything up to and including [release 0.2](#).

Software Technical Information

Language Python (2.7, 3.4, 3.5, 3.6)

Licence LGPL 2.1 or later

Documentation Tool Sphinx/RST

Application Documentation <http://contact-map.readthedocs.io/>

Relevant Training Material <http://contact-map.readthedocs.io/en/latest/examples.html>

Software Module Developed by David W.H. Swenson

1.3.24 Contact Map Parallelization

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module adds the ability to parallelize the calculation of contact frequencies (see the `contact_map` module). It includes improvements to the core of the `contact_map` package to facilitate parallelization, as well as integration with a framework for practical parallelization.

Purpose of Module

Contacts are defined as when two atoms, or atoms within two groups of atoms (residues), are within some cutoff distance of each other. The contact map is the set of all contacts in a given snapshot. The contact frequency is the fraction of a trajectory in which each pair of contacts is present. The contact frequency therefore requires calculation of the contact map for each individual frame in the trajectory.

The original `contact_map` code included OpenMP (shared-memory) parallelization of the calculation of a single contact map (a loop over atoms). Each contact map in a contact frequency (the loop over the frames of a trajectory) was done sequentially. However, each frame is completely independent, and can be processed on a separate node. This module implements that parallelization.

This module interfaces with the `dask.distributed` package for task-based parallelization. The trajectory is separated into segments, with the `dask` network calculating the contact frequency of each segment in parallel (reading from a common file source). Then the partial contact frequencies are combined into one `ContactFrequency` object. This also includes methods, such as serialization into JSON strings, that would be useful for parallelization by other tools.

Background Information

This is part of the `contact_map` package, which in turn builds on tools in `MDTraj`.

The parallelization is based on `dask.distributed`. See its docs for details on setting up a `dask` scheduler/worker network.

Building and Testing

The `contact_map` package can be installed with conda, using `conda install -c conda-forge contact_map`. This module is included in version 0.3.0, which can be specifically installed with `conda install -c conda-forge contact_map==0.3.0`.

`dask.distributed` must be installed separately, which can be done with `conda install -c conda-forge dask distributed`.

Tests for this module can be run with `pytest`. Install `pytest` with `pip install pytest` and then run the command `py.test` from within the directory with the source code, or `py.test --pyargs contact_map` from anywhere after installation. Tests specific to integration with `dask.distributed` will be marked as “skipped” if that framework is not installed.

Source Code

This module is composed of the following pull requests in the `contact_map` repository:

- https://github.com/dwhswenson/contact_map/pull/3
- https://github.com/dwhswenson/contact_map/pull/29
- https://github.com/dwhswenson/contact_map/pull/30

Software Technical Information

Name OpenMMTools

Language Python (3.6, 3.7)

Licence MIT

Documentation Tool Sphinx

Application Documentation <http://openmmtools.readthedocs.org>

Relevant Training Material <http://openmmtools.readthedocs.org>

Software Module Developed by David W.H. Swenson

1.3.25 Double-Well Dimer Testsystems

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Examples*
- *Source Code*

One of the common systems used to study rare events is the double-well dimer in a bath of repulsive particles. In this system, two particles are linked by a “bond” that allows condensed and extended metastable states. This module adds this system, and tools for created several variants of it, to the `OpenMMTools` package.

Purpose of Module

The symmetric double-well dimer is a widely-used model for developing new rare events methodologies. However, implementing simple models in software packages that are designed for biological systems, such as OpenMM, can be difficult for a novice user. As a result, many developers of new methods will implement their methods twice: first to interface with simple models such as the double-well dimer using in-house MD codes, then a second time to interface with more powerful tools, such as OpenMM, to simulate complex systems such as biomolecules. This module provides tools that facilitate setting up custom versions of the double-well dimer for OpenMM, allowing users to develop their new methodologies directly for the same platform that they will use for larger practical applications.

The widely-used double-well dimer model is a symmetric quartic potential, given by:

$$V_{dw}(r) = h \left(1 - \left(\frac{r - r_0 - w}{w} \right)^2 \right)^2$$

where r is the distance between the particles, h is the height of the barrier, r_0 is the energy minimum for the condensed metastable state, and w sets the distance for the extended metastable state according to $r_{ex} = r_0 + 2w$.

This “bonded” interaction is added for specific pairs of particles, on top of a background of WCA (purely repulsive) “nonbonded” interactions between all particles. The WCA interaction is:

$$V_{WCA}(r) = \begin{cases} 4\epsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right) + \epsilon & \text{if } r \leq 2^{1/6}\sigma \\ 0 & \text{if } r > 2^{1/6}\sigma \end{cases}$$

where σ is a characteristic distance and ϵ is a characteristic energy scale.

The quartic double well is a simple model of rare events, where the expected reaction coordinate is obvious. However, it is can be very useful for benchmarking new methods. The OpenMMTools package includes a suite of systems to be used in testing and benchmarking, and was a natural place to add these.

Although the most widely-used approach has been to have a single dimer in the bath of WCA particles, this module provides two possible extensions that have been previously used in the literature. The first extension is to allow multiple independent dimers, as was done in [Swenson2014]. This is done by changing the `ndimers` parameter in the `DoubleWellDimer_WCAFluid` test system. The other extension is to create a polymer chain of double-well bonds, as was done in [Rogal2008]. This is done by changing the `nchained` parameter in the `DoubleWellChain_WCAFluid` test system.

Background Information

This builds on the `testsystems` module of OpenMMTools. The OpenMMTools source is hosted at <http://github.com/choderalab/openmmtools>. These contributions will be included in OpenMMTools 0.17.0.

Building and Testing

This has been incorporated into OpenMMTools 0.17. Up-to-date installation information can be found in the OpenMMTools documentation; as of this writing it simply requires installing `conda`, and then using the command `conda install -c conda-forge -c omnia openmmtools`. Note that this requires a Python 3-based environment; OpenMMTools does not support Python 2.

Extra requirements for the tests can be installed with `conda install -c conda-forge -c omnia nose nose-timer pymbar`. The full suite of tests can be run from the `openmmtools` directory with the command `nosetests`. The tests specific to this (and other test systems) can be run from the root directory of the repository with: `nosetests openmmtools/tests/test_testsystem.py`.

Examples

Examples of the tools in this module can be seen in a Jupyter notebook that can be viewed or downloaded from a GitHub gist at: <https://gist.github.com/dwhswenson/bb79a137a1d65629c22e7b00aa569d76>

Source Code

The source for this module was contributed to OpenMMTools. The relevant pull request is:

- <http://github.com/choderalab/openmmtools/pull/389>

1.3.26 Integrating LAMMPS with OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Examples*
- *Testing and Performance*
- *Source Code*

Authors: Jan-Hendrick Prinz and Jony Castagna

This module shows how LAMMPS can be used as Molecular Dynamic (MD) engine in OpenPathSampling (OPS) and it also provide a benchmark for the impact of OPS overhead over the MD engine.

Purpose of Module

OpenPathSampling uses OpenMM as default engine for calculating the sampled trajectories. Other engines as GRO-MACS and LAMMPS can be used (despite not yet available in the official release) allowing to exploit different computer architectures like hybrid CPU-GPU and to simulate more complex problems.

In general OPS gathers a frame (i.e. a state of the physical system at a point in time, typically consists of coordinates, velocities, and periodic cell vectors) after a defined number of time steps. The MD engine has to produce the sequence of frames and wait for OPS to provide new input values. This generate of course an overheads which has a negative impact on the overall performance of the simulation.

In this module we present the source code for the integration of OPS with LAMMPS as well as a benchmark for of a simple test case to show the impact on the performance due to OPS overhead.

The integration with LAMMPS has been developed by Jan-Hendrick Prinz (<https://github.com/jhprinz>) and consists of a Python script where the number of time steps per frame has to be specified (see below for the link to the source file).

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Details about how to use OPS with LAMMPS are provided in the:

- iPython notebook for LAMMPS: https://github.com/jhprinz/openpathsampling/blob/1235c472217d32b26011cdd6db0ac6287c994ab2/examples/misc/introduction_lammps.ipynb

Examples

The script which integrate LAMMPS with OPS can be applied to any case running in LAMMPS. For example, the Lennard-Jones test (32K atoms) case presented in the ECAM deliverable D7.2 has been used to benchmark the OPS overhead when using LAMMPS as presented in next section.

Testing and Performance

The table shows the performance of LAMMPS with OPS for the Lennard-Jones (32K atoms) test case using 100 time steps per frame (more frequent queries) and 1000 time steps (less frequent queries) for a total of 100K time steps. The MD engine time (i.e., LAMMPS only time) and the total time (OPS + LAMMPS) using different number of nodes (with 24 cores per node) is presented.

Results have been obtained using the JURECA (http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html) supercomputer.

| Steps between frames | Nodes | Md Engine Time [s] | Total time [s] |
|----------------------|-------|--------------------|----------------|
| 100 | 1 | 106.25 | 108.48 |
| 100 | 2 | 61.40 | 62.31 |
| 100 | 4 | 36.46 | 37.28 |
| 1000 | 1 | 100.93 | 102.48 |
| 1000 | 2 | 53.56 | 54.45 |
| 1000 | 4 | 29.71 | 30.67 |

Using 100 or 1000 time steps per frame, the overhead due to the OPS is within a maximum of 3%. However, one should note that when increasing the number of time steps per frame to 1000, the time spent in the MD engine decreases due to less overhead from stopping and starting the engine. A suggested improvement to OPS has been to allow the engine to continue the trajectory while the frame is being evaluated by OPS, which should help eliminate this overhead. The OPS overhead remains relatively static and there is little discernible difference between the overheads for the two measurements. Given that OPS is effectively a serialisation point for the calculation, more intensive trajectories should also, therefore, lead to improved scalability results since they will reduce this ratio of serial to parallel workload.

Source Code

The source code for integrating LAMMPS with OPS can be found at:

- <https://github.com/openpathsampling/openpathsampling/pull/697>

1.3.27 OpenPathSampling CLI Core

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (3.6+)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling-cli.readthedocs.io/>

Relevant Training Material <http://openpathsampling.org/>

Licence MIT

Software module developed by David W.H. Swenson

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

This module provides the core infrastructure for the OpenPathSampling command line interface.

Purpose of Module

OpenPathSampling (OPS) is a powerful Python library for path sampling simulations. However, using it requires expertise in Python programming. In order to make OPS more accessible to a broader audience, we have begun development of a command line interface (CLI) to work with OPS files and to run OPS simulations.

The usage model is that there is a single command, `openpathsampling`, which takes subcommands and delegates them to different routines. This is the same model as, e.g., `git`, which has subcommands like `git commit` and `git clone`.

This module provides the underlying infrastructure/platform that the specific CLI subcommands will be built upon. A subsequent module will provide several specific subcommands.

Key functionality included here:

- **Parameter decorators:** Different subcommands will frequently use the same options and arguments. In order to maintain consistency in parameter usage and help statements, common parameters have been implemented as a set of reusable decorators.

- **Plug-in infrastructure:** To promote customizability, the OPS CLI uses a plug-in infrastructure. This allows us to ship a small set of default subcommands, but to allow users to easily create and share custom subcommands for their own projects or workflows. This module introduces two mechanisms by which a plug-in can register with the CLI. One is based on [native namespace packages](#), which makes it easy for developers to create and distribute plug-ins that automatically register with OPS when installed with standard Python tools. The other approach is file-based, and simply involves placing the plug-in Python file in the user's `~/ .openpathsampling/cli-plugins/` directory. This approach is useful for quick development of plug-ins that may be intended to be shared within a specific research group or used to simplify reproducibility of workflows for a single project, but are not intended to be widely distributed and maintained for the long term.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Detailed documentation on the CLI can be found at <https://openpathsampling-cli.readthedocs.io/>

Testing

The OPS CLI can be installed with either pip or conda:

```
pip install openpathsampling-cli
# or
conda install -c conda-forge openpathsampling-cli
```

Tests in the OpenPathSampling CLI use [pytest](#). The requirements for testing are `pytest` and `nose`, both of which can also be installed with either `pip` or `conda`.

With the package and its testing tools installed, tests can be run with:

```
py.test --pyargs paths_cli
```

Examples

This module deals with the underlying platform on which the rest of the OPS CLI is built. As such, there are no direct examples. However, there are examples of how to use the platform, i.e., how to write plugins. Some of these can be found in the `example_plugins` directory. Additionally, the specific commands implemented for the OPS CLI 0.1, which will be the subject of a second module, can be thought of as examples.

Source Code

The source code for the OpenPathSampling CLI can be found in the its GitHub repository: <http://github.com/openpathsampling/openpathsampling-cli>.

This module covers the “core” code for the OpenPathSampling CLI as of release version 0.1.

Specifically, it includes the following files, and their associated test suites:

- `cli.py`
- `param_core.py`

- `parameters.py`
- `plugin_management.py`

1.3.28 OpenPathSampling CLI Commands

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (3.6+)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling-cli.readthedocs.io/>

Relevant Training Material <http://openpathsampling.org/>

Licence MIT

Software module developed by David W.H. Swenson

- *Purpose of Module*
 - *Simulation commands*
 - *Commands for dealing with files*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

This module provides the commands available to the OpenPathSampling command line interface as of version 0.1.

Purpose of Module

As discussed in the *OpenPathSampling CLI Core* module, a command line interface will make path sampling easier for a broader range of users. The OPS CLI is initially designed to support a few functions that will be most useful to users. More commands will be added over time, but the initial commands are outlined below.

Simulation commands

- `visit-all`: create initial trajectories by running MD until all states have been visited (works for MSTIS or any 2-state system); must provide states, engine, and initial snapshot on command line
- `equilibrate`: run equilibration for path sampling (until first decorrelated trajectory); must provide move scheme and initial conditions on the command line
- `pathsampling`: run path sampling with a given move scheme (suitable for custom TPS schemes as well as TIS/RETIS); must provide move scheme, initial conditions, and number of MC steps on command line

Commands for dealing with files

- `contents`: list all the named objects in an OPS storage, organized by store (type); this is extremely useful to get the name of an object to use
- `append`: add an object from one OPS storage into another one; this is useful for getting everything into a single file before running a simulation

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Detailed documentation on the CLI can be found at <https://openpathsampling-cli.readthedocs.io/>

Testing

The OPS CLI can be installed with either pip or conda:

```
pip install openpathsampling-cli
# or
conda install openpathsampling-cli
```

Tests in the OpenPathSampling CLI use `pytest`. The requirements for testing are `pytest` and `nose`, both of which can also be installed with either pip or conda.

With the package and its testing tools installed, tests can be run with:

```
py.test --pyargs paths_cli
```

Examples

Basic examples of how to use the OPS CLI can be found in:

- <http://openpathsampling.org/latest/cli.html>
- https://gitlab.e-cam2020.eu:10443/dwhswenson/ops_tutorial/-/blob/master/5_advanced_customize_shooting.ipynb

Source Code

The source code for the OpenPathSampling CLI can be found in the its GitHub repository: <http://github.com/openpathsampling/openpathsampling-cli>.

This module covers the code for commands included in the OpenPathSampling CLI as of release version 0.1.

Specifically, it includes the following files, and their associated test suites:

- `commands/append.py`
- `commands/contents.py`
- `commands/equilibrate.py`

- `commands/pathsampling.py`
- `commands/visit_all.py`

Nine of these modules were part of [E-CAM Deliverable 1.2](#). Those modules provided improvements and new features in software for trajectory sampling and for studying the thermodynamics and kinetics of rare events.

1.4 Machine Learning Potentials

Many systems in computational physics and chemistry can be successfully studied with empirical force fields at the atomistic level. In the context of these “molecular mechanics” models, atoms are treated as particles without internal structure and their interactions are defined via rather simple expressions deduced from physical/chemical intuition. Usually a small number of free parameters is enough to tune the potential to reproduce experimental properties with good agreement. However, there are systems for which a satisfying description within this framework is not possible. Take as an example the formation and breaking of covalent bonds. This is the territory of *ab initio* methods which use quantum mechanics to accurately model the behavior of the system. Unfortunately the additional level of detail comes at a cost. Even in small systems *ab initio* methods are usually many orders of magnitude slower than empirical force fields. Moreover, the computational cost increases unfavorably with the number of atoms which makes it impractical to perform large simulations.

With rising influence of machine learning algorithms in science and technology a new category of interatomic potentials has emerged. Machine learning potentials (MLPs) aim at bridging the gap between *ab initio* methods and empirical force fields. In contrast to the latter, MLPs are not bound by a predetermined fixed functional form of the interaction but rather build on the flexibility of an underlying machine learning model, such as artificial neural networks. These are known for their capability to reproduce any complicated function, which in this case is the desired potential energy surface, but rely on a separate training stage before they are ready for use. During this phase the MLP “learns” from a large data set how energies and forces depend on atomic positions. The reference energy landscape is typically computed from expensive *ab initio* methods. Once the training is completed the MLP can accurately predict energies and forces for new (unseen during training) atomic configurations at a fraction of the cost of the reference method. Hence, with MLPs times scales become accessible in molecular dynamics simulations close to those of empirical potentials while maintaining the *ab initio* level of accuracy.

Today MLPs exist in various forms and combine different atomic environment descriptors as inputs for all kinds of machine learning models.

- [Overview of machine learning potentials](#)

A very successful variant is the high-dimensional neural network potential (HDNNP) which combines make use of artificial neural networks to predict atomic energy contributions:

- [Original publication introducing HDNNPs by Behler and Parrinello](#)
- [Descriptors used in HDNNPs: Atom-centered symmetry functions](#)
- [About the construction of HDNNPs](#)

1.5 n2p2

The software `n2p2` (NeuralNetworkPotentialPackage) implements the HDNNP method in a C++ library and applications for training and prediction. Its most important feature in the HPC context is the interface to the popular molecular dynamics package `LAMMPS` which allows to use HDNNPs in massively parallelized simulation runs. Further information can be found in these two publications:

- [n2p2 design and LAMMPS parallel performance](#)
- [Parallel training implemented in n2p2](#)

The following modules extend the functionality of *n2p2*, some are already merged into the [main repository](#), others will also work independently and will be integrated in the future:

Software Technical Information

The information in this section describes *n2p2* as a whole. Information specific to the additions in this module are in subsequent sections.

Name *n2p2* (NeuralNetworkPotentialPackage)

Language C++, Python (3.6+), Jupyter notebook

Licence GPL-3.0-or-later

Documentation Tool Doxygen, Sphinx

Application Documentation <http://compphysvienna.github.io/n2p2/>

Relevant Training Material <http://compphysvienna.github.io/n2p2/>

Software Module Developed by Andreas Singraber

1.5.1 *n2p2* - CG descriptor analysis

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module adds tools to the *n2p2* package which allow to assess the quality of atomic environment descriptors. This is particularly useful when designing a neural network potential based coarse-grained model (NNP-CG).

Purpose of Module

Creating a coarse-grained (CG) model from the full description of a system is a two-step process: (1) selecting a reduced set of degrees of freedom and (2) defining interactions depending on these coarse-grained variables. For example, in a common coarse-graining approach for molecular systems the atomistic picture is replaced by a simpler description with CG particles sitting at the center-of-mass coordinates of the actual molecules. The corresponding interactions between CG sites can be modelled with empirical force fields but also, as has been recently shown in¹ and², with machine learning potentials. To simplify the construction of NNP based coarse-grained models in *n2p2*, this module adds software to estimate the quality of atomic environment descriptors, which in turn hints on the expected performance of the coarse-grained description.

The overall goal of the descriptor analysis is to show qualitatively whether there is a correlation between the raw atomic environment descriptors (and their derivatives) and the atomic forces. If no or very little correlation can be found we can assume that the descriptors do not encode enough information to construct a (free) energy landscape. On the other hand, if “similar” descriptors correspond to “similar” forces there is a good chance that a machine learning algorithm is capable of detecting this link and a machine learning potential can be fitted. In order to find a possible

¹ Zhang, L.; Han, J.; Wang, H.; Car, R.; E, W. DeePCG: Constructing Coarse-Grained Models via Deep Neural Networks. *J. Chem. Phys.* 2018, 149 (3), 034101.

² John, S. T.; Csányi, G. Many-Body Coarse-Grained Interactions Using Gaussian Approximation Potentials. *J. Phys. Chem. B* 2017, 121 (48), 10934–10949.

correlation between descriptors and forces the following approach is used: First, a clustering algorithm (k-means or HDBSCAN) searches for groups in the high-dimensional descriptor space of all atoms. Then, for every detected cluster the statistical distribution of the corresponding atomic forces is compared to the statistics of all remaining atomic forces. A hypothesis test (Welch's t-test) is applied to decide whether the link between descriptors and forces is statistically significant. The percentage of clusters which show a clear link is then an indicator for a good descriptor-force correlation.

In order to perform the analysis described above *n2p2* was extended by two software pieces:

1. **A new application based on the C++ libraries:** `nnp-atomenv`

This application allows to generate files containing the atomic environment data required for the cluster analysis.

2. **A new Jupyter notebook with the actual analysis:** `analyze-descriptors.ipynb`

The script depends on common Python libraries (*numpy*, *scipy*, *scikit-learn*) and reads in data provided by `nnp-atomenv`. It then clusters the data, performs statistical tests and presents graphical results.

Background Information

This module is based on *n2p2*, a C++ code for generation and application of neural network potentials used in molecular dynamics simulations. The source code and documentation are located here:

- *n2p2* documentation: <http://compphysvienna.github.io/n2p2/>
- *n2p2* source code: <http://github.com/CompPhysVienna/n2p2>

Building and Testing

The code changes from this module are already merged with the main *n2p2* repository (see the section below for corresponding pull requests).

Note: By the time of reading these instructions *n2p2* was most likely developed further. To recall the state of the software at the time of writing these instructions please use these commands:

```
git clone https://github.com/CompPhysVienna/n2p2
cd n2p2
git checkout 3cfe391377d2792ac29baf8394b3dce712afdad2
```

To build the new tool `nnp-atomenv` the usual *n2p2* build instructions apply:

```
cd src
make nnp-atomenv -j
```

The `analyze-descriptors.ipynb` Jupyter notebook requires some Python packages to be installed:

- `numpy`
- `scipy`
- `matplotlib`
- `seaborn`
- `scikit-learn`
- `hdbscan`
- `pickle`

Step-by-step instructions on how the descriptor analysis is prepared and performed is available at [this dedicated documentation page](#)

Regression testing is used in *n2p2* automatically for each commit to the main repository. This module also adds the corresponding tests for the `nnp-atomenv` tool in `test/cpp/`. The build log showing the correct run of tests is available [here](#).

Source Code

The new functionality introduced by this module is collected in two pull requests:

- [New tool for symmetry function quality analysis](#)
- [Complete coarse-graining/descriptor analysis documentation](#)

The easiest way to view the source code changes is to use the *Files changed* tab in the above pull request pages.

Software Technical Information

The information in this section describes *n2p2* and *LAMMPS* as a whole. Information specific to the additions in this module are in subsequent sections.

Name *n2p2*, LAMMPS

Language C++

Licence GPL-3.0-or-later (*n2p2*), GPL-2.0 (LAMMPS)

Documentation Tool Doxygen, Sphinx

Application Documentation <http://compphysvienna.github.io/n2p2/> (n2p2) <https://lammps.sandia.gov/> (LAMMPS)

Relevant Training Material <http://compphysvienna.github.io/n2p2/> (n2p2) <https://lammps.sandia.gov/> (LAMMPS)

Software Module Developed by Andreas Singraber

1.5.2 n2p2 - Improved link to HPC MD software

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
 - *LAMMPS user package*
 - *CabanaMD interface*
- *Source Code*
 - *Changes in LAMMPS*
 - *Changes in n2p2*

This module documents efforts to improve the interaction of *n2p2* with existing HPC software, in particular the molecular dynamics (MD) software package *LAMMPS*.

Purpose of Module

Although *n2p2* was already shipped with source files for patching LAMMPS before, the build process required manual intervention of users. To avoid this in future versions of *LAMMPS* a pull request was created to include the *n2p2/LAMMPS* interface by default as a [user package](#). In order to conform with *LAMMPS* [contribution guidelines](#) multiple issues were resolved, triggering these changes/additions to *LAMMPS* and *n2p2*:

- Modify the traditional build process (via makefiles) to include *n2p2*
- Modify the CMake build process to search and include *n2p2*
- Create additional documentation about the build settings
- Adapt documentation of the LAMMPS `pair_style nnp` command
- Create a suitable example which can be shipped with LAMMPS
- Change *n2p2* to conform with *LAMMPS* bigbig settings (see [here](#))
- Change the source files `pair_nnp.(cpp/h)` to conform with the *LAMMPS* coding style

Furthermore, the *n2p2* build system was adapted to allow for multiple interfaces to other software packages, with an option to select only those of interest to the user. This can be achieved by providing the `INTERFACES` variable in the build stage, e.g. use

```
make libnnpif INTERFACES="LAMMPS CabanaMD"
```

to build both the LAMMPS and the CabanaMD interface and include it in the `libnnpif` library.

As a first application, the user contributed [CabanaMD interface](#) was integrated in the new build process. *CabanaMD* is an [ECP proxy application](#) which makes use of the [Kokkos](#) performance portability library and *n2p2* to port neural network potentials in MD simulations to GPUs and other HPC hardware.

Background Information

This module is based on *n2p2*, a C++ code for generation and application of neural network potentials used in molecular dynamics simulations. The source code and documentation are located here:

- *n2p2* documentation: <http://compphysvienna.github.io/n2p2/>
- *n2p2* source code: <http://github.com/CompPhysVienna/n2p2>

In addition the source files for the LAMMPS patch are based on *LAMMPS*, a C++ code for massively parallelized molecular dynamics simulations. The source code and documentation are located here:

- *LAMMPS* documentation: <https://lammps.sandia.gov/>
- *LAMMPS* source code: <http://github.com/lammps/lammps>

Building and Testing

Important: By the time of reading these instructions the packages were most likely developed further and it cannot be guaranteed that the procedures given below will give the desired results. To retrieve the state of each software at the time of writing these lines please uncomment and use the lines with `git checkout <commit-hash>`.

LAMMPS user package

To test whether the *LAMMPS* user package `USER-NNP` works together with *n2p2* as expected we have to download *LAMMPS* from the `pair-style-nnp` feature branch and use the current master version of *n2p2*. First, to get and compile *n2p2*:

```
git clone https://github.com/CompPhysVienna/n2p2
cd n2p2/src
# git checkout 428db3ee61f9943feaedfaaeb5e096289983d46
make libnnpif -j
cd ../../
```

Next we retrieve the *LAMMPS* feature branch:

```
git clone -b pair-style-nnp --single-branch https://github.com/singraber/lammps
cd lammps
# git checkout ed53e2bbff2465dd05ba015a05843b2bb328360c
```

and compile the code with the `USER-NNP` package enabled using the CMake build approach:

```
mkdir build
cd build
cmake -D PKG_USER-NNP=yes -D N2P2_DIR=<path-to-n2p2> ../cmake
make -j
```

Alternatively, we could also use the traditional build process using makefiles:

```
cd src
make yes-user-nnp
make N2P2_DIR=<path-to-n2p2> mpi -j
```

In either case the *LAMMPS* binary should be created (`lammps/build/lmp` or `lammps/src/lmp_mpi`) and we can test if it works correctly with the provided example:

```
cd ../examples/USER/nnp
# Binary from CMake build process:
mpirun -np 4 ../../../../build/lmp -in in.nnp
# or from the traditional build process:
# mpirun -np 4 ../../../../src/lmp_mpi -in in.nnp
```

CabanaMD interface

While the *n2p2* build process for the *CabanaMD* interface is trivial (it requires only the collection of some header files) the compilation steps on the *CabanaMD* side are not trivial. Furthermore, testing requires a suitable GPU with a compatible compiler environment. Hence it is not feasible to provide general build instructions for testing here. However, the *n2p2* documentation offers an example build procedure for a specific hardware setup [here](#).

Source Code

Changes in LAMMPS

The easiest way to view the source code changes in LAMMPS covered by this module is to use the [GitHub pull request](#) page. There, use the *Files changed* tab to review all changes.

Changes in n2p2

The following commits collect all changes required to follow the *LAMMPS* contribution guidelines:

- Updated makefiles for new LAMMPS build process
- Add flag information when n2p2 runs
- Changed build flags prefix from NNP_ to N2P2_
- Changed Atom::(Neighbor::)tag to int64_t

The commits which restructured the makefiles to allow multiple selectable interface library parts can be found here (they are part of the CabanaMD pull request):

- Restructured interface library
- Renamed source files and updated docs

The *CabanaMD* example build instructions were added to the *n2p2* documentation in this commit:

- Add CabanaMD build example docs for reference

Software Technical Information

The information in this section describes *n2p2* as a whole. Information specific to the additions in this module are in subsequent sections.

Name n2p2 (NeuralNetworkPotentialPackage)

Language C++

Licence GPL-3.0-or-later

Documentation Tool Doxygen, Sphinx

Application Documentation <http://compphysvienna.github.io/n2p2/>

Relevant Training Material <http://compphysvienna.github.io/n2p2/>

Software Module Developed by Martin P. Bircher and Andreas Singraber

1.5.3 n2p2 - Polynomial Symmetry Functions

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module introduces a new set of atomic environment descriptors for high-dimensional neural network potentials (HDNNPs) in *n2p2*. Polynomial symmetry functions¹ are designed to mimic closely the behavior of traditional Behler-Parrinello symmetry functions² but with a significantly reduced computational cost.

¹ Bircher, M. P.; Singraber, A.; Dellago, C. Improved Description of Atomic Environments Using Low-Cost Polynomial Functions with Compact Support. arXiv:2010.14414 [cond-mat, physics:physics] 2020.

² Behler, J. Atom-Centered Symmetry Functions for Constructing High-Dimensional Neural Network Potentials. J. Chem. Phys. 2011, 134 (7), 074106.

Purpose of Module

The symmetry functions proposed in the original work of Behler and Parrinello² contain expressions of the form $\exp(-\eta r_{ij}^2) f_c(r_{ij})$ in the innermost loop over all neighbors of atoms. Often the cutoff function $f_c(r)$ is chosen to be a cosine or hyperbolic tangent. Considering the computational cost of these transcendental functions an alternative formulation of symmetry functions based on polynomials like

$$((15 - 6x)x - 10)x^3 + 1$$

has been published recently¹. Here, cheap polynomials are combined to form compact functions in the radial and angular domain which mimic the behavior of Behler-Parrinello type symmetry functions at a significantly reduced execution time. The benefits, benchmarks and many example applications are presented in great detail in¹.

This module's changes to the *n2p2* code comprise of new classes for different types of polynomial symmetry functions (PSFs), some helper classes and a redesign of the symmetry function caching mechanism:

- Helper classes `CoreFunction` and `CompactFunction` allow unified access to the compact function building blocks of PSFs.
- Six different types of PSFs were implemented in these classes:
 - `SymFncCompRad`
 - `SymFncCompAngn`
 - `SymFncCompAngw`
 - `SymFncCompRadWeighted`
 - `SymFncCompAngnWeighted`
 - `SymFncCompAngwWeighted`

Here, `Rad` and `Angn/Angw` indicate radial and angular symmetry functions variants, respectively. The suffix `Weighted` refers to an element weighting proposed in³. For each new class in this list also a symmetry function group⁴ version was implemented, following the same naming scheme prefixed with `SymGrp`. See also [this section](#) of the *n2p2* documentation for a more detailed description of the PSFs and their parameters.

- The computation of a set of descriptors allows the reuse of intermediate results across multiple symmetry functions with varying parameters. The previously existing cutoff function caching⁴ of *n2p2* was significantly improved. By overriding the `getCacheIdentifiers()` member function each `SymFnc..` class can provide identifier strings for required cache fields. The `Mode::setupSymmetryFunctionCache()` function collects the requirements of all symmetry functions and assigns cache positions in the `Atom::Neighbor::cache` array.

Background Information

This module is based on *n2p2*, a C++ code for generation and application of neural network potentials used in molecular dynamics simulations. The source code and documentation are located here:

- *n2p2* documentation: <http://compphysvienna.github.io/n2p2/>
- *n2p2* source code: <http://github.com/CompPhysVienna/n2p2>

³ Gastegger, M.; Schwiedrzik, L.; Bittermann, M.; Berzsenyi, F.; Marquetand, P. WACSF—Weighted Atom-Centered Symmetry Functions as Descriptors in Machine Learning Potentials. *J. Chem. Phys.* 2018, 148 (24), 241709.

⁴ Singraber, A.; Behler, J.; Dellago, C. Library-Based LAMMPS Implementation of High-Dimensional Neural Network Potentials. *J. Chem. Theory Comput.* 2019, 15 (3), 1827–1840.

Building and Testing

The code changes from this module are already merged with the main repository of *n2p2* (see [pull request](#)).

Because the introduction of a new set of symmetry function enhances the core library of *n2p2* several applications shipped with *n2p2* will be affected by the changes. The easiest way to test the new functionality is to run the examples provided in these `examples/nnp-predict/` folders which make use of PSFs:

- Anisole_SCAN
- DMABN_SCAN
- Ethylbenzene_SCAN

First, since the changes from this module are already merged with the main repository of *n2p2* (see [pull request](#)) it is sufficient to [download the latest version](#). Then, compile the `nnp-predict` tool by running

```
make nnp-predict -j
```

in the `src` directory. Next, switch to one of the above example directories and run the prediction tool:

```
../../../../bin/nnp-predict 0
```

In the `SETUP: SYMMETRY FUNCTIONS` section of the output there should be symmetry functions with type (column `tp`) between 20 and 25 which identifies [different variants](#) of PSFs. In addition, the section `SETUP: SYMMETRY FUNCTION CACHE` contains an overview of the cache usage.

Regression testing is implemented in *n2p2* and automatically performed upon submission of a pull request via [Travis CI](#). The log file showing the successful pass of all tests for the specific pull request can be found [here](#). The tests include the above prediction examples and also perform a comparison of analytic and numeric derivatives of symmetry functions.

Source Code

The easiest way to view the source code changes covered by this module is to use the [GitHub pull request page](#). There, use the *Files changed* tab to review all changes.

Software Technical Information

The information in this section describes *n2p2* as a whole. Information specific to the additions in this module are in subsequent sections.

Name *n2p2* (NeuralNetworkPotentialPackage)

Language C++

Licence [GPL-3.0-or-later](#)

Documentation Tool Doxygen, Sphinx

Application Documentation <http://compphysvienna.github.io/n2p2/>

Relevant Training Material <http://compphysvienna.github.io/n2p2/>

Software Module Developed by Andreas Singraber

1.5.4 n2p2 - Symmetry Function Memory Footprint Reduction

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module improves memory management in *n2p2*. More specifically, a new strategy to store symmetry function derivatives is implemented. In this way the memory footprint during training is drastically reduced.

Purpose of Module

Training high-dimensional neural network potentials (HDNNPs) means to minimize the error between predictions and the reference information in a data set of atomic configurations. There, the desired potential energy surface is supplied in the form of an energy per configuration and forces acting on each atom. Consider the HDNNP expression for forces

$$F_{i,\alpha} = - \sum_{j=0}^{N_{\text{atoms}}} \sum_{k=0}^{N_{\text{sym.func.}}} \frac{\partial E_j}{\partial G_{j,k}} \frac{\partial G_{j,k}}{\partial x_{i,\alpha}},$$

where $G_{j,k}$ denotes the k -th symmetry function of atom j . Only the first expression $\frac{\partial E_j}{\partial G_{j,k}}$ depends on the neural network weights and therefore changes during the training process. The symmetry function derivatives with respect to atom coordinates $\frac{\partial G_{j,k}}{\partial x_{i,\alpha}}$, however, stay fixed for each atomic configuration in the data set. Given the high computational cost of symmetry functions it is essential to pre-calculate and store them in memory. While this strategy speeds up the training procedure significantly¹ it also drastically increases the memory footprint, which easily reaches more than 100 GB for common data set sizes.

This module alters the core C++ library of *n2p2* in order to reduce the memory consumption of all depending applications and provides benchmark results quantifying the improvement. The idea is to exploit that for specific combinations of neighboring atoms i and j , the expression $\frac{\partial G_{j,k}}{\partial x_{i,\alpha}}$ always equals zero. Consider a three-component system with elements A, B and C. In addition, let atoms i and j be of element A and B, respectively. Then, the derivative of a symmetry function $G_{j,k}$ with signature B-C (i.e. only sensitive to neighbor atoms of type C) with respect to i 's coordinates vanishes. Hence, by taking these element combination relations automatically into account a significant portion of the memory usage can be avoided. Depending on the symmetry function setup, savings of about 30 to 50% can be achieved for typical systems. These improvements will be particularly helpful for [developing HDNNPs for coarse-grained models](#).

Code changes cover most of the classes in the *libnnp* core library where they add functionality to identify relevant (nonzero) element combinations for the symmetry function derivative computation. Additional CI tests ensure that results are not affected.

Background Information

This module is based on *n2p2*, a C++ code for generation and application of neural network potentials used in molecular dynamics simulations. The source code and documentation are located here:

- *n2p2* documentation: <http://compphysvienna.github.io/n2p2/>
- *n2p2* source code: <http://github.com/CompPhysVienna/n2p2>

¹ Singraber, A.; Morawietz, T.; Behler, J.; Dellago, C. Parallel Multistream Training of High-Dimensional Neural Network Potentials. *J. Chem. Theory Comput.* 2019, 15 (5), 3075–3092.

Building and Testing

Because the change in memory management affects the core library of *n2p2* several applications shipped with *n2p2* will benefit from reduced memory consumption. However, the biggest effect can be observed during training with the `nnp-train` application. In the `src` directory type

```
make nnp-train
```

to build this *n2p2* tool (see the [build documentation](#) for more details). Switch to one of the folders inside the `examples/nnp-train` directory and run `nnp-train` (after a successful build the binary is copied to the `bin` directory). The screen output will contain a section labelled `SETUP: SYMMETRY FUNCTION MEMORY` which will highlight the memory savings.

The code changes from this module are already merged with the main repository of *n2p2* (see [pull request](#)). The improved memory management is enabled by default when *n2p2* is compiled. However, there are use cases (see [this discussion](#)) where the “full” memory layout is more desirable. Hence, a [compilation flag](#) allows to switch between the two choices. The documentation also shows benchmark results which demonstrate the potential memory savings.

Regression testing is implemented in *n2p2* and automatically performed upon submission of a pull request via [Travis CI](#). The log file showing the successful pass of all tests for the specific pull request can be found [here](#).

Source Code

The easiest way to view the source code changes covered by this module is to use the [GitHub pull request page](#). There, use the *Files changed* [tab](#) to review all changes.

Software Technical Information

Name *n2p2*: Symmetry Function Parameter Generator

Language [Python3 \(3.7\)](#)

Licence [GPL-3.0-or-later](#)

Documentation Tool [Sphinx](#), [ReST](#)

Application Documentation https://github.com/flobuch/n2p2/tree/symfunc_paramgen/tools/python/symfunc_paramgen/doc

Relevant Training Material https://github.com/flobuch/n2p2/tree/symfunc_paramgen/tools/python/symfunc_paramgen/examples

Software Module Developed by Florian Buchner

1.5.5 Symmetry Function Parameter Generator for *n2p2*

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Examples*
- *Source Code*

This module implements schemes from the literature ([GaSc2018], [ImAn2018]) for automatically generating parameter sets for Behler-Parrinello-type symmetry functions ([BePa2007], [Beh2011]), and variations thereof, in neural network potential applications. It is designed to work in conjunction with the `n2p2` package, but can be used as a standalone, too.

Purpose of Module

To represent potential energy surfaces via an artificial neural network, in a first step, the `n2p2` software package uses Behler-Parrinello-type symmetry functions ([BePa2007], [Beh2011]), and variations thereof. These serve as descriptors of an atom's local chemical environment, that make manifest, already at the input level, the output's invariance w.r.t. translations, rotations, and particle number permutations. They are obtained via a transformation from the cartesian coordinates of the atom and its neighbor atoms.

The choice of these symmetry functions is an important step in the application of a neural network potential. Both the number of symmetry functions to be used, and the parameters of those symmetry functions (symmetry functions with different parameters being sensitive to different regions in an atom's surroundings), need to be decided on. In principle, using more symmetry functions yields a more complete description of an atom's chemical environment and thus improves accuracy. On the other hand, the numerical computation of those symmetry functions in fact tends to be the most computationally expensive step in the application of a neural network potential, so it is undesirable to use too many symmetry functions.

Now, what this module does is implement algorithms from the literature ([GaSc2018], [ImAn2018]) for automatically generating sets of these symmetry function parameters. The aim of these algorithms is to create symmetry function parameter sets that capture all the possible spatial correlations of atoms with their neighbors as completely as possible, while still being economical (i.e., as few symmetry functions as possible), and to do so in a more systematic fashion than when parameters are chosen by hand.

Note that the implemented procedures for generating symmetry function parameter sets are agnostic to the actual dataset of atom configurations that the neural network potential is applied to, and the correlations of atoms therein. They are merely a way of covering all regions in an atom's environment as completely, yet at the same time as parsimoniously, as possible, without any knowledge of where neighbor atoms in a given system are actually most likely to be located. The symmetry functions generated this way are what is referred to as the 'pool of candidate SFs' in [ImAn2018]. This is alluding to the fact that this 'pool of candidate SFs' could then be further sparsified, keeping only those symmetry functions that have the greatest descriptive power for a given dataset. Functionality for this is, however, not currently implemented in this module.

The main module file is `sfparamgen.py`, in `tools/python/symfunc_paramgen/src`. It implements the class `SymFuncParamGenerator`, which provides methods for the parameter generation described above, as well as for outputting the parameter sets in the format that is required for the parameter file `input.nn` used by `n2p2`.

Background Information

This module is designed to be used in conjunction with `n2p2`, a software package for high-dimensional neural network potentials in computational physics and chemistry. For more information on `n2p2` itself, see:

- `n2p2` documentation: <https://compphysvienna.github.io/n2p2/index.html>
- `n2p2` source code: <https://github.com/CompPhysVienna/n2p2>

That being said, this module does not directly interface the core of `n2p2` or call any of its functionality. The communication of this module with the core of `n2p2` is limited to outputting symmetry function parameter sets in a format that `n2p2` can read (the format in which symmetry functions are specified in the parameter file `input.nn` of `n2p2`). Therefore, the module's functionality for generating symmetry function parameter sets can in principle be used independently of `n2p2`.

Building and Testing

Seeing as this module itself is just a lightweight Python tool and does not directly interface the core of n2p2, it does not require building. Realistically, however, you will want to use it in conjunction with n2p2's functionality for neural network potentials, for which you need to build n2p2. This is described [here](#).

Follow these steps to test the module:

1. Install the `pytest` package.
2. Navigate to the `tools/python/symfunc_paramgen/tests` directory.
3. Run `pytest` in your terminal.
4. For an additional code coverage report install the `pytest-cov` package.
5. Go to the `tools/python/symfunc_paramgen/tests` directory.
6. Execute `pytest --cov=sfparamgen --cov-report=html ..`

Examples

See the `example.ipynb` IPython notebook in the `tools/python/symfunc_paramgen/examples` directory (here is a direct link: https://github.com/flobuch/n2p2/tree/symfunc_paramgen/tools/python/symfunc_paramgen/examples). Inside the examples folder, run the example by typing `jupyter notebook example.ipynb` in your terminal.

Source Code

The source code for this module can be found [here](#).

Ultimately, this module is intended to be merged into the official n2p2 code. For the status of the corresponding pull request, see [here](#).

Software Technical Information

Name NNTSSD - Tools for Neural Network Training Set Size Dependence

Language Python3

Licence GPL-3.0-or-later

Documentation Tool Sphinx

Application Documentation Available [here](#)

Relevant Training Material Included in the Documentation above.

Software Module Developed by Madlen Reiner

1.5.6 n2p2: Tools for Training Set Size Dependence

- *Purpose of Module*
- *Background Information*

- *Building, Testing and Examples*
- *Source Code*

This module provides tools to analyse the training set size dependence of residual error of neural network potentials (NNPs). It is specifically written to be used with the NNP `n2p2`.

Purpose of Module

NTTSSD is a module that allows

- automated dataset creation of varied sizes
- training of the neural network
- analysis of the learning curves obtained in the training process

in order to determine representative learning curves showing residual errors for varied sizes of training sets. It also provides tools that allow

- the usage of external test sets, which might be useful for developing epoch optimization approaches
- the usage of separate validation datasets, which are used to obtain TSSD curves that are independent from test sets that are used for epoch optimization
- graphic representation of learning curves and training performance
- a user-friendly way of running NTTSSD methods by filling in an input file

Other methods within the module allow

- processing of input data (namely splitting datasets)
- analysis of training performance (dependence of residual error of the number of training epochs)

Background Information

Neural network potentials are used in molecular dynamics simulation to reproduce potential energy surfaces of ab initio methods. This module addresses the question of dependence of the NNP's prediction error (characterized by the RMSE in energy and forces) on the size of the training dataset.

Building, Testing and Examples

Building instructions for NTTSSD, information regarding software tests and examples can be found [here](#). The additions to `n2p2` presented here are not yet merged with the main `n2p2` repository. Before following the above instructions please check out the `n2p2_training_size` branch in [the author's fork of n2p2](#) using these commands:

```
git clone git@github.com:MadlenReiner/n2p2.git
cd n2p2
git checkout n2p2_training_size
```

Then, run the build process of `n2p2`

```
cd src
make
```

to create the training tools required for NTTSSD. In some cases it may be required to set paths to external libraries in `src/makefile.gnu`.

Source Code

The source code of this module can be found in the `tools/python/NTTSSD/source` of the `n2p2_training_size` branch in the author's fork:

- [Link to NTTSSD source code](#)

Another way of reviewing the code additions to `n2p2` is to visit the corresponding pull request:

- [Link to the pull request to include NTTSSD in n2p2](#)

Change to the tab *Files changed* to get an overview of all changes.

1.6 Pilot Projects

One of primary activity of E-CAM is to engage with pilot projects with industrial partners. These projects are conceived together with the partner and typically are to facilitate or improve the scope of computational simulation within the partner. The related code development for the pilot projects are open source (where the licence of the underlying software allows this) and are described in the modules associated with the pilot projects.

More information on Classical MD pilot projects can be found on the main E-CAM website:

- [Project on binding kinetics](#)
- [Project on food and pharmaceutical proteins](#)

The following modules were developed specifically for the Classical MD pilot projects.

Software Technical Information

This module extends the `contact_maps` project.

Name `contact_maps`

Language Python 2.7, 3.5, 3.6

Licence LGPL 2.1+

Documentation Tool Sphinx/RST

Application Documentation <http://contact-map.readthedocs.io/>

Relevant Training Material TODO

Software Module Developed by David W.H. Swenson

1.6.1 Contact Concurrences

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
 - *Examples*
- *Source Code*

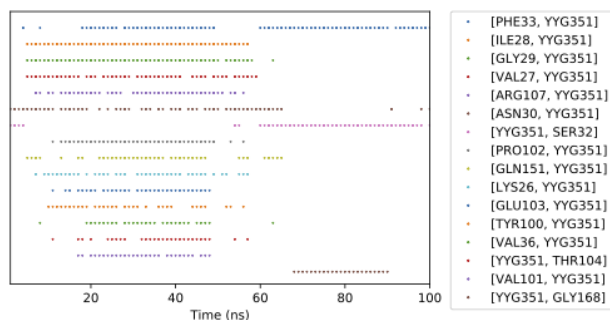
This module deals with the analysis of contacts between parts of biomolecules based on “contact concurrences,” i.e., what contacts occur simultaneously during a trajectory. This is useful when using contacts as a definition of a metastable state in a trajectory.

Purpose of Module

Contact frequencies, as developed in the module *Contact Map*, are a useful tool for studying biomolecular systems, such as binding/unbinding of a ligand from a protein. However, they suffer from one problem when trying to use them to define metastable states: since they are averaged over time, they don’t show time-dependent behavior. To identify a stable state, time-dependent behavior must be considered.

For example, a particular contact pair might have a frequency of 0.1 during a 100ns trajectory. But this could be achieved in several ways. If the contact events are randomly distributed through time, this contact probably isn’t characteristic of a metastable state. On the other hand, if the contact is constantly present during the last 10 ns (and not otherwise present), it might represent a metastable state. More importantly, there might be multiple contacts that are *all* present during those last 10 ns. Those concurrent contacts could be used to define a metastable state. This module helps identify and analyze those concurrent contacts by providing a tool to visualize them.

The figure shows the output of the contact concurrence visualization for the contacts between an inhibitor (labelled YYG) and various residues of the protein GSK3B. The plot shows when each contact occurred. The x-axis is time. Each dot represents that a specific contact pair is present at that time. The contact pairs are separated along the vertical axis.



This trajectory shows two groups of stable contacts between the protein and the ligand; i.e. there is a change in the stable state. This allows us to visually identify the contacts involved in each state. Both states involve the ligand being in contact with Phe33, but the earlier state includes contacts with Ile28, Gly29, etc., while the later state includes contacts with Ser32 and Gly168.

This is an important tool for identifying stable states based on long-lived groups of contacts, and is being used as part of the *E-CAM pilot project on binding kinetics*. It has also been used a part of a bachelor’s thesis project to develop an automated approach to identifying metastable intermediates during binding/unbinding processes.

Classes implemented in this module include:

- `Concurrence`: Superclass for contact concurrence objects, enabling future custom concurrence types.
- `AtomContactConcurrence`: Contact concurrences for atom-atom contacts.
- `ResidueContactConcurrence`: Contact concurrences for residue-residue contacts (based on minimum distance between constituent atoms).
- `ConcurrencePlotter` and `plot_concurrences`: Class and convenience function (respectively) for making plots of contact concurrence.
- `ContactsDict`: Dict-like object giving access to atom or residue contacts based on string keys. Also added `ContactObject.contacts` property, which returns a `ContactsDict` object for the `ContactObject`.

Background Information

This module is part of the *contact_map* project, which builds on tools from *MDTraj*.

Building and Testing

This module will be included in the 0.4 release of `contact_map`. After that release, it can be easily installed with conda, using `conda install -c conda-forge contact_map`, or `conda install -c conda-forge contact_map==0.4.0` for the first version that includes this module. To see the current release, go to <https://pypi.org/project/contact-map/#history>.

Until the release, this module can only be installed through a developer install of `contact_map`. This involves downloading the `contact_map` repository, installing the requirements, and then installing the `contact_map` package from source. Instructions can be found on the [installation page](#) of the `contact_map` documentation.

Once installed, tests are run using `pytest`. To check that the code has been correctly installed, run `python -c "import contact_map"` from the command line. To run the tests, install `pytest` and run the command `py.test --pyargs contact_map`.

Examples

An example can be found in the documentation to the `contact_map` paper: [\[docs | GitHub\]](#)

Source Code

The source code for this module is contained in the following pull requests in the `contact_map` repository:

- https://github.com/dwhswenson/contact_map/pull/28
- https://github.com/dwhswenson/contact_map/pull/47

1.6.2 Particle Insertion Core

Software Technical Information

This is the core module for the particle insertion suite of codes

Languages C, Python 2.7, LAMMPS Scripting language

Licence MIT -however note that LAMMPS is now changing from GPL to LGPL so when used together with LAMMPS LGPL applies

Documentation Tool ReST

Application Documentation See [PIcore repository](#)

Relevant Training Material None

- *Purpose of the Module*
- *Background Information*
- *General Formulation*
- *Algorithms*
- *Source Code*
- *Compilation and Linking*

Purpose of the Module

This software module computes the change in free energy associated with the insertion or deletion of Lennard Jones particles in dilute or dense conditions in a variety of Thermodynamic Ensembles, where statistical sampling through molecular dynamics is performed under LAMMPS but will be extended to other molecular dynamics engines at a later date. Lennard-Jones type interactions are the key source of difficulty associated with particle insertion or deletion, which is why this module is a core module, as other interactions including Coulombic and bond, angle and dihedral interactions will be added in a second module. It differs from the main community approach used to date to compute such changes as it does not use soft-core potentials. Its key advantages over soft-core potentials are: (a) electrostatic interactions can in principle be performed simultaneously with particle insertion (this and other functionalities will be added in a new module); and, (b) essentially exact long-range dispersive interactions using [dispersion Particle Mesh Ewald](#) (PMME) or EWALD if desired can be selected at runtime by the user.

Background Information

Particle insertion can be used to compute the free energy associated with hydration/drying, the insertion of cavities in fluids/crystals, changes in salt levels, changes in solvent mixtures, and alchemical changes such as the mutation of amino-acids. in crystals. It can also be used to compute the free energy of solvent mixtures and the addition of salts, which is used in the purification processing industrially, for instance in the purification of pharmaceutical active ingredients. Particle insertion can in principle also be used to compute the free energy associated with changes in the pH, that is the proton transfer from a titratable site to the bulk, for example in water.

Our approach consists of rescaling the effective size of inserted atoms through a parameter λ so that all interactions between inserted atoms and interactions between inserted atoms and atoms already present in the system are zero when $\lambda = 0$, creating at most an integrable singularity which we can safely handle. In the context of Lennard-Jones type pair potentials, our approach at a mathematical level is similar to Simonson, who investigated the mathematical conditions required to [avoid the singularity of insertion](#). It turns out that a non-linear dependence of the interaction on λ between inserted atoms and those already present is required (i.e. a simple linear dependence on λ necessarily introduces a singularity).

This module and upcoming modules include computing the free energy changes associated with the following applications

- (a) hydration and drying;
- (b) the addition of multiple molecules into a condensed environment;
- (c) residue mutation and alchemy;
- (d) constant pH simulations, this also will also exploit modules created in E-CAM work package 3 (quantum dynamics); and,
- (e) free energy changes in chemical potentials associated with changes in solvent mixtures.

General Formulation

Consider a system consisting of $N + M$ degrees of freedom and the Hamiltonian

$$H(r, p, \lambda) = H_0 + K E_{insert} + \Delta V(r, \lambda)$$

where H_0 corresponds to an unperturbed Hamiltonian, and the perturbation $\Delta V(r, \lambda)$ depends nonlinearly on a control parameter λ . The first set of N degrees of freedom is denoted by A and the second set of M degrees of freedom is

denoted by B . To explore equilibrium properties of the system, thermostats, and barostats are used to sample either the NVT (canonical) ensemble or the NPT (Gibbs) ensemble. The perturbation is devised so that when $\lambda = 0$, $\Delta V(r, \lambda) = 0$, B is in purely virtual. When $\lambda = 1$, B corresponds to a fully physical augmentation of the original system.

In the present software module, we consider only interaction Lennard Jones atoms.

$$\Delta V(r, \lambda) = V_{ij}(r, \lambda)$$

where for each inserted atom i

$$\hat{\sigma}(\lambda)_i = \lambda \sigma_i$$

$$\hat{\epsilon}(\lambda)_i = \lambda \epsilon_i$$

and the mixing rule for Van der Waals diameters and binding energy between different atoms uses the geometric mean. The dependence of σ on λ has the consequence that the mean σ between a pair of inserted atoms scales as λ , but scales as $\sqrt{\lambda}$ when one atom in the pair is inserted and the other is already present. These choices of perturbations guarantees that the particle insertion and deletion catastrophes are avoided.

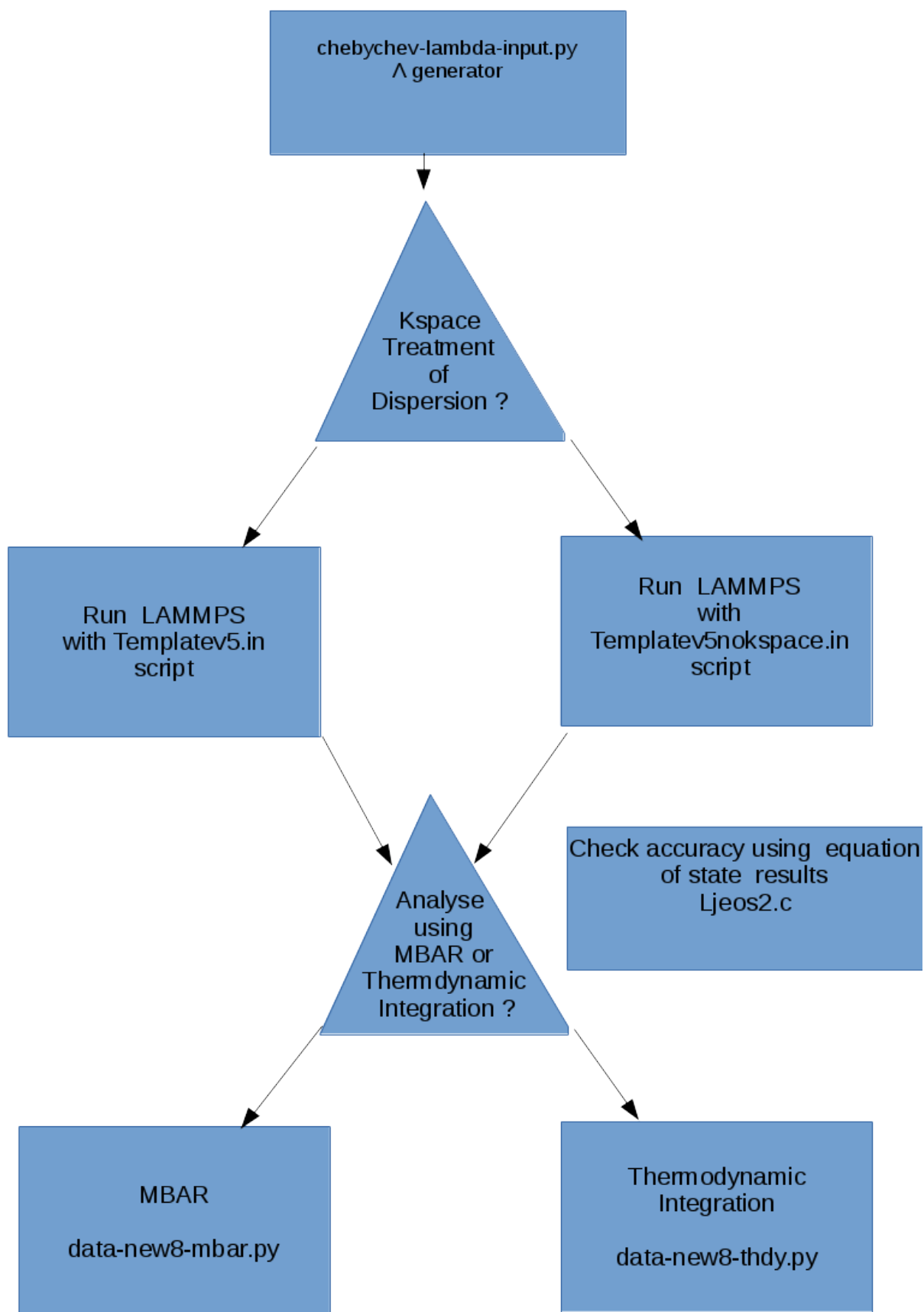
Algorithms

At the core of the PI core module there are four functions/codes. The first written in python generates the interpolation points which are the zero's of suitably transformed Chebyshev functions.

The second code written in LAMMPS scripting language performs the simulation in user-defined ensembles at the selected interpolation values of λ , at a user-specified frequency, computing two-point central difference estimates of derivatives of the potential energy needed for thermodynamic integration, computing the energy functions for all values of λ in the context of MBAR. The user also specifies the locations of the inserted particles. The user also specifies whether Particle Mesh Ewald or EWALD should be used for dispersive interactions.

The third code written in python takes the output data from LAMMPS, prepares it so that free energy differences in the selected ensemble can be computed using MBAR provided by the pymbar suite of Python codes of the Chodera group.

The fourth code, also written in python take the LAMMPS output and performs the thermodynamic integration.



Source Code

All files can be found in the `PIcore` subdirectory of the [particle_insertion](#) git repository.

Compilation and Linking

See [PIcore README](#) for full details.

Scaling and Performance

As the module uses LAMMPS, the performance and scaling of this module should essentially be the same, provided data for thermodynamic integration and MBAR are not generated too often, as is demonstrated below. In the case of thermodynamic integration, this is due to the central difference approximation of derivatives, and in the case of MBAR, it is due to the fact that many virtual moves are made which can be extremely costly if the number of interpolating points is large. Also, when using PMME, the initial setup cost is computationally expensive, and should, therefore, be done as infrequently as possible. A future module in preparation will circumvent the use of central difference approximations of derivatives. The scaling performance of PI-CORE was tested on Jureca multi node. The results for weak scaling (where the number of core and the system size are doubled from 4 to 768 core) are as follows.

Weak Scaling:

| Number of MPI Core | timesteps/s |
|--------------------|-------------|
| 4 | 1664.793 |
| 8 | 1534.013 |
| 16 | 1458.936 |
| 24 | 1454.075 |
| 48 | 1350.257 |
| 96 | 1301.325 |
| 192 | 1263.402 |
| 384 | 1212.539 |
| 768 | 1108.306 |

and for the strong scaling (where the number of core are doubled from 4 to 384 but the system size is fixed equal to 768 times the original system size considered for one core/processor for weak scaling) Strong Scaling:

| Number of MPI Core | timesteps/s |
|--------------------|-------------|
| 4 | 9.197 |
| 8 | 17.447 |
| 16 | 34.641 |
| 24 | 53.345 |
| 48 | 104.504 |
| 96 | 204.434 |
| 192 | 369.178 |
| 384 | 634.022 |

1.6.3 Particle Insertion Hydration

Software Technical Information

This is the core module for the particle insertion suite of codes

Languages C, Python 2.7, LAMMPS Scripting language

Licence MIT -however, note that LAMMPS is GPL so when used together GPL applies

Documentation Tool All source code should be documented so please indicate what tool has been used for documentation. We can help you with Doxygen and ReST but if you use other tools it might be harder for us to help if there are problems.

Application Documentation See [PIhydration README file](#)

Relevant Training Material Add a link to any relevant training material.

- *Purpose of the Module*
- *Background Information*
- *General Formulation*
- *Algorithms*
- *Source Code*
- *Compilation and Linking*
- *Scaling and Performance*

Purpose of the Module

This software module computes the change in free energy associated with the insertion or deletion of water in dilute or dense conditions in a variety of Thermodynamic Ensembles, where statistical sampling through molecular dynamics is performed under [LAMMPS](#) but will be extended to other molecular dynamics engines at a later date. It builds on the PI Core module of codes by adding electrostatic, bond, and angle λ dependent interactions including SHAKE to the Lennard-Jones interactions that were dealt with in PCore. It differs from the main community approach used to date to compute such changes as it does not use soft-core potentials. Its key advantages over soft-core potentials are: (a) electrostatic interactions can in principle be performed simultaneously with particle insertion (this and other functionalities will be added in a new module); and, (b) essentially exact long-range dispersive interactions using [dispersion Particle Mesh Ewald](#) (PMME) or EWALD if desired can be selected at runtime by the user.

Background Information

Particle insertion can be used to compute the free energy associated with hydration/drying, the insertion of cavities in fluids/crystals, changes in salt levels, changes in solvent mixtures, and alchemical changes such as the mutation of amino-acids. in crystals. It can also be used to compute the free energy of solvent mixtures and the addition of salts, which is used in the purification processing industrially, for instance in the purification of pharmaceutical active ingredients. Particle insertion can in principle also be used to compute the free energy associated with changes in the pH, that is the proton transfer from a titratable site to the bulk, for example in water.

Our approach consists of rescaling electrostatic charges of inserted atoms so that they converge to zero faster than inserted Van der Waals atoms where the later uses the geometric mean for Lennard Jones diameters and binding energies, and that bond, angle, and dihedral spring constants and where necessary also bond lengths scale to zero in the same fashion

the effective size of inserted atoms through a parameter λ so that all interactions between inserted atoms and interactions between inserted atoms and atoms already present in the system are zero when $\lambda = 0$, creating at most an integrable singularity which we can safely handle. In the context of Lennard-Jones type pair potentials, our approach at a mathematical level is similar to Simonson, who investigated the mathematical conditions required to [avoid the singularity of insertion](#). It turns out that a non-linear dependence of the interaction on λ between inserted atoms and those already present is required (i.e. a simple linear dependence on λ necessarily introduces a singularity).

The applications of this module use in upcoming modules include computing the free energy changes associated with:

- (a) hydration **and** drying;
- (b) the addition of multiple molecules into a condensed environment;
- (c) residue mutation **and** alchemy;
- (d) constant pH simulations, this also will also exploit modules created **in** E-CAM
 \hookrightarrow work package 3
 (quantum dynamics); **and**,
- (e) free energy changes **in** chemical potentials associated **with** changes **in** solvent
 \hookrightarrow mixtures.

General Formulation

Consider a system consisting of $N + M$ degrees of freedom and the Hamiltonian

$$H(r, p, \lambda) = H_0 + K E_{insert} + \Delta V(r, \lambda)$$

where H_0 corresponds to an unperturbed Hamiltonian, and the perturbation $\Delta V(r, \lambda)$ depends nonlinearly on a control parameter λ . The first set of N degrees of freedom is denoted by A and the second set of M degrees of freedom is denoted by B . To explore equilibrium properties of the system, thermostats, and barostats are used to sample either the NVT (canonical) ensemble or the NPT (Gibbs) ensemble. The perturbation is devised so that when $\lambda = 0$, $\Delta V(r, \lambda) = 0$, B is in purely virtual. When $\lambda = 1$, B corresponds to a fully physical augmentation of the original system.

In the present software module, we include in the perturbation interaction Lennard Jones potentials, harmonic bond and angle interactions, and electrostatic interactions:

$$\Delta V(r, \lambda) = V_{lj}(r, \lambda) + V_b(r, \lambda) + V_a(r, \lambda) + V_{el}(r, \lambda).$$

where for each inserted atom i

$$\hat{\sigma}(\lambda)_i = \lambda \sigma_i$$

$$\hat{\epsilon}(\lambda)_i = \lambda \epsilon_i$$

$$\hat{q}(\lambda)_i = \lambda^p$$

and the mixing rule for Van der Waals diameters and binding energy between different atoms uses the geometric mean for atoms pairs where one or more of the atoms is inserted but retains the mixing rule for atoms already present. The dependence of σ on λ has the consequence that the mean σ between a pair of inserted atoms scales as λ , but scales as $\sqrt{\lambda}$ when one atom in the pair is inserted and the other is already present. The dependence of ϵ on λ ensures that forces behave regularly when λ is very small. These choices of perturbations guarantees that the particle insertion and deletion catastrophes are avoided. Regarding electrostatic interactions, the exponent p allows the rate of convergence electrostatic interactions to zero to be faster than the rate at which that the effective diameters between corresponding Lennard Jones atoms go to zero, so as to ensure divergences are avoided. Currently $p = 1.5$. The spring constants for harmonic, angular and torsional interactions involving inserted atoms are currently simply multiplied

by λ . It is also possible to replace bond, angle and torsional interactions involving only inserted atoms with shake constraints. In such cases, the shake constraints are continuously on. For cases where arithmetic sum rules apply to the original system, an additional lambda bases perturbation stage can be applied to transform geometric mean based mixing rules for Lennard Jones interactions to arithmetic mean rules governing interactions between inserted atoms or inserted atoms and original atoms.

Algorithms

At the core of the PI core module there are four functions/codes. The first written in python generates the interpolation points which are the zero's of suitably transformed Chebyshev functions.

The second code written in LAMMPS scripting language performs the simulation in user-defined ensembles at the selected interpolation values of `:math:'lambda'`, at a user-specified frequency, computing two-point central difference estimates of derivatives of the potential energy needed for thermodynamic integration, computing the energy functions for all values of `:math:'lambda'` in the context of MBAR. The user also specifies the locations of the inserted particles. The user also specifies whether Particle Mesh Ewald or EWALD should be used for dispersive interactions.

The third code written in python takes the output data from LAMMPPS, prepares it so that free energy differences in the selected ensemble can be computed using MBAR provided by the pymbar suite of python codes of the Chodera group.

The fourth code, also written in python take the LAMMPS output and performs the thermodynamic integration.

Source Code

All files can be found in the `PIhydration` subdirectory of the [particle_insertion git repository](#).

Compilation and Linking

See [PIhydration README](#) for full details.

Scaling and Performance

As the module uses LAMMPS, the performance and scaling of this module should essentially be the same, provided data for thermodynamic integration and MBAR is not generated too often. In the case of thermodynamic integration, this is due to the central difference approximation of derivatives, and in the case of MBAR, it is due to the fact that many virtual moves are made which can be extremely costly if the number of interpolating points is large. Also, when using PMME, the initial setup cost is computationally expensive, and should, therefore, be done as infrequently as possible. A future module in preparation will circumvent the use of central difference approximations of derivatives.

Software Technical Information

This module extends the lammps python interface to allow accessing various force-field potential parameters from python.

Language Python (3+)

Licence The software for this specific module is licensed under [GNU General Lesser Public License v3.0](#)

Documentation Tool [Sphinx](#), follows LAMMPS format (ReST)

Application Documentation [Documentation](#)

Relevant Training Material Not currently available.

Software Module Developed by Shrinath Kumar, Zein Jaafar and Donal MacKernan

1.6.4 LAMMPS-pyinterfaceExt

- *Purpose of Module*
 - *Background Information*
 - *Building and Testing*
- *Source Code*

The module contains patch files for the [Stable release 29 October 2020](#) version of LAMMPS, to enable accessing simulation force-field parameters from python.

Purpose of Module

When performing alchemical free energy calculations, it is necessary to change the attributes of various particles in a simulations - Atom properties such as charge and mass or Force field properties such as ϵ or σ of Lennard-Jones potentials.

The LAMMPS library along with its Python interface provides the ability to directly access and change many such attributes in a running simulation. However, for pair potentials while it is possible to change their parameters it is currently not possible to read the existing parameters from a running simulations. This is required for the alchemical free energy calculations using the Particle Insertion approach as described in the [Particle Insertion Core](#) module, where the scaling of the forcefield attribute depends on the existing attributes.

This module address this limitation by extending the LAMMPS library and the Python interface to add a function allowing read access to pair-potential parameters. It also adds an extract method to the `pair_*.cpp` files associated with some commonly used pair-potential in LAMMPS as described in [fix adapt](#).

Background Information

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a very versatile MD engine. The current stable release of which can be obtained from the link [LAMMPS stable 29Oct2020](#). In particular, due to it's powerful library and python interface it allows great control and easy scripting of any time of simulation

The Particle Insertion approach for alchemical free energy calculations is currently only implemented using the LAMMPS MD engine. The set of patches in this module exist to accommodate the requirements of those modules.

Building and Testing

Download and extract the LAMMPS source. Before building LAMMPS download the patch file and apply the patch using the `patch` command from the root of the LAMMPS source directory.

```
patch < lmp_pyExt.patch
```

Follow the normal LAMMPS building instructions at <https://lammps.sandia.gov/doc/Build.html> to build and install LAMMPS along with the applied patch. Make sure to enable `pkg_python` when building.

Source Code

Available as a patch file from [here](#)

Software Technical Information

Name minDist2segments_KKT

Language C++

Licence [MIT](#)

Documentation Tool Sphinx

Application Documentation [Doxygen documentation](#)

Relevant Training Material [PDF documentation](#)

Software Module Developed by Pascal Carrivain

1.6.5 E-CAM minDist2segments_KKT module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The minDist2segments_KKT module returns the minimal distance between two [line segments](#). It uses the Karush-Kuhn-Tucker conditions ([KKT](#)) for the minimization under constraints.

Purpose of Module

To study the long term memory of the initial conformation of a highly entangled polymer we need to preserve the topology. It means that two polymer bonds cannot cross. It is of great importance for the study of post-mitotic chromosome unfolding. To resolve the excluded volume constraints one could use a soft or hard potential between the two points associated to the minimal distance.

- Polymer simulation.
- To resolve the excluded volume constraints.
- It is used in a scientific collaborations.
- Publications: not currently available.

Note: We would use the present module to avoid topology violation in an entangled polymer system. This module is used by other ongoing work.

Background Information

You can find pdf file with a detailed derivation of the minimal distance between two segments using the Karush-Kuhn-Tucker conditions on the [minDist2segments_KKT GitLab repository](#).

Building and Testing

I provide a simple Makefile you can find at the same location that the source code. You need C++11 in order to use the pseudo-random number generator. The example also has OpenMP acceleration, edit the Makefile to enable it. Before the compilation you can clean the previous build with the `make mrproper` command.

The purpose of the module is to calculate the minimal distance between two segments. For each distance we compare the result to an “exact enumeration” of all the possible distances and return a warning if the two results differ by more than the enumeration precision.

Source Code

The source code and more information can be find at [minDist2segments_KKT GitLab repository](#).

Software Technical Information

Name minDist2segments_KKT_for_SRP

Language C/C++, LAMMPS

Licence MIT

Documentation Tool sphynx

Application Documentation [doxygen documentation](#)

Relevant Training Material [pdf documentation](#)

Software Module Developed by Pascal Carrivain

1.6.6 E-CAM minDist2segments_KKT_for_SRP module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The minDist2segments_KKT_for_SRP module returns the minimal distance between two [line segments](#). It uses the Karush-Kuhn-Tucker conditions (KKT) for the minimization of distance under constraints. The module implements the previous function for the [SRP fix in LAMMPS](#). Indeed, the SRP function to compute the minimal distance does not always give the correct solution.

Purpose of Module

To study the long term memory of the initial conformation of a highly entangled polymer we need to preserve the topology. That means that two polymer bonds cannot cross. It is of great importance for the study of post-mitotic chromosome unfolding. Minimal distance between two bonds can be used in [Dissipative-Particle-Dynamics](#) to prevent bond crossings (see the reference [[Kumar2001](#)] and [[Sirk2012](#)]) too. To resolve the excluded volume constraints one could use a repulsive potential between the two points associated to the minimal distance (see the reference [[Kumar2001](#)]). We propose a new option in the computation of the minimal distance in the [SRP fix](#) for LAMMPS. Indeed, SRP fix computes the minimal distance between two infinite lines and reset the solution to occur along the interior of the bond. This method is not always accurate. The KKT conditions allows to solve the problem of minimal distance such finite segment length constraint holds.

Note: It is part of [E-CAM post-doc pilot project](#).

Background Information

You can find pdf file with a detailed derivation of the minimal distance between two segments using the Karush-Kuhn-Tucker conditions on the [minDist2segments_KKT GitLab repository](#). The modifications are to an existing code base [SRP fix](#) for LAMMPS.

Building and Testing

I provide simple modifications to the [SRP fix](#) files in the LAMMPS source code. In order to use minimal distance between two segments with KKT conditions you need to pass **min_KKT** to the **distance** argument of the [SRP fix](#). The instructions to install, test and run the module can be found on the [minDist2segments_KKT GitLab repository](#). The purpose of the module is to calculate the minimal distance between two segments. For each distance we compare the result to an “exact enumeration” of all the possible distances and return a warning if the two results differ by more than the enumeration precision.

Source Code

You can find the modifications for the [SRP fix](#) files on the [minDist2segments_KKT GitLab repository](#) for [_SRP](#) folder.

Software Technical Information

Name velocities_resolve_EVC

Language C

Licence [MIT](#)

Documentation Tool Doxygen

Application Documentation [Reference manual](#)

Relevant Training Material [PDF documentation](#)

Software Module Developed by Pascal Carrivain

1.6.7 E-CAM velocities_resolve_EVC module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

`velocities_resolve_EVC` is a module that resolves the excluded volume constraint with a velocity formulation (no potential applied between two overlapped bonds). `velocities_resolve_EVC` uses the module *E-CAM minDist2segments_KKT module* to find the minimal distance between two bonds.

Purpose of Module

To study the long term memory of the initial conformation of a highly entangled polymer we need to preserve the topology. It means that two bonds cannot cross. It is of great importance for the study of post-mitotic chromosome unfolding. To resolve the excluded volume constraints you could use a soft or hard potential between the two points associated to the minimal distance. Here, we propose to change the relative velocity between overlapped bonds to resolve the excluded volume constraint in one time-step of molecular dynamics.

- Polymer simulation.
- To resolve the excluded volume constraints.
- It is used in a scientific collaboration.
- Publications: not currently available.

Note: We would use the present module to avoid topology violation in an entangled polymer system. The present module uses the E-CAM module *E-CAM minDist2segments_KKT module*.

Note: This module is a part of a pilot project (E-CAM post-doc). We would use it to avoid topology violation in an entangled polymer system.

Background Information

You can find a PDF file with a detailed derivation of the velocity-based method we use to resolve the excluded volume constraint in one time-step of molecular dynamics on the `velocities_resolve_EVC` [GitLab repository](#).

Building and Testing

I provide a simple Makefile you can find at the same location as the source code. You need C++11 in order to use pseudo-random number generator. Before the compilation you can clean the previous build with “make mrproper” command. The purpose of the module is to resolve excluded volume constraints. Therefore, we provide a simple example of a system of N bonds with volume interactions. We test every n iterations the average overlap pairwise.

Source Code

The source code and more information can be find at [velocities_resolve_EVC](https://gitlab.com/pcarrivain/velocities_resolve_EVC) GitLab repository.

Software Technical Information

Name `velocities_resolve_EVC_for_LAMMPS`

Language C/C++, LAMMPS

Licence MIT

Documentation Tool doxygen

Application Documentation 'https://gitlab.com/pcarrivain/velocities_resolve_evc/-/blob/master/refman.pdf'

Relevant Training Material 'https://gitlab.com/pcarrivain/velocities_resolve_evc/-/blob/master/velocities_resolve_EVC.pdf'

Software Module Developed by Pascal Carrivain

1.6.8 E-CAM velocities_resolve_EVC_for_LAMMPS module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The **velocities_resolve_EVC_for_LAMMPS** is a module that resolve the excluded volume constraint with a velocity formulation (no potential applied between two bonds). It is an implementation for **LAMMPS** of an already existing module [velocities_resolve_EVC](https://gitlab.com/pcarrivain/velocities_resolve_EVC) GitLab repository. The **velocities_resolve_EVC_for_LAMMPS** uses the module **minDist2segments_KKT_for_SRP** (you can find on the [minDist2segments_KKT_for_SRP](https://gitlab.com/pcarrivain/minDist2segments_KKT_for_SRP) GitLab repository) to find the minimal distance between two bonds.

Purpose of Module

To study the long term memory of the initial conformation of a highly entangled polymer we need to preserve the topology. It means that two bonds cannot cross. It is of great importance for the study of post-mitotic chromosome unfolding. Preservation of topology is also used in the framework of [Dissipative-Particle-Dynamics](#) in particular for the study of rheological properties. To resolve the excluded volume constraints one could use a soft or hard potential between the two points (each point belong to one of the two overlapping bonds) associated to the minimal distance. Here, we propose to change the relative velocity between overlapped bonds to resolve the excluded volume constraint in one time-step of molecular dynamics. We propose to implement this functionality as a new fix for **LAMMPS**.

- It is used in a scientific collaboration.
- Publications: not currently available.

Note: The present module uses the E-CAM module **minDist2segments_KKT_for_SRP** you can find on the [minDist2segments_KKT](https://gitlab.com/pcarrivain/minDist2segments_KKT) GitLab repository. It also uses the E-CAM module **velocities_resolve_EVC** you can find on

the [velocities_resolve_EVC](#) GitLab repository. This module is a part of a E-CAM post-doc pilot project.

Background Information

You can find a pdf file with a detailed derivation of the velocity-based method we use to resolve the excluded volume constraint in one time-step of molecular dynamics on the [velocities_resolve_EVC](#) GitLab repository.

Building and Testing

The instruction to build and run test are available on the GitLab repository. The purpose of the module is to resolve excluded volume constraints for polymer system. Therefore, we provide a simple [LAMMPS](#) input file of a system of C chains of N bonds each with volume interactions. In particular, we use the [LAMMPS](#) implementation of [FENE bond](#). The algorithm we propose here checks every time-step the maximal overlap and exit if it exceeds a threshold you gave. It also compute a specific quantity to determine if two bonds cross during one time-step.

Source Code

The source code and more details can be find on the [velocities_resolve_EVC](#) GitLab repository.

Software Technical Information

Name Verlet_list_for_ODE

Language C/C++ and [Open-Dynamics-Engine](#) software API

Licence [MIT](#)

Documentation Tool Doxygen

Application Documentation [Doxygen documentation](#)

Relevant Training Material None

Software Module Developed by Pascal Carrivain

1.6.9 E-CAM `Verlet_list_for_ODE` module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The `Verlet_list_for_ODE` module introduces [Verlet-list](#) for the rigid-body dynamics [Open-Dynamics-Engine](#) software.

Purpose of Module

Rigid-body dynamics is useful for mechanical articulated systems. In addition, the tool allows the user to simulate complex shape and resolve excluded volume constraints. It is used in the industry of video games to accurately reproduce physics. However, a software like [Open-Dynamics-Engine](#) computes pairwise overlap every time-step. The engine starts with a partition of the space and then loops over all the blocks of partition. For each blocks it runs nested loops to check the overlaps between the objects inside the block.

The module implements external functions that can be used to compute the [Verlet-list](#). Therefore, the user does not call the pairwise overlap check every time-step. He only needs to loop over the [Verlet-list](#) with the pairwise objects within a given cut-off distance. However, the [Verlet-list](#) has to be updated according to the displacement length of the objects.

The module can be used to speed-up the [Open-Dynamics-Engine](#) simulation of polymers and complex objects system.

We test the module with two examples: chromatin fiber and bacterial circular DNA. Chromatin fiber is an assembly of DNA wrapped around [nucleosomes](#) that compact the genome. We model the DNA at the scale of 10.5 base-pair (one helix turn) as an articulated system. The nucleosomes is built with complex shape. We run a [Langevin dynamics](#) and check that our [Verlet-list](#) implementation gives the same results [Open-Dynamics-Engine](#) would give.

Background Information

You can find a detailed description on the [Verlet_list_for_ODE](#) [GitLab](#) repository.

Building and Testing

First of all you need to download and build the 0.16 version of [Open-Dynamics-Engine](#). You can find the steps on the [Verlet_list_for_ODE](#) [GitLab](#) repository.

In order to compile the two examples (tests) I provide a template `Makefile` you can find at the same location that the source code. You need C++11 in order to use pseudo-random number generator.

It has [OpenMP](#) acceleration, edit the `Makefile` to enable it. The example uses threads from `std` as well.

Before the compilation you can clean the previous build with `make mrproper` command. The [Verlet-list](#) implementation returns the number of collisions that can be confronted with the result returns by [Open-Dynamics-Engine](#).

Source Code

The source code and more information can be find at [Verlet_list_for_ODE](#) [GitLab](#) repository.

Software Technical Information

Name `openmm_copolymer`

Language Python 3.7, [OpenMM API](#)

Licence [MIT](#)

Documentation Tool [Sphinx](#)

Application Documentation [pydoc3.7](#)

Relevant Training Material https://gitlab.com/pcarrivain/openmm_copolymer

Software Module Developed by [Pascal Carrivain](#)

1.6.10 E-CAM openmm_copolymer module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

openmm_copolymer is a module that samples conformation of a [block-copolymer](#) given an *epigenome* state file. This module takes advantage of the [OpenMM API](#) and GPU acceleration. It builds a [Kremer-Grest](#) polymer model with uni-dimensional epigenetic information and constructs the epigenetic interactions based on the model you design. You simply need to feed the module with an *epigenome* state file, the interaction model and the mechanical properties of the polymer.

You can use the module to model small part of an *epigenome* or the whole genome confined inside the cell nucleus.

Purpose of Module

The epigenetic and the tri-dimensional structure of fly genome is studied by means of a [block-copolymer](#) (polymer made of more than one monomer species). The epigenetic information does not involve alterations in the DNA but [histone](#) tails modifications. This uni-dimensional information can be projected along the contour of a [block-copolymer](#) model.

Then, there is pairwise interaction of monomers according to the epigenetic states, leading to specific pattern of interactions. The interaction patterns can be visualized using contacts map: two-dimensional map with positions along the polymer and a third dimension with color scale for the intensity of contacts.

Since 2000, biologists can produce this same kind of data thanks to the *high-throughput-sequencing* methods 3C, 4C, 5C and Hi-C: [Chromosome-Conformation-Capture](#). Recently, biologists have shown that the interactions pattern is correlated with the epigenetic information. However, the strength and model of interactions between epigenetic states are not always clearly known.

In addition to the *high-throughput-sequencing* methods, we can study the spatial distances inside part of the genome with the help of [FISH](#) and high-resolution methods. All the spatial distances can be simply extracted from the model we built with the help of *openmm_copolymer* module.

The module we propose uses the [OpenMM API](#) with GPU acceleration to sample as many epigenetic parameters as possible.

It is possible to use effective interactions (gaussian overlap or [Lennard-Jones potential](#)) to model the epigenetics. The module introduces the possibility to replace effective epigenetic interactions with [binders model](#) too. In this case, the binder is like a protein that can bind to a specific site of the genome. A simple input file is enough to tell the script about the binder-binder and monomer-binder interactions.

The module includes pairing potential, nucleus confinement potential as-well-as genome examples.

It can be used by polymer physicists, biophysicists for epigenetic modeling, to understand the link between epigenetic and tri-dimensional structure of a genome, to estimate first-passage-time encounter of two loci. It is used in a scientific collaboration to study a specific promoter-enhancer system and [homeotic gene](#) complexes in the fruit-fly organism (ENS Lyon, France). However, the publication is not currently available.

Background Information

We use the [OpenMM API](#) for molecular dynamics. We implemented functionalities to build a [Kremer-Grest](#) polymer system with uni-dimensional epigenetic information. We also implement functions to build the quantities biologists extract from [high-throughput-sequencing](#) and [FISH](#) experiments.

You can find pdf file with a detailed description on the [openmm_copolymer GitLab repository](#). The module will be constantly improved with new functionalities.

Building and Testing

The instructions to install, test and run the module can be find on the [openmm_copolymer GitLab repository](#).

Source Code

The source code can be found on the [openmm_copolymer GitLab repository](#).

Software Technical Information

Name OpenMM_Plectoneme

Language Python 3.7, OpenMM API

Licence [MIT](#)

Documentation Tool Sphinx

Application Documentation [README.md](#)

Relevant Training Material [pdf documentation](#)

Software Module Developed by Pascal Carrivain

1.6.11 E-CAM openmm_plectoneme module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The `openmm_plectoneme` is a module that introduces twist rigidity to a polymer and samples the accessible conformations under torsional constraints. This module takes advantage of the OpenMM software and GPU acceleration to perform simulation at the scale of the DNA helix. It builds a *Kremer-Grest* polymer model with virtual sites to attach a frame to each bead. The frames are used to describe the contour of the molecule and to introduce bending and twisting forces.

Purpose of Module

Bacterial DNA is known to form specific conformations called *plectonemes* because of internal twisting constraints. This physical mechanism participates in the compaction of the genome. The *plectonemes* are braided structures you

often compare with phone cables. In order to study such a system we need to introduce a [linking number](#) deficit into a circular polymer.

The Linking number ($Lk = Tw + Wr$) is the sum of the twist (Tw , cumulative helicity of the DNA) and the writhe (Wr , global intracity). In the case of circular DNA that is topologically constrained any variation of the twist affects the Writhe and therefore the conformation.

In particular, does a slow change of the twist lead to the same conformation that the one we get from a rapidly change in the twist? We then tackle the question : does the introduction protocol of Linking number inside a circular molecule matter? Indeed, does a rapidly Linking number injection freeze the conformation in braided structures where *plectonemes* do not merge/move along the DNA ? Does the memory of initial conformation matter ?

We can use this module to model single-molecule DNA under [magnetic or optical tweezers](#) too. In this kind of setup the molecule is clamped on a plate and to a magnetic bead at the other extremity. The bead is used to apply stretching force and/or rotational constraint. The position of the bead is used to monitor the response of the molecule to the mechanical constraints. From the mechanical constraints you can extract the mechanical properties of your molecule of interest.

This module assist the creation of polymer described by [FENE bond](#) and [WCA repulsive potential](#) to resolve the excluded volume constraints.

On top of that, the module introduces the twist and mechanical response to twisting constraint with the help of *virtual sites* functionalities from OpenMM API. The module proposes functions to help the data analysis with High-Performance-Computing Dask software and Python module Numba.

For example, the estimation of the Writhe that is a computation over all the possible pairwise of bonds is highly expensive and can be fasten. In addition to that, we introduce an algorithm to detect the positions, length and shape of *plectonemes*. It is useful to follow the dynamics of these braided structures and try to answer the previous questions.

This module can be used by polymer physicist to understand the conformation of bacterial DNA under torsional constraints for example. Indeed, it used in a scientific collaboration with Ivan Junier from TIMC-IMAG, Grenoble, France and Ralf Everaers, ENS Lyon, France. However, the publication is not currently available.

Background Information

We use the OpenMM toolkit for molecular dynamics. We implemented functionalities to build a frame (that follows the contour of the polymer) and add twisting energy to a *Kremer-Grest* polymer system. We implemented function to extract *plectonemes*, [writhe](#) and [twist](#) from polymer conformations.

Building and Testing

The instructions to install, test and run the module can be find on the [openmm_plectoneme GitLab repository](#). The test of the twist implementation can be find at the same location.

We are currently working on a benchmark between the present module and already published [Monte-Carlo](#) and [rigid body dynamics](#) codes.

Source Code

The source code and more information can be find on the [openmm_plectoneme GitLab repository](#).

Software Technical Information

Name `polymer_data_analysis`

Language Python 3.7, Numba, Numpy, Dask

Licence MIT

Documentation Tool Sphinx

Application Documentation pydoc3.7

Relevant Training Material https://gitlab.com/pcarrivain/openmm_plectoneme/blob/master

Software Module Developed by Pascal Carrivain

1.6.12 E-CAM polymer_data_analysis module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The polymer_data_analysis module provides functions to compute quantities like gyration radius, internal distances, contact maps, contact probabilities, etc. This module takes advantage of the Numba parallelisation implementation and JIT (just-in-time) compilation. It also uses Dask to deploy numerous data analysis on job queuing systems.

Purpose of Module

The module takes advantage of the Python language as well as Numpy to write simple scripts that run complete data analysis of polymer systems. It also uses Numba to perform fast computation and easily handle nested loops.

However, computation of quantities like contact maps or internal distances needs an algorithm that scales like the square of the system size. If computation cannot use Numba, the modules proposes to deploy data analysis on job queuing systems.

You can also compute quantities like twist and writhe to study bacteria conformation.

We would like to provide a pipeline to help biophysicist extract quantities from simulations.

Background Information

The module uses Python language, Numpy, Numba and Dask.

Building and Testing

The instructions to install, test and run the module can be find on the data analysis GitLab repository.

Source Code

The source code and more information can be find on the data analysis GitLab repository.

Software Technical Information**Name** 2spaces_on_gpu**Language** C, OpenCL**Licence** MIT**Documentation Tool** Doxygen**Application Documentation** ‘https://gitlab.com/pcarrivain/2spaces_gpu/-/blob/master/latex/refman.pdf’**Relevant Training Material** not available yet**Software Module Developed by** Pascal Carrivain

1.6.13 E-CAM 2spaces_on_gpu module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The 2spaces_on_gpu module implements the 2-spaces algorithm on a GPU (see “Background Information” section). This algorithm is designed to move one-half of the polymer in one Monte-Carlo iteration. It also preserves the excluded volume constraints. An [OpenCL](#) implementation has been written so it can be used on CPUs or GPUs.

Purpose of Module

A polymer of size L is supposed to reach equilibrium after a time of order L^3 . Therefore, it could be difficult to study the equilibrium properties of large polymers. The 2-spaces algorithm improves the efficiency of each Monte-Carlo iteration by moving half of the polymer. We can use the GPUs to take care of one of the sub-moves in each Monte-Carlo step.

It is used in a scientific collaboration (ENS Lyon).

Background Information

Please consider reading the two research articles [Massively Parallel Architectures and Polymer Simulation](#) and [Cellular automata for polymer simulation with application to polymer melts and polymer collapse including implications for protein folding](#) for details about the method.

Building and Testing

I provide a simple `Makefile` as well as an [OpenCL](#) kernel and main source code to run the model. You need C++11 in order to use pseudo-random number generator. Before the compilation you can clean the previous build with the `make mrproper` command. Details about building, testing and running the code is available in the [2spaces_on_gpu GitLab repository](#).

Source Code

The source code and more information can be found on the [2spaces_on_gpu](#) GitLab repository.

1.7 Extended Software Development Workshops (ESDWs)

The first ESDW for the Classical MD workpackage was held in Traunkirchen, Austria, in November 2016, with a follow-up to be held in Vienna in April 2017. The following modules have been produced:

1.7.1 Transition State Ensemble in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (3.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: Sander Roet

This module is an addition to OpenPathSampling to calculate the snapshots that correspond to the transition state ensemble from a list of trajectories.

Purpose of Module

Often in transition path sampling we want to get an idea about the features of the transition. This is done by generating an ensemble of snapshots that correspond to a committor of approximately 50%. This ensemble gives information about the transition state and the shape of the barrier. This code provides a straightforward way of calculating this ensemble for a given list of trajectories.

This module tries to efficiently find a single transition state frame from each trajectory. This is done by bisection of the trajectory, depending on the current committor. For example, if the current committor is too high (too much ends up in state B) the next index is selected halfway towards the left edge and the current index is set as the new right edge. This is repeated until a committor within a given range is reached or no new frame can be selected.

In the end this module returns a dictionary of shape `{snapshot: comittor value}` which then can be used for analysis.

The implementation in this module includes:

- A `TransitionStateEnsemble` subclass of `PathSimulator` to run the transition state ensemble simulation.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

To test this module you need to download the source files package (see the `Source Code` section below) and install it using `pip install -e .` from the root directory of the package. In the `ops_tse/tests` folder type `nosetests test_ops_tse.py` to test the module using the `nose` package.

Examples

- An IPython 2-D toy example can be found in the `examples` directory of the the source files (see the `Source Code` section below). Open it using `jupyter notebook simple_tse_example.ipynb` (see Jupyter notebook documentation at <http://jupyter.org/> for more details)

Source Code

The source code for this module can be found in: https://gitlab.e-cam2020.eu/Classical-MD_openpathsampling/TSE/tree/master

1.7.2 Reactive flux in OpenPathSampling

Software Technical Information

The information in this section describes `OpenPathSampling` as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: Andreas Singraber

This module implements the reactive flux method in OpenPathSampling.

Purpose of Module

The reactive flux method in combination with a free energy calculation allows to derive the rate constant of a rare event. This is accomplished by a shooting algorithm similar to a committor analysis where fleeting trajectories starting from the dividing surface are generated and statistics about their state with respect to a collective variable is collected. There are many flavors of the reactive flux method, this module implements the effective positive flux method as described by van Erp and Bolhuis (see e.g. <http://dx.doi.org/10.1016/j.jcp.2004.11.003>).

The implementation introduces the following new classes:

- `ReactiveFluxSimulation` inherits from `ShootFromSnapshotsSimulation` and implements the shooting algorithm similar to `CommittorSimulation`. First, backward trajectories from the initial snapshots are started and followed until they either hit state A or recross the dividing surface. In the latter case the trajectory is rejected. If instead the trajectory reaches A, a forward shot is performed until the trajectory reaches either A (rejected) or B (accepted). The forward trajectory is allowed to recross the barrier any number of times but must end up in B without reaching A. To implement this behaviour of a forward shot depending on the final state of the backward trajectory a `NonCanonicalConditionalSequentialMover` and the `NonCanonicalConditionalSequentialMoveChange` were derived from existing classes available in OpenPathSampling. The stable states, the dividing surface and other regions are identified via a user-defined reaction coordinate and resulting trajectories are saved in a `Storage` object.
- The class `ReactiveFluxAnalysis` provides functionality to analyze previously generated and stored trajectories similar to its parent class `ShootingPointAnalysis`. In addition to trajectories the user needs to provide the gradient of the reaction coordinate at the dividing surface. With the stored velocities at the trajectory starting points it is possible to compute the time derivate of the reaction coordinate and therefore (together with results from a free energy calculation) derive the total flux and the flux for each initial snapshot. Methods to visualize e.g. per-snapshot results in 1D- and 2D-histograms are also provided.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

To test this module you need to download the source files package (see the `Source Code` section below) and install it using `python setup.py install` from the root directory of the package. In the `ops_rf/tests` folder type `nosetests testrfanalysis.py` to test the module using the `nose` package.

Examples

See the `rf-example.ipynb` IPython notebook in the source directory, here is the direct link: https://gitlab.e-cam2020.eu/Classical-MD_openpathsampling/RF/blob/master/ops_rf/rf-example.ipynb To run the example execute `jupyter notebook rf-example.ipynb` in your terminal.

Source Code

The source code for this module can be found in: https://gitlab.e-cam2020.eu/Classical-MD_openpathsampling/RF/tree/master

1.7.3 Maximum likelihood optimization of reaction coordinates

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material https://gitlab.e-cam2020.eu/Classical-MD_openpathsampling/MaxLikelihood/tree/master/examples

Licence LGPL v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: Clemens Moritz and Raffaella Cabriolu

This module implements an OpenPathSampling library that provides a maximum likelihood analysis to obtain an optimal reaction coordinate by combining multiple collective variables.

Purpose of Module

OpenPathSampling (OPS) is a software package that simulates complex processes using path sampling techniques and yields reactive trajectories between states of interest in a given system. However, such trajectories do not automatically lead to a physical understanding of the reaction mechanism. To gain such an understanding it is desirable to find a set of collective variables (CVs) that carry physically important information about the process.

The size and the shape of a crystalline cluster in a freezing liquid, the number of native contacts in a folding protein or bond length and bond angles in chemical reactions are examples of such CVs. The aim of this module is to find

an optimized combination of multiple CVs into a single coordinate, that monitors the progress of the reaction. Such a coordinate is commonly called a reaction coordinate.

In methods used to study complex processes, having a good reaction coordinate either significantly improves their efficiency or it is a prerequisite for the reliability of their results.

The reaction coordinate is constructed by optimizing the likelihood function

$$L = \prod_{\text{yes}} p(r(q_i)) \prod_{\text{no}} (1 - p(r(q_i))),$$

where r is a reaction coordinate model that combines several CVs, q_i , into a reaction coordinate and p is the probability model that maps this coordinate to a probability of having a successful outcome (yes). The definition of a successful outcome depends on the chosen probability model. Both r and p depend on a set of coefficients that are used to maximize L .

For more details on the method, please refer to the references given in *Background Information*.

Classes and objects implemented in this module:

- `TargetFunctionDescription` class. Wrapper around functions that carries additional information such as the number of parameters which shall be varied during subsequent optimization.
- `REACTION_COORDINATE_MODELS` dictionary of objects of the class `TargetFunctionDescription`. Collection of commonly used reaction coordinate models. At the moment two combinations of collective variables are available: a linear function, and a quadratic function. Additionally the user can define custom functions.
- `PROBABILITY_MODELS` dictionary of objects of the class `TargetFunctionDescription`. Collection of commonly used probability models. At the moment two functions are available: a sigmoidal function as a model for committor probabilities and a symmetric peaked function as a model for the probability of finding a transition path starting from the configuration r .
- `MaxLikelihoodCVAnalysis` class. It implements the maximum likelihood analysis. There are two methods implemented: one for optimization based on committor probabilities, using an `openpathsampling.ShootingPointAnalysis` object (`optimize_from_spa`), and an other one where the user can perform optimizations for custom problems, using the `optimize` method.

Background Information

This module is built on the OpenPathSampling library. More information about it are given in the following links:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Information about the method can be found in these publications:

- Peters, B. & Trout, B. L. "Obtaining reaction coordinates by likelihood maximization." J. Chem. Phys. 125, 54108 (2006).
- Peters, B., Beckham, G. T. & Trout, B. L. "Extensions to the likelihood maximization approach for finding reaction coordinates." J. Chem. Phys. 127, 34109 (2007).
- Peters, B. "Reaction Coordinates and Mechanistic Hypothesis Tests." Annu. Rev. Phys. Chem. 67, annurev-physchem-040215-112215 (2016).

Testing

To test this module you need to download the source files package (see the `Source Code` section below) and install it using `python setup.py install` from the root directory of the package. In the `ops_maxlikelihood/tests` folder type `nosetests` to test the module using the `nose` package.

Examples

The example of the Maximum Likelihood module on the 2D toy model implemented in OPS is given in the directory `examples`. Open it using `jupyter notebook ExampleMaximumLikelihood2DToyModel.ipynb` (see Jupyter notebook documentation at <http://jupyter.org/> for more details).

Source Code

Source code can be found at https://gitlab.e-cam2020.eu/Classical-MD_openpathsampling/MaxLikelihood

1.7.4 Interface Optimization

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Author: Anastasiia Maslechko

This module consist of functions to evaluate the new values of the interface positions in Transition Interface Sampling (TIS) calculation. Two approaches are implemented (for more details please look in the *Background Information*).

Purpose of Module

In TIS calculations the full path space is investigated through several ensembles by putting the interfaces. For each interface during the simulation crossing probability is evaluated. At the end the reaction rate depends on the crossing probabilities obtained from each ensemble. Without a full knowledge of the reaction mechanism and information about free energy profile user most likely fails to set up manually initial setting, which are going to lead to the efficient usage of the software resources, and most importantly to low statistical errors in the simulation run.

One of the challenges in Transition Path Sampling is to define a proper *collective variable* (CV). Interface-optimization module is a way to improve code performance within selected CV.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

In the Interface Optimization module a search of the optimal set of the interfaces is done in iterative manner. The key idea is described in the papers (please, see the references below), and the goal is to approach almost the same crossing probabilities in each ensemble:

- Ernesto E. Borrero and Fernando A. Escobedo. Optimizing the sampling and staging for simulations of rare events via forward flux sampling schemes. The Journal of Chemical Physics 129, 024115 (2008); doi: <http://dx.doi.org/10.1063/1.2953325>
- Ernesto E. Borrero, Marcus Weinwurm, and Christoph Dellago. Optimizing transition interface sampling simulations. The Journal of Chemical Physics 134, 244118 (2011); doi: <http://dx.doi.org/10.1063/1.3601919>

The first thing to do is to start the simulation for some number of cycles with initial guess (set of interfaces). On the basis of the obtained data analysis is done as follows: values of the current interface positions and the “crossing probabilities” (these are most likely non-converged values due to the least number of cycles) are used in the interface-optimization calculation. Note: termination of the simulation and getting of the interfaces and crossing probabilities must be done by an experimenter.

The next step is to build a mapping function f , which satisfies several properties:

1. The form of the function helps to “equalize” the crossing probabilities on the next simulation run.
2. It is bijective.
3. It is monotonic in order to have inverse image.

The next step is to find the number of the interfaces (n_{int}) to satisfy users requirements. After that we need to divide a region from the starting point til 1 on the ordinate on the $n_{int} - 1$ intervals: by mapping of the obtained points on the function curve we can get a new set of interfaces.

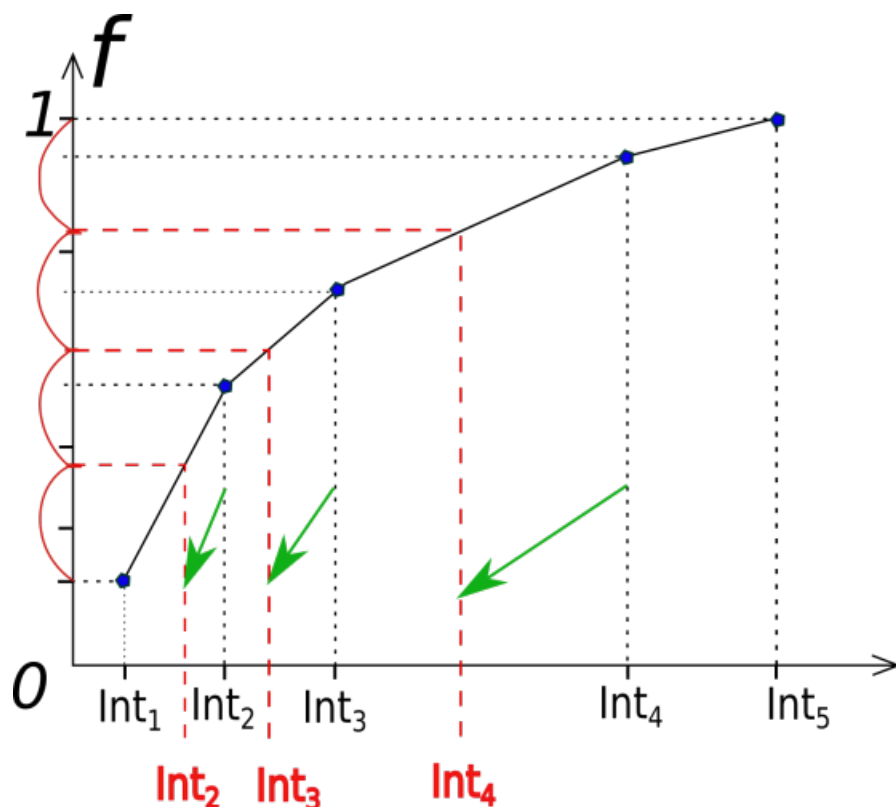
Important functions implemented in this module:

- `find_interface_and_cross`: reads the output of the analyze-tool after a certain number of cycles (defined by the user). Object like *TISTransition* must be used as an input.
- `save_n_interfaces`: a method to calculate the new interface positions, using first approach: the number of interfaces kept the same, only the values of their positions are going to be changed.
- `save_p_interfaces`: a method to calculate the new interface positions, using second approach: the number of the interfaces might be changed, “expected” crossing probability for each ensemble has a predefined value.

Testing

Tests in OpenPathSampling use the `nose` package.

The tests for this module can be run by downloading its source code (check the section `Source Code` below), installing its requirements, and running the command `nosetests` in a root directory (or in particular for the file `test_interface_optimization.py`).



Examples

To check an example look for the file `toy_mistis_interface_optimization.ipynb` in the source code (or in the directory `toy_model_mistis` from the OPS-examples section). To open it use `jupyter notebook toy_mistis_interface_optimization.ipynb` (see Jupyter notebook documentation at <http://jupyter.org/> for more details). In order to run it download the data from http://www.dropbox.com/s/qaeczkugwxkrdfy/toy_mistis_1k OPS1.nc, as described inside of the example-file.

Source Code

The source code for this module can be found in: https://gitlab.e-cam2020.eu:10443/Classical-MD_openpathsampling/Interface.

The second ESDW for the Classical MD workpackage was held in Leiden, Holland, in August 2017. The following modules have been produced:

1.7.5 Spring Shooting in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7, 3.7, 3.8, 3.9)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence MIT

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: Sander Roet

This module implements the spring shooting method in OpenPathSampling

Purpose of Module

Transition path sampling is most efficient when paths are generated from the top of the free energy barrier. However, complex (biomolecular) activated processes, such as nucleation or protein binding/unbinding, can have asymmetric and peaked barriers. Using uniform selection on these type of processes will not be efficient, as it, on average, results in selected points that are not on the top of the barrier. Paths generated from these points have a low acceptance probability and accepted transition paths decorrelate slowly, resulting in a low overall efficiency. Spring shooting was developed to increase the efficiency of path sampling of these types of barriers, without any prior knowledge of the barrier shape. The spring shooting algorithm uses a shooting point selector that is biased with a spring potential. This bias pulls the selection of points towards the transition state at the top of the barrier. The paths that are generated from points selected by this biased selector therefore have an increased acceptance probability and decorrelation between accepted transition paths is also increased. This results in a higher overall efficiency. The spring shooting algorithm is described by Brotzakis and Bolhuis (<http://dx.doi.org/10.1063/1.4965882>).

In summary, the spring shooting selection algorithm is a selector for the one-way shooting method for transition path sampling, which uses a bias. This bias is of the shape $\min[1, e^{s\kappa\Delta\tau}]$. Where $s = -1$ for forward shooting and $s = 1$ for backward shooting, κ is the given spring constant and $\Delta\tau = \tau' - \tau$ is the number of shifted frames of the new shooting point τ' compared to the previous accepted shooting point τ . The choice of τ' is limited to the interval $[-\Delta\tau_{max}, \Delta\tau_{max}]$. The shooting move is rejected if a τ' is selected that is outside of the current path and is accepted if the trajectory satisfies the path ensemble.

The main difference of this module compared to the paper is that instead of using a rejection algorithm to sample from the correct distribution, the correct distribution is sampled directly.

The implementation introduces the following new classes:

- `SpringShootingSelector` inherits from `ShootingPointSelector` and implements the shooting point selection, using a bias of the shape $\min[1, e^{s\kappa\Delta\tau}]$. At initialization it also takes an `initial_guess` as the initial reference point. This will default to `floor(len(trajectory)/2)`. To correctly keep track of the history the selector has to be the same instance for the forward and backward mover. The `pick` function has to be called with a direction in order to be able to use the correct value for s . The selector then draws a random number in the range $[0, 1)$, multiplies it with the sum of the biases. It then sums the biases from $-\Delta\tau_{max}$ to $\Delta\tau_{max}$ and returns the index when this sum is bigger than the random number. The `restart_from_step` function makes it possible to restart the selector at a specific step. It takes an `MCStep` and reconstructs the correct history from the `MoveDetails`.

- `SpringMover` inherits from `EngineMover` and is the parent class for the `ForwardSpringMover` and `BackwardSpringMover` classes. It calls the `selector.pick` function with `self.direction` to get the correct shooting point. It also build adds the needed `MoveDetails` in order to be able to restart the selector. If an invalid snapshot has been chosen it will not run dynamics but will build a `Sample` with an acceptance probability of 0.0.
- `SpringShootingMover` inherits from `SpecializedRandomChoiceMover` and behaves similar, except it takes the extra arguments: `delta_max`, `k_spring` and `initial_guess`. These are then used to make the `SpringShootingSelector` and this selector is given to both the forward and backward mover.
- `SpringShootingStrategy` inherits from `SingleEnsembleMoveStrategy` and will make the `SpringShootingMover` for every ensemble, with the given values of `delta_max`, `k_spring` and `initial_guess`.
- `SpringShootingMoveScheme` inherits from `MoveScheme` and it will use the `SpringShootingStrategy` to build all the necessary movers for the given network, with the given values of `delta_max`, `k_spring` and `initial_guess`. Also the functions `to_dict` and `from_dict` have been adapted to save and load with all the data

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

This module has been included in the `OpenPathSampling` core. Its tests can be run by setting up a developer install of `OpenPathSampling` and running the command `py.test` from the root directory of the repository.

Examples

- An example on how to set up a simulation using the `SpringShootingMoveScheme` and the comparison with the `UniformSelector` can be found in `spring_shooting_example.ipynb` in the `examples/misc` directory (<https://github.com/openpathsampling/openpathsampling/tree/master/examples/misc>). Open it using jupyter notebook `spring_shooting_example.ipynb` (see Jupyter notebook documentation at <http://jupyter.org/> for more details)

Source Code

This module has been merged into `OpenPathSampling`. It was added in the following pull request:

- <https://github.com/openpathsampling/openpathsampling/pull/850>

1.7.6 OPS-based module: Shooting range shooter

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7) [+Python (3.6)]

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

This module implements the “shooting from the top” algorithm as detailed in the paper “Transition path sampling of rare events by shooting from the top”.

Purpose of Module

The purpose of this algorithm is to increase the number of generated transitions in a transition path sampling simulation by exclusively shooting from the transition state ensemble (TSE)/the top of the barrier (hence the name). Naturally this only works if the approximate location of the TSE is already known and can be given as a function of the atomic coordinates. In this module any `openpathsampling.Volume` object can be used by the user to define the shooting range volume. This enables the user to define the shooting range for example as a function of one or more collective variables. See also the *Transition State Ensemble in OpenPathSampling* for finding the TSE.

The implementation in this module includes:

- A `ShootingRangeSelector` subclass of `openpathsampling.ShootingPointSelector` to pick shooting points only in the predefined shooting range volume.

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in OpenPathSampling and `sr_shooter` use the `nose` package.

To test this module you need to first install OpenPathSampling, then download the source files for this package (see the *Source Code* section below) and install it using `python setup.py install` or `pip install -e .`

from the root directory of the package. In the root folder then type `nosetests` to test the module using the `nose` package.

Examples

There are two [example jupyter notebooks](#) in the example directory of the repository:

One shows the [general setup of a two way shooting transition path sampling with a shooting range on a toy system](#).

The other is a [comparison between one way shooting and two way shooting from the shooting range](#) and shows that path space is explored faster with two way shooting when using a (well placed) shooting range. The reason being that the shots initiated at the barrier top have a high probability of success and two way shooting decorrelates faster (if using randomized velocities even faster).

Source Code

The source code for this module can be found in https://gitlab.e-cam2020.eu/hejung/sr_shooter.

1.7.7 Web Throwing in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7, 3.7, 3.8, 3.9)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: Sander Roet, Anders Lervik, Enrico Riccardi

This module implements the web throwing method in OpenPathSampling

Purpose of Module

Web throwing is a Monte Carlo move in path space designed to improve the efficiency of transition interface sampling (TIS). It consists of a smart selection of shooting points and shooting moves that respects super detailed balance. The web throwing algorithm generates paths that have a much lower correlation with their original paths, even if it is more

computational expensive than standard shooting. The strategy thus can significantly reduce the computational time required to study transition events and to quantify their rates. The web throwing algorithm is described by Riccardi, Dahlen, and van Erp (<http://dx.doi.org/10.1021/acs.jpcclett.7b01617>)

The web throwing method is an shooting method for transition interface sampling that decorrelates the trajectory between an interface λ and an associated surface of unlikely return λ_{SOUR} . It does this by doing `n_cycles` of Transition Path Sampling (TPS) from λ_{SOUR} to λ . The first frame in this volume is selected for a forwards shot and the last frame in this volume is selected for a backwards shot. After the `n_cycles` the new sub-trajectory is extended in both ways to satisfy the TIS ensemble. The sampling efficiency is increased significantly if λ_{SOUR} and λ are positioned before and after the barrier in the potential, respectively.

The implementation introduces the following new classes:

- `SecondFrameSelector` inherits from `ShootingPointSelector` and selects the second frame (`index = 1`) from a trajectory.
- `SecondToLastFrameSelector` inherits from `ShootingPointSelector` and selects the second to last frame of a trajectory (`index = len(trajectory)-2`).
- `WebEnsemble` inherits from `SequentialEnsemble` and implements the sampling ensemble between the Volume λ_{SOUR} and the Volume λ in which the web throwing occurs.
- `WebBackwardExtendEnsemble` inherits from `SequentialEnsemble` and implements the ensemble the partial path has to fit in after the backwards extend. This ensemble consists of one frame in the initial state followed by frames outside of any states. This should only be used in backward moves, as this only has an acceptable stop in this direction.
- `WebShootingMover` inherits from `RandomChoiceMover` and implements the TPS shooting algorithm in the `WebEnsemble` using the `SecondFrameSelector` for the `ForwardShootMover` and the `SecondToLastFrameSelector` for the `BackwardShootMover`.
- `WebCycleMover` inherits from `SequentialMover` and implements `n_cycles` sequential `WebShootingMover` in the `WebEnsemble`.
- `WebExtendMover` inherits from `ConditionalSequentialMover` and extend the sub trajectory after web throwing. It first simulates backwards to reach the initial TIS state. If this reaches any other state, the whole move is rejected. If this indeed finds the correct state the sub trajectory is extended in the forward direction to complete the TIS ensemble.
- `WebThrowingMover` inherits from `SubPathMover` and is the canonical mover for the web throwing algorithm. From a TIS ensemble it first selects a random sub-trajectory in the `web_ensemble`. Then it calls the `WebCycleMover` with `n_cycles`. Finally it calls the `WebExtendMover` to extend the sub-trajectory back into the TIS ensemble.
- `WebThrowingStrategy` inherits from `MoveStrategy` and it builds the `WebThrowingMover` for every `{interface: lambda_sour}` in the `sours` dictionary. Every interface should have only 1 `lambda_sour` associated with it.
- `WebThrowingScheme` inherits from `MoveScheme`. This is a `MoveScheme` that only does web throwing. This is mostly a convenience class used in examples and testing.

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

The tests for this module can be run by installing `OpenPathSampling`, downloading source code for the module (see the `Source Code` section below), and installing it by running `python setup.py install` from the root directory of the package. Test this module with the `nose` package, by running the command `nosetests` from the root directory of the repository.

Examples

All examples can be found in the `examples` directory (https://gitlab.e-cam2020.eu/sroet/web_throwing/tree/master/examples). Open the ipython notebook (files with the extension `.ipynb`) of interest by running `jupyter notebook file_of_interest.ipynb` in this directory, replacing `file_of_interest.ipynb` by the file you want to open. (see Jupyter notebook documentation at <http://jupyter.org/> for more details)

The examples are:

- An example on how to set up a multiple interface set TIS (MISTIS) simulation with only web throwing and the analysis of the web throwing moves can be found in `mistis_using_only_webthrowing.ipynb`.
- An example on how to add the web throwing to a default MISTIS simulation can be found in `adding_webthrowing_to_mistis.ipynb`. This example also shows how to change the selection weight of this move and some analysis.

Source Code

The source code for this module can be found in: https://gitlab.e-cam2020.eu/sroet/web_throwing/tree/master.

1.7.8 PLUMED Wrapper for OpenPathSampling

Software Technical Information

The information in this section describes `OpenPathSampling` as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: Alberto Pérez de Alba Ortíz

This module interfaces OpenPathSampling (OPS) with PLUMED, an open-source library with a rich catalogue of Collective Variables (CVs).

Special thanks to Gareth A. Tribello for facilitating the use of the PLUMED Cython wrapper.

- G.A. Tribello, M. Bonomi, D. Branduardi, C. Camilloni, G. Bussi, PLUMED2: New feathers for an old bird, Comp. Phys. Comm. 185, 604 (2014); <https://doi.org/10.1016/j.cpc.2013.09.018>

Purpose of Module

Transition path sampling simulations and analysis rely on accurate state definitions. Such states are typically defined as volumes in a CV-space. OPS already supports a number of CVs, including the ones defined in the MDTraj Python library. PLUMED, an open-source C++ library, offers a wide variety of extra CVs, which are enabled in OPS by this module.

Many of PLUMED's dozens of CVs have a biomolecular focus, but they are also general enough for other applications. PLUMED's popularity (over 500 citations in 4 years after the release of PLUMED2) is greatly based on the fact that it works with many MD codes. OPS is now added to that list. The PLUMED code is well-maintained and documented for both users and developers. Several tutorials and a mailing list are available to address FAQs. For more information about the PLUMED code, visit: <http://www.plumed.org/home>

In this module, the class `PLUMEDInterface` is a subclass of the Cython wrapper class `Plumed` contained in the PLUMED installation. For initialization, `PLUMEDInterface` requires an `MDTrajTopology` and accepts additional PLUMED settings:

- `pathtoplumed=""` is the path to the PLUMED installation, where the `sourceme.sh` script is run to set all relevant flags. By default, the string is empty and the currently sourced PLUMED is used.
- `timestep=1.` is the time step size in PLUMED units (ps).
- `kbt=1.` is $k_B T$ in PLUMED units (kJ/mol).
- `molinfo=""` is a file to be used as `STRUCTURE` for the `MOLINFO` PLUMED command. It allows to provide extra information about the molecules. Consult: https://plumed.github.io/doc-v2.4/user-doc/html/_m_o_l_i_n_f_o.html
- `logfile=plumed.log` is the name of the log file written by the `PLUMEDInterface`.

The initialized `PLUMEDInterface` can be subsequently used to make functions that calculate CVs for a given Trajectory. This is done via the `PLUMEDCV` class, a subclass of `CoordinateFunctionCV`.

In PLUMED input files, a common syntax is: `label: keywords`. The class `PLUMEDCV` takes `name` and `definition` as arguments, which are respectively equivalent to PLUMED's `label` and `keywords`. The `PLUMEDCV` class also takes the `PLUMEDInterface` as argument. This allows for a single `PLUMEDInterface` to contain the `MDTrajTopology`, additional PLUMED keywords and previously defined CVs that can be reused for the same system. Both `PLUMEDInterface` and `PLUMEDCV` are storable.

This module supports (as listed in PLUMED documentation):

- **Groups and Virtual Atoms:** are directly set in the `PLUMEDInterface` via the `PLUMEDInterface.set(name, definition)` function. The `PLUMEDInterface.get()` function allows to consult the commands that have been already set. Some commands do not need a name, while some others must be run before any other command (e.g. UNITS).
- **CV Documentation:** all CVs are created by calling `PLUMEDCV(name, PLUMEDInterface, definition)`. The returned function can be applied to a Trajectory. CVs with components should specify the `components=["c1", "c2", "c3", ...]` keyword and the corresponding PLUMED keywords in the definition.

- Distances from reference configurations: also created by calling `PLUMEDCV(name, PLUMEDInterface, definition)`. Most of them require external files with reference configurations.
- Functions: also created by calling `PLUMEDCV(name, PLUMEDInterface, definition)`. They should be created using the same `PLUMEDInterface` that contains the previously defined CVs that are part of the function.
- Multicolvar and Exploiting contact matrices are not tested.

For examples see the `Examples` section below.

For further PLUMED usage details see: <http://plumed.github.io/doc-master/user-doc/html/index.html>

Background Information

This module builds on `OpenPathSampling`, a Python package for path sampling simulations. To learn more about `OpenPathSampling`, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Tests in `OpenPathSampling` use the `nose` package.

The tests for this module can be run by downloading its source code (see the `Source Code` section below), installing its requirements, and running the command `nosetests` from the root directory of the repository.

Examples

- Examples on how create and calculate PLUMED CVs can be found in `plumed_wrapper_example.ipynb` in the `examples` directory (https://gitlab.e-cam2020.eu/apdealbao/plumed_wrapper/tree/master/plumed_wrapper/examples). Open it using jupyter notebook `plumed_wrapper_example.ipynb` (see <http://jupyter.org/> for more details).

Source Code

The source code for this module can be found in: https://gitlab.e-cam2020.eu/apdealbao/plumed_wrapper/tree/master

It can be installed by running `pip install -e .` from the root directory of the package.

It requires to have the PLUMED development version (with the Cython wrapper) installed from: <https://github.com/plumed/plumed2>; and to source the file `/path/to/plumed2/sourceme.sh`

For details on PLUMED installation, see: http://plumed.github.io/doc-master/user-doc/html/_installation.html

Before using this module, please test the Cython PLUMED wrapper by attempting to import `plumed` in Python. If this is not successful, please refer to PLUMED installation documentation (above), or to the mailing list: <https://groups.google.com/forum/#!forum/plumed-users>

1.7.9 S-shooting in OpenPathSampling

Software Technical Information

The information in this section describes OpenPathSampling as a whole. Information specific to the additions in this module are in subsequent sections.

Language Python (2.7)

Documentation Tool Sphinx, numpydoc format (ReST)

Application Documentation <http://openpathsampling.org>

Relevant Training Material <http://openpathsampling.org/latest/examples/>

Licence LGPL, v. 2.1 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Examples*
- *Source Code*

Authors: Andreas Singraber

This module implements the S-shooting method¹ in OpenPathSampling.

Purpose of Module

S-shooting¹ is a recently developed method to determine rate constants of rare events. It is similar in spirit to the reactive flux method but its relaxed requirements help to overcome practical problems. The method is based on a simple shooting algorithm where trajectories are propagated forward and backward in time for a fixed number of timesteps. The starting points need to be provided and must lie in the saddle point region. This so-called S region (hence the name S-shooting) is defined via a suitable reaction coordinate and must to separate the stable states A and B in such a way that no trajectory can connect A with B without visiting S. In contrast to the reactive flux method the time derivative of the reaction coordinate is not required, which makes this approach applicable to systems exhibiting diffusive dynamics along the reaction coordinate. The S-shooting method can also be applied if the initial shooting points are taken from a biased simulation. Thus, it is a natural follow-up to free energy calculations like umbrella sampling and, in combination with free energy curves, allows the computation of rate constants.

The implementation of the S-shooting method in OpenPathSampling (OPS) is split into two main parts:

- Forward and backward trajectories started from initial snapshots are harvested and glued together calling the `SShootingSimulation` class. The user needs to provide the initial snapshots, a suitable definition of the S region and the desired trajectory length.
- The S-shooting analysis is performed upon calling the `SShootingAnalysis` class. Mandatory arguments include the definition of the stable states (A and B) and of the S region. In case the initial snapshots are taken from a biased simulation a bias function may be provided as an optional argument.

This module comes also with an IPython example notebook demonstrating the method by applying it to a one-dimensional system (a brownian walker in a double-well potential).

¹ Menzl, G., Singraber, A. & Dellago, C. S-shooting: a Bennett–Chandler-like method for the computation of rate constants from committor trajectories. Faraday Discuss. 195, 345–364 (2017), <https://doi.org/10.1039/C6FD00124F>

Background Information

This module builds on OpenPathSampling, a Python package for path sampling simulations. To learn more about OpenPathSampling, you might be interested in reading:

- OPS documentation: <http://openpathsampling.org>
- OPS source code: <http://github.com/openpathsampling/openpathsampling>

Testing

Follow these steps to test the module:

1. Download and install OpenPathSampling (see <http://openpathsampling.org/latest/install.html>).

Caution: This module has been developed alongside a specific OPS version available at that time. If incompatibilities arise as OPS is further enhanced, please use version 0.9.5 available here: <https://github.com/openpathsampling/openpathsampling/releases/tag/v0.9.5>.

2. Install the `nose` package.
3. Download the source files of the module (see the *Source Code* section below).
4. Install the module: change to the `S-Shooting` directory and run `python setup.py install`.
5. Run the tests: execute `nosetests` in the `S-Shooting` directory.

Examples

See the `sshooting-example.ipynb` IPython notebook in the source directory, here is the direct link: https://gitlab.e-cam2020.eu/singraber/S-Shooting/blob/master/ops_s_shooting/sshooting-example.ipynb To run the example execute `jupyter notebook sshooting-example.ipynb` in your terminal.

Source Code

The source code for this module is located here: <https://gitlab.e-cam2020.eu/singraber/S-Shooting>

Tip: Ultimately, this module will be merged into the official OPS code. Check the status of the corresponding pull request here: <https://github.com/openpathsampling/openpathsampling/pull/787>.

The third ESDW for the Classical MD work package was held in Turin, Italy in July 2018. The following have been produced as a result:

Software Technical Information

Name `pyscal`

Language Python (2.7, 3.4, 3.5, 3.6)

Licence GNU General Public License v3.0

Documentation Tool Sphinx/RST

Application Documentation <https://pyscal.readthedocs.io/en/latest/>

Relevant Training Material <https://mybinder.org/v2/gh/srmnitc/pyscal/master?filepath=examples%2F>

Software Module Developed by Sarath Menon Grisell Díaz Leines Jutta Rogal

1.7.10 pyscal

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

pyscal is a python module for the calculation of local atomic structural environments including Steinhardt's bond orientational order parameters¹ during post-processing of atomistic simulation data. The core functionality of pyscal is written in C++ with python wrappers using **pybind11** which allows for fast calculations and easy extensions in python.

Purpose of Module

Steinhardt's order parameters are widely used for the identification of crystal structures³. They are also used to distinguish if an atom is in a solid or liquid environment⁴. **pyscal** is inspired by the **BondOrderAnalysis** code, but has since incorporated many additional features and modifications. The **pyscal** module includes the following functionalities:

- calculation of Steinhardt's order parameters and their averaged version².
- links with the **Voro++** code, for the calculation of Steinhardt parameters weighted using the face areas of Voronoi polyhedra³.
- classification of atoms as solid or liquid⁴.
- clustering of particles based on a user defined property.
- methods for calculating radial distribution functions, Voronoi volumes of particles, number of vertices and face area of Voronoi polyhedra, and coordination numbers.

Background Information

See the [application documentation](#) for full details.

The utilisation of Dask within the project came about as a result of the **E-CAM High Throughput Computing ESDW** held in Turin in 2018 and 2019.

Building and Testing

Installation

¹ Steinhardt, P. J., Nelson, D. R., & Ronchetti, M. (1983). *Physical Review B*, 28.

³ Mickel, W., Kapfer, S. C., Schröder-Turk, G. E., & Mecke, K. (2013). *The Journal of Chemical Physics*, 138.

⁴ Auer, S., & Frenkel, D. (2005). *Advances in Polymer Science*, 173.

² Lechner, W., & Dellago, C. (2008). *The Journal of Chemical Physics*, 129.

pyscal can be installed directly using [Conda](#) by the following statement-

```
conda install -c pyscal pyscal
```

pyscal can be built from the repository by-

```
git clone https://github.com/srmnitc/pyscal.git
cd pyscal
python setup.py install --user
```

Testing

pyscal contains automated tests which use the [pytest](#) python library, which can be installed by `pip install pytest`. The tests can be run by executing the command `pytest tests/` from the main code directory.

Examples

Examples using pyscal can be found [here](#). An [interactive notebook](#) using binder is also available.

Source Code

The [source code](#) of the module can be found on GitHub.

Software Technical Information

Name jobqueue_features

Language Python

Licence [MIT](#)

Documentation Tool In-source documentation

Application Documentation Not currently available.. Example usage provided.

Relevant Training Material Not currently available.

Software Module Developed by Adam Włodarczyk (Wrocław Centre of Networking and Supercomputing), Alan O’Cais (Juelich Supercomputing Centre)

1.7.11 E-CAM High Throughput Computing Library

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

E-CAM is interested in the challenge of bridging timescales. To study molecular dynamics with atomistic detail, timesteps must be used on the order of a femtosecond. Many problems in biological chemistry, materials science, and other fields involve events that only spontaneously occur after a millisecond or longer (for example, biomolecular conformational changes, or nucleation processes). That means that around 10^{12} time steps would be needed to see a single millisecond-scale event. This is the problem of “rare events” in theoretical and computational chemistry.

Modern supercomputers are beginning to make it possible to obtain trajectories long enough to observe some of these processes, but to fully characterize a transition with proper statistics, many examples are needed. In order to obtain many examples the same application must be run many thousands of times with varying inputs. To manage this kind of computation a task scheduling high throughput computing (HTC) library is needed. The main elements of mentioned scheduling library are: task definition, task scheduling and task execution.

While traditionally an HTC workload is looked down upon in the HPC space, the scientific use case for extreme-scale resources exists and algorithms that require a coordinated approach make efficient libraries that implement this approach increasingly important in the HPC space. The 5 Petaflop booster technology of [JURECA](#) is an interesting concept with respect to this approach since the offloading approach of heavy computation marries perfectly to the concept outlined here.

Purpose of Module

This module is the first in a sequence that will form the overall capabilities of the library. In particular this module deals with creating a set of decorators to wrap around the [Dask-Jobqueue](#) Python library, which aspires to make the development time cost of leveraging it lower for our use cases.

Background Information

The initial motivation for this library is driven by the ensemble-type calculations that are required in many scientific fields, and in particular in the materials science domain in which the E-CAM Centre of Excellence operates. The scope for parallelisation is best contextualised by the [Dask](#) documentation:

A common approach to parallel execution in user-space is task scheduling. In task scheduling we break our program into many medium-sized tasks or units of computation, often a function call on a non-trivial amount of data. We represent these tasks as nodes in a graph with edges between nodes if one task depends on data produced by another. We call upon a task scheduler to execute this graph in a way that respects these data dependencies and leverages parallelism where possible, multiple independent tasks can be run simultaneously.

Many solutions exist. This is a common approach in parallel execution frameworks. Often task scheduling logic hides within other larger frameworks (Luigi, Storm, Spark, IPython Parallel, and so on) and so is often reinvented.

Dask is a specification that encodes task schedules with minimal incidental complexity using terms common to all Python projects, namely dicts, tuples, and callables. Ideally this minimum solution is easy to adopt and understand by a broad community.

While we were attracted by this approach, Dask did not support *task-level* parallelisation (in particular multi-node tasks). We researched other options (including Celery, PyCOMPSs, IPyParallel and others) and organised a workshop that explored some of these (see <https://www.cecarn.org/workshop-0-1650.html> for further details).

Building and Testing

The library is a Python module and can be installed with

```
python setup.py install
```

More details about how to install a Python package can be found at, for example, [Install Python packages on the research computing systems at IU](#)

To run the tests for the decorators within the library, you need the `pytest` Python package. You can run all the relevant tests from the `jobqueue_features` directory with


```
pytest tests/test_decorators.py
```

Examples of usage can be found in the `examples` directory.

Source Code

The latest version of the library is available on the [jobqueue_features GitHub repository](#), the file specific to this module is `decorators.py`.

(The code that was originally created for this module can be seen in the specific commit [4590a0e427112f](#) which can be found in the original private repository of the code.)

Software Technical Information

Name `jobqueue_features`

Language Python, YAML

Licence [MIT](#)

Documentation Tool In-source documentation

Application Documentation Not currently available. Example usage provided.

Relevant Training Material Not currently available.

Software Module Developed by Adam Włodarczyk (Wrocław Centre of Networking and Supercomputing), Alan O’Cais (Juelich Supercomputing Centre)

1.7.12 HTC Library Configuration in YAML

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module is the second in a sequence that will form the overall capabilities of the library (see *E-CAM High Throughput Computing Library* for the previous module). This module deals with creating a more comprehensive configuration format for the [Dask-Jobqueue](#) Python library in YAML format.

Purpose of Module

The goal is to allow numerous `cluster` instances (which is a place where tasks are executed) to be defined more broadly and cover all possibilities that the queueing system might offer as well as in configurations that are required to execute MPI/OpenMP tasks.

The implementation is generic but the specific example provided is for SLURM on the [JURECA](#) system.

Background Information

This module builds upon the work described in *E-CAM High Throughput Computing Library* and the mechanism already provided by the [Dask configuration](#) and the [Dask-Jobqueue configuration](#)

Building and Testing

The library is a Python module and can be installed with

```
python setup.py install
```

More details about how to install a Python package can be found at, for example, [Install Python packages on the research computing systems at IU](#)

To run the tests for the decorators within the library, you need the `pytest` Python package. You can run all the relevant tests from the `jobqueue_features` directory with

```
pytest tests/test_cluster.py
```

Source Code

The latest version of the library is available on the [jobqueue_features GitHub repository](#)

The code that was originally created specifically for this module can be seen in the [HTC/Yaml Merge Request](#) which can be found in the original private repository of the code.

Software Technical Information

Name `jobqueue_features`

Language Python, YAML

Licence [MIT](#)

Documentation Tool In-source documentation

Application Documentation Not currently available. Example usage provided.

Relevant Training Material Not currently available.

Software Module Developed by Adam Włodarczyk (Wrocław Centre of Networking and Supercomputing), Alan O’Cais (Juelich Supercomputing Centre)

1.7.13 HTC Multi-node Tasks

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module is the third in a sequence that will form the overall capabilities of the HTC library (see [HTC Library Configuration in YAML](#) for the previous module). This module deals with enabling tasks to be run over a set of nodes (specifically MPI/OpenMP tasks).

Purpose of Module

The initial goal is to allow the HTC library to control tasks that are executed via the MPI launcher command. The task tracked by Dask is actually the process created by the launcher. The launcher is a forked process from within the library.

The implementation is intended to be generic but the specific example implementation provided is for `srun` launcher that is used on [JURECA](#) system.

Background Information

This module builds upon the work described in [HTC Library Configuration in YAML](#).

Building and Testing

The library is a Python module and can be installed with

```
python setup.py install
```

More details about how to install a Python package can be found at, for example, [Install Python packages on the research computing systems at IU](#)

To run the tests for the decorators within the library, you need the `pytest` Python package. You can run all the relevant tests from the `jobqueue_features` directory with

```
pytest tests/test_mpi_wrapper.py
```

Specific examples of usage for the [JURECA](#) system are available in the `examples` subdirectory.

Source Code

The latest version of the library is available on the [jobqueue_features GitHub repository](#)

The code that was originally created specifically for this module can be seen in the [HTC/MPI Merge Request](#) which can be found in the original private repository of the code. Additional, more complex, examples were provided in the [HTC/MPI examples Merge Request](#)

Software Technical Information

Name `jobqueue_features`

Language Python, YAML

Licence [MIT](#)

Documentation Tool In-source documentation

Application Documentation Not currently available. Example usage provided.

Relevant Training Material Not currently available.

Software Module Developed by Adam Włodarczyk (Wrocław Centre of Networking and Supercomputing), Alan O’Cais (Juelich Supercomputing Centre)

1.7.14 Adding HTC Library to EasyBuild

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module is the fourth in a sequence that will form the overall capabilities of the HTC library (see [HTC Multi-node Tasks](#) for the previous module). This module deals with installing the software on HPC systems in a coherent manner through the tool [EasyBuild](#).

Purpose of Module

The HTC library requires configuration for the target system. Typically, this configuration is applicable system-wide. If the software is provided in the main software stack of the system, this configuration can also be provided centrally. The goal of the integration with EasyBuild is to highlight how this configuration can be made with an explicit example of the configuration for the [JURECA](#) system.

Background Information

EasyBuild is a software build and installation framework that allows you to manage (scientific) software on High Performance Computing (HPC) systems in an efficient way. Full details on can be found in the [EasyBuild documentation](#).

EasyBuild already has support for Python packages, what we describe here is the specific configuration required to install a particular version of the library on a specific software stack on JURECA.

Building and Testing

To build the software requires EasyBuild (see [installation instructions for EasyBuild here](#)) and the build command:

```
eb jobqueue_features-0.0.4-intel-para-2018b-Python-3.6.6.eb
```

However, please note that this will only work “out of the box” for those with software installation rights on the JURECA system. The provided sources (as described below) are intended as templates for those who are familiar with EasyBuild to adapt to their system (the only expected adaption would be to change the `toolchain` to suit their own system).

Source Code

The latest version of the library itself is available on the [jobqueue_features GitHub repository](#).

There is an open [Pull Request for the JURECA software stack](#) that provides all necessary dependencies for the library.

The configuration file required for JURECA is included below (a version for Python 2 can also be created by simply changing the Python dependency version):

```

easyblock = 'PythonBundle'

name = 'jobqueue_features'
version = '0.0.4'
versionsuffix = '-Python-%(pyver)s'

homepage = 'https://github.com/E-CAM/jobqueue_features'
description = """
A Python module that adds features to dask-jobqueue to handle MPI workloads and
↳different clusters.
Examples of usage can be found in the examples folder of the installation ($JOBQUEUE_
↳FEATURES_EXAMPLES)
"""

toolchain = {'name': 'intel-para', 'version': '2018b'}

dependencies = [
    ('Python', '3.6.6'),
    ('Dask', 'Nov2018Bundle', versionsuffix),
]

use_pip = True

exts_list = [
    ('typing', '3.6.6', {
        'source_urls': ['https://pypi.python.org/packages/source/t/typing/'],
        'checksums': [
↳'4027c5f6127a6267a435201981ba156de91ad0d1d98e9ddc2aa173453453492d'],
        }),
    ('pytest-cov', '2.6.0', {
        'source_urls': ['https://pypi.python.org/packages/source/p/pytest-cov/'],
        'checksums': [
↳'e360f048b7dae3f2f2a9a4d067b2dd6b6a015d384d1577c994a43f3f7cbad762'],
        }),
    (name, version, {
        'patches': ['jobqueue_features-%s.patch' % version],
        'source_tmpl': 'v%(version)s.tar.gz',
        'source_urls': ['https://github.com/E-CAM/jobqueue_features/archive/'],
        'checksums': [
↳'0152ff89f237225656348865073f73f46bda7a17c97e3bc1de8227eea450fb09', # v0.
↳0.4.tar.gz
↳'698204ef68f5842c82c5f04bfb614335254fae293f00ca65719559582c1fb181', #
↳jobqueue_features-env.patch
        ],
    }),
]

postinstallcmds = [
    'cp -r %(builddir)s/%(name)s/%(name)s-%(version)s/examples %(installdir)s/examples
↳',
    'mkdir %(installdir)s/config && cp %(builddir)s/%(name)s/%(name)s-%(version)s/
↳%(name)s/%(name)s.yaml %(installdir)s/config'
]

modextravars = {
    'DASK_ROOT_CONFIG': '%(installdir)s/config',
    'JOBQUEUE_FEATURES_EXAMPLES': '%(installdir)s/examples',
}

```

(continues on next page)

(continued from previous page)

```

}

sanity_check_paths = {
    'files': ['config/jobqueue_features.yaml'],
    'dirs': ['lib/python%(pyshortver)s/site-packages', 'examples', 'config'],
}

moduleclass = 'devel'

```

Software Technical Information

Name jobqueue_features

Language Python, YAML

Licence MIT

Documentation Tool In-source documentation

Application Documentation Not currently available. Example usage provided.

Relevant Training Material Not currently available.

Software Module Developed by Adam Włodarczyk (Wrocław Centre of Networking and Supercomputing), Alan O’Cais (Juelich Supercomputing Centre)

1.7.15 HTC MPI-Aware Tasks

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module is the fifth in a sequence that form the overall capabilities of the HTC library (see *HTC Multi-node Tasks* for the most relevant previous module where support for forked MPI workloads was added). This module deals with enabling tasks to be run over a set of nodes (specifically MPI/OpenMP tasks) where the tasks themselves are MPI aware.

Purpose of Module

In *HTC Multi-node Tasks* we added support for the HTC library to control tasks that are executed via the MPI launcher command. In that case, the task tracked by Dask is actually the process created by the launcher. For fully MPI-aware tasks, Dask itself is part of the MPI environment, running on the root process. The other processes wait for the code to be executed to come from root process. This is possible because Python is JIT compiled so we can serialise and send the instructions to the other processes (hiding complexity behind additional function calls).

The implementation is intended to be generic but the specific example implementation provided is for `srun` launcher that is used on JURECA system.

Background Information

This module builds upon the work described in *HTC Multi-node Tasks*.

There is significant complexity in this use case since the task is only sent to the root process and must be packaged and sent to other processes before they can execute anything. The other processes must then go into a waiting state for next state to be sent from root, and when the workers are supposed to shut down, they should all exit cleanly.

Building and Testing

The library is a Python module and can be installed with

```
python setup.py install
```

More details about how to install a Python package can be found at, for example, [Install Python packages on the research computing systems at IU](#)

To run the tests for the decorators within the library, you need the `pytest` Python package. You can run all the relevant tests from the `jobqueue_features` directory with

```
pytest tests/test_mpi_wrapper.py
```

Specific examples of usage for the JURECA system are available in the `examples` subdirectory.

Source Code

The latest version of the library is available on the [jobqueue_features GitHub repository](#)

The code that was originally created specifically for this module can be seen in the [MPI-capable tasks Merge Request](#). This includes a [specific example of the use case](#)

Software Technical Information

Name `jobqueue_features`

Language Python, YAML

Licence [MIT](#)

Documentation Tool In-source documentation

Application Documentation Not currently available. Example usage provided.

Relevant Training Material Not currently available.

Software Module Developed by Alan O’Cais (Juelich Supercomputing Centre) Adam Włodarczyk (Wrocław Centre of Networking and Supercomputing), Miłosz Białczak (Wrocław Centre of Networking and Supercomputing),

1.7.16 Extending available MPI runtime environments

- *Purpose of Module*

- [Background Information](#)
- [Building and Testing](#)
- [Source Code](#)

This module is another in a sequence that form the overall capabilities of the HTC library (see [HTC MPI-Aware Tasks](#) for the most relevant previous module where support for forked MPI workloads was added). This module adds support for additional MPI runtimes to make the library a more portable solution between HPC systems.

Purpose of Module

This module extends the supported MPI runtimes of `jobqueue_features`, beyond the original SLURM and `mpiexec`, to OpenMPI, Intel MPI and MPICH. This support includes the relevant arguments to provide reasonable process pinning arguments to the runtimes based on the system architecture and resources requested for each worker.

Background Information

To date, we have only included MPI launchers that do not require complex configuration (`srun` and `mpiexec`). In order to extend the supported MPI launchers we also need to be able to take into account the distribution of processes and threads by the launcher. We have this information since it is dictated by the system configuration file and the arguments the user provides when creating the Dask cluster to which they submit their tasks.

The main goal here is to make a best effort mapping between the user request and the MPI launcher options that will distribute and pin the processes/threads across the target system.

Building and Testing

The library is a Python module and can be installed with

```
python setup.py install
```

More details about how to install a Python package can be found at, for example, [Install Python packages on the research computing systems at IU](#)

To run the tests for the MPI launchers within the library, you need the `pytest` Python package. You can run all the relevant tests from the `jobqueue_features` directory with

```
pytest tests/test_mpi_wrapper.py
```

Source Code

The latest version of the library is available on the [jobqueue_features GitHub repository](#)

The code that was originally created specifically for this module can be seen in the [Merge Request](#) that added support for OpenMPI and Intel MPI, and the [Merge Request](#) that added support for MPICH.

Software Technical Information

Name Dask-traj.

Language Python (3.6, 3.7)**License** LGPL 2.1 or later**Documentation Tool** Sphinx, numpydoc format (ReST)**Application Documentation** <https://dask-traj.readthedocs.io/en/latest/>**Relevant Training Material** <https://github.com/sroet/dask-traj/tree/master/examples>**Software Module Developed by** Sander Roet

1.7.17 Dask-traj

- *Purpose of Module*
- *Current Limitations*
- *Building and Testing*
 - *Examples*
- *Source Code*

For analysis of MD simulations **MDTraj** is a fast and commonly used analysis. However MDTraj has limitations, such as the requirement that the whole trajectory and result of the computation fits into memory. This module rewrites part of MDTraj to work with **Dask** in order to achieve out-of-memory computations, and combined with **dask-distributed** results in possible out-of-machine parallelization, essential for HPCs and a (surprising) speed-up even on a single machine.

Purpose of Module

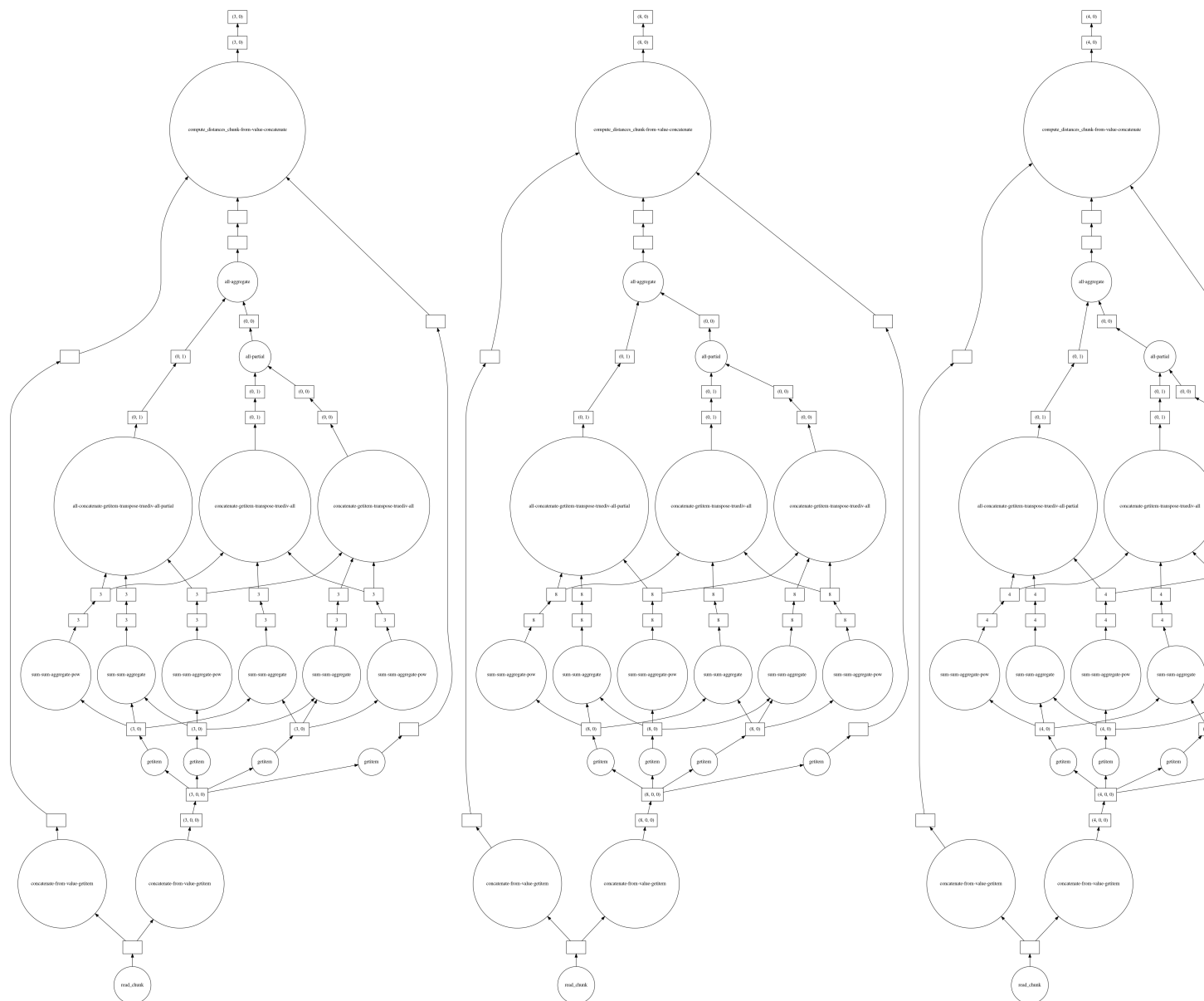
Using **MDTraj** is a fast and easy way to analyze MD trajectories. However, MDTraj has a couple limitations:

- The whole trajectory needs to fit into memory, or gathering results becomes inconvenient
- The result of the computation also needs to fit into memory
- All processes need access to all the memory, preventing out-of-machine parallelization, and HPC scaling

Dask-traj solves all 3 limitations by rewriting the MDTraj functions to work with **dask.arrays**. This is done for both the trajectory and the computation functions. As dask.arrays know how to spill to disk, this lifts the requirement to fit into memory on both.

Together with **dask-distributed** it also allows the computation to be executed in a distributed way, which allows scaling out of a single machine. In preliminary tests this approach even leads to a speedup on a single machine, which is surprising as MDTraj is already a parallel code.

The splitting of everything in Dask-traj is done in the time-axis of the MD trajectory and as a lot of analysis is embarrassingly parallel, this leads to nice non-communicating compute graphs as shown here.



Current Limitations

One very important point of `dask-traj` is that we `seek` in the trajectory file. So if your files are stored in a format that does not have an efficient seek method, the loading of Trajectories will not get a speed-up, and might even be slower than MDTraj.

Also, due to the way the code is written in MDTraj, only a subset of functions are available at the moment, but this will be expanded further in the future. If you have a use-case that requires the conversion of a MDTraj functionality, not yet present in `dask-traj`, please [make an issue](#) and I will focus on that.

Building and Testing

This code can be installed with conda using `conda install -c dask_traj`. To install the specific version associated with this module, use `conda install -c conda-forge dask_traj==0.2.2`

This code can also be installed with `pip` by running `pip install dask-traj`

Finally, this code can also be installed by downloading the source code (see the `Source Code` section below), and running `python setup.py install` from the root directory.

Tests for this module can be run with `pytest`. Install `pytest` with `pip install pytest` and then run the command `py.test` from within the directory with the source code, or `py.test --pyargs dask_traj` from anywhere after installation.

Examples

The examples require some extra dependencies to be installed, namely: `* jupyter * distributed * python-graphviz`

Which are all installable through `conda` and `pip`.

- An example on how to do analysis using Dask-traj can be found in [dask-traj_example.ipynb](#)
- An example on how to combine dask-traj with `dask.distributed` can be found in [dask-traj_distributed example.ipynb](#)

These examples can also be found in the `examples` directory in the source code. They can be run by using `jupyter notebook` from that directory (see `Jupyter notebook` documentation at <http://jupyter.org/> for more details)

Source Code

The source code for this module, and modules that build on it, is hosted at <https://github.com/sroet/dask-traj>. This module specifically includes everything up to and including [release 0.2.2](#)

1.7.18 ESDW Lyon 2019

Software Technical Information

Name `pytbc`

Language Python (3.7)

Licence GNU General Public License v3.0

Documentation Tool Sphinx/RST

Application Documentation <https://clangit.gitlab.io/pytbc/>

Relevant Training Material https://clangit.gitlab.io/pytbc/notebooks/example_TBC.html <http://campari.sourceforge.net/V3/tutorials.html>

Software Module Developed by Cassiano Langini

Contributions by Marco Bacci Andreas Vitalis Davide Garolini

pytbc

- *Purpose of Module*

- *Background Information*
- *Building and Testing*
- *Source Code*

pytbc contains Python bindings to the tree-based clustering algorithm by Vitalis and Caflisch [Vitalis2012] implemented in **Campari**. The algorithm is written in Fortran90 and the Python bindings allow for more flexibility and possibility of integration with other packages avoiding file-based I/O. The binding interface is generated with **f90wrap** to have access to derived types and then compiled with **f2py**.

Purpose of Module

The clustering algorithm published in [Vitalis2012] is a hierarchical multi-resolution clustering algorithm built on an efficient tree data structure. It is based on the Birch clustering algorithm [CIT]. **pytbc** wraps the basic functionality of the algorithm which can be used with the most common clustering distances.

Background Information

See the [project page](#) for full details.

Building and Testing

Installation

Up to date installation instructions can be found at <https://gitlab.com/clangi/pytbc#installation>

Testing and Examples

Example notebooks that leverage the package can be found at <https://gitlab.com/clangi/pytbc/-/tree/master/docs/source/notebooks>

Source Code

The *source code* of ‘*pytbc*’ is available on GitLab.com <<https://gitlab.com/clangi/pytbc>>‘_.

1.7.19 ESDW Clifden 2019

The ESDW on “Inverse Molecular Design & Inference: building a Molecular Foundry” in Clifden, Ireland in November 2019 was the starting point for the modules below.

Software Technical Information

This module is a python port of particle insertion suite of codes It also extends and generalizes the apporoch.

Language Python (3.6+)

Licence The software for this specific module is licensed under [BSD-3-Clause](#)

Documentation Tool [pdoc](#), [numpydoc](#) format (ReST)

Application Documentation [Documentation](#)

Relevant Training Material Usage instructions in this document and usage examples [here](#)

Software Module Developed by Shrinath Kumar, Zein Jaafar and Donal MacKernan

PI-auto-utility

- *Purpose of module*
- *Background Information*
- *Usage instructions*
 - *Prerequisites*
 - *Using python with lammmps*
 - *Using this module*
 - * *Setting up the simulation*
 - * *Inserting particles*
 - * *Deleting particles*
- *Examples*
- *Source Code*

This module ports the already existing modules *Particle Insertion Core* and *Particle Insertion Hydration* from the LAMMPS scripting language to Python. It allows to apply the method described in those modules to a larger variety of systems. Additionally it also generalizes the method, thereby allowing use of different forcefields.

Purpose of module

This module performs particle insertion/deletion of any type of particle in dilute or dense conditions in a variety of thermodynamic ensembles via a novel perturbative approach using LAMMPS and PyLammps, a python interface to LAMMPS. This will be extended to other MD engines such as GROMACS at a later stage.

This type of alchemical insertion and deletion is useful in a whole host of situations, where one would like to compute the free energy changes associated with adding or removing particle/molecule from a complex. Common applications would include:

- Computing the binding energy of ligands to proteins
- Computing the binding energy of protein-protein complexes
- Computing the free energy change associated with increasing or decreasing solvent (hydration/dehydration)
- Computing the free energy change associated with mixing solvents

The main advantage of this type of alchemical free energy calculation is that it does not use soft-core potential as many of the approaches to date do. As such, there are less alchemical pathways to compute as the electrostatic and VdW interactions can be switched along with all other types of interactions. This results in being able to compute the free free energy differences faster with less simulation time. The other main advantage is that due to the mathematical form of rescaling used, [the singularity of insertion](#) can be avoided.

Background Information

Please see [PIhydration background information](#). The only difference in this module is that the functional form of the scaling parameter λ can be chosen freely by the user.

Usage instructions

Prerequisites

For this module a custom fork of lammps is required. You can obtain it from [here](#). Clone the repository and follow the standard lammps installation procedure. This fork extends lammps's python interface to provide some important additional functionality. The module **will not** work with just the standard lammps installation.

Additionally you also need [pymbar](#) and [numpy](#) if you wish to compute free energies using mbar. You may install this through conda or pip.

```
# conda
conda install -c omnia pymbar
conda install numpy
# pip
pip install pymbar numpy
```

Using python with lammps

A LAMMPS simulation can use python (and this module) in one of two ways:

- Using python to wrap lammps through the its library interface or using one of the provided wrappers. This then allows for a python script to create one or more instance of LAMMPS and launch simulations.
- Calling python from a lammps input script using an embedded interpreter. For more details see [here](#).

This module can be used both ways but when using the embedded interpreter, care must be taken to ensure that your python script/module can be found on the search path for imports. The interactive version of Python will add the current directory to the search path for convenience but this is not done automatically when embedded.

Using this module

The implementation in this module includes:

- An `InsertionManager` class for encapsulating all the information regarding coordinates and topology for the system of particles to be inserted or deleted. Instances of this class can be used to store templates of molecules. Which can then be used to repeatedly insert particles. This class also provides some basic functionality to change coordinates and topology of the system to be inserted. It is by no means fully comprehensive and it is usually just easier to create a new data file if there are drastic changes to be made.
- Functions `insert` and `delete`, which operate on instances of the `InsertionManager` class. As their name implies, they perform the insertion and deletion of particles into another system.
- A utility class and function named `MbarWriter` and `compute_mbar_fe` which allow for computing free energies of insertion and deletion using MBAR. This uses choderalab's [pymbar](#) implementation.

Setting up the simulation

Before running a simulations you must ensure that your lammmps simulation has allocated enough space for all the types in your existing system plus the types in the system to be inserted or deleted. If you already have a data file this is most easily done using the `extras/<interaction_name>/types` argument of the `read_data` command when you use it for the first time.

Note: If the system you are inserting contains interactions that are not present in the original system you also need to use the `extras/<interaction_name>/per/atom` argument of the `read_data` command to leave space for for the number of interactions per atom. Consult the [read_data](#) documentation for more information.

Once you have your system setup you can begin to use this module by importing the `pyinsertion` module in your python code. This contains the top level classes and functions that perform insertion and deletion of particles.

Inserting particles

To start, you require two files for corresponding to the system of particles to be inserted.

1. A file that contains the coordinates and topology of of the system. This file should be in a format that Lammmps's `read_data` command can accept. However this file should **not** contain any force-field information, such as `pair_coefficients`, despite the fact that including such information is perfectly legal according to the `read_data` command.
2. The force-field information for the system to be inserted should be in a separate file, the format of which is described below.
 - Comments begin with the '#' character. They may appear at the start of a line or at the end of a line.
 - The file must contain one or more force-field sections corresponding to standard interaction types (like pairs,angles,dihedrals,bonds,impropers).
 - A section begins by enclosing its name in square brackets, like so `[pair]`.
 - This *must* be immediately followed on the next line by the keyword `style:` and then the style of the interaction along with any global arguments they require. Any valid Lammmps interaction style can be used expect for the style `hybrid`. Currently `hybrid` styles are not supported by this module.
 - The next line *must* contain the keyword `indices:` followed by a list of integers which correspond to the interaction coefficients that will be perturbed beginning with zero. For example, `indices: 1 2` will instruct the program to scale the second and third coefficient but leave the first coefficient unchanged. This is useful in situations like bond interaction where one would typically like to scale the strength of the bond but leave the equilibrium distance unchanged.
 - Finally, one or more lines of coefficient data in the form `type_id one or more args` corresponding to the specified style. Type ids must begin at with 1.
 - A section is ended with a single newline.
 - Sections can be in any order.

An example of a simple force-field file is shown below.

```
# Final force-field coefficients for inserted particles
[pair]
style: lj/cut 1.0 1.0
indices: 0 1
#      type      eps      sigma      rcut
#      1         1.0      1.0        5
#      2         0.5      1.2        5
```

These two files along with a pointer to an active lammps instance are required by the constructor of the `InsertionManager` class which is the main class of interest for insertions. In addition to these three mandatory arguments, there number of optional arguments you can specify to the constructor, such as list of `$\lambda` values to use for scaling the interaction coefficients. There are many other such optional arguments. See the code documentation for a full description of all the parameters.

Once you have an instance of the `InsertionManager` class you can perform the actual insertion by calling it's `insert` member function. This requires two parameters. The length of the relaxation period in timesteps and the number of samples required from each `$\lambda` value.

The output from this is by default just the potential energy data at each `$\lambda` point but can be changed to include any Lammps thermo-style variables. This can be achieved by passing a string or a list of strings to the `output_style` keyword of the `insert` function. The data is written out to a file named `mbar.dat` by default but can be changed by using the `outfile` keyword of the `insert` function. It is written in a format that can be easily used with the `pymbar` a python implementation of the multistate Bennett acceptance ratio.

Deleting particles

Deleting particles is much the same as inserting particles. In fact, the deletion procedure mathematically is just the inverse of the insertion approach. The same method detailed above for insertion can be used. The only change is to specify the `$\lambda` values in descending order. Thus, the `delete` member function is just a thinly veiled wrapper of the `insert` function with some additional error checking. It takes the exact same parameters as the `insert` function.

Examples

The examples link to a collection of ipython-notebook which go through some “toy” examples. These attempt to explain the functionality of this module in a practice way.

Source Code

Module Source Code

However, please note that the source code is currently under embargo until associated works are published, if you would like to be obtain a copy of the code, please contact Dr. Donal MacKernan at donal.mackernan@ucd.ie

Comparative Metadynamics

Software Technical information

This module facilitates extrapolating free energy surface (FES) feature information from short, non-converged simulations of mutated systems

Language Python (3+)

Licence The software for this specific module “Comparative Metadynamics” is licensed under [BSD-3-Clause](#)

Documentation Tool `pdoc`, `numpydoc` format (ReST)

Application Documentation [Documentation](#)

Relevant Training Material See usage examples [here](#)

Software Module Developed by Zein Jaafar, Shrinath Kumar and Donal MacKernan

- *Abstract*
- *Background Information*
- *Purpose of Module*
- *Applications*
- *Performance*
- *Software Prerequisites*
- *Usage*
 - *Examples*
- *Source Code*

Abstract

The module performs a long simulation of some given system and then many shorter simulations of mutations of the aforementioned system. Using the Free Energy Surface (FES) of the original system as a basepoint allows for meaningful information about the impact of a mutation on the system's FES to be extracted from only the short simulations.

Background Information

The use of Molecular Dynamics (MD) is highly relevant in nearly all STEM fields. Analysing MD simulations can be done by defining Collective Variables (CVs), functions of the positions of some or all of the atoms in a simulation. Then, periodically during the course of an MD simulations, the energy of the system is computed alongside all of the defined CV's. This allows the construction of a Free Energy Surface (FES) by expressing the free energy as a function of the CV's. In order to speed up the exploration of the CV space, a method called Metadynamics may be employed where a biasing potential is added to force the system to explore the CV space rather than allowing it to naturally explore the entire CV space as in regular MD.

Optimisation through mutation is a process whereby a system is optimised to perform some specific task by mutation, which broadly encompasses altering the system in any way. If the system's ability to perform said task can be characterised through the use of CV's then its ability to perform this task will manifest in some feature or collection of features in the FES. Thus, the process of optimisation through mutation will break down broadly into three steps, which are usually repeated many times.

1. Mutating the system
2. Simulating through Metadynamics
3. Analysing the FES

Purpose of Module

The purpose of this module is to speed up the process of optimisation through mutation by quickly classifying roughly how much a mutation will optimise the system or not. This quick classification will allow a much wider exploration of the possible mutations which might optimise a system.

This is done by using the FES of the original system as a starting point. In order to obtain this FES a well-converged simulation for the original system must be conducted. Then a feature of interest on the FES is chosen and potential walls are placed around it to limit the exploration of the CV space and further speed up simulation. The system is mutated and then a very short metadynamics run is performed on the mutated system. The key point is that when simulating the mutated system with metadynamics, the biasing potential used to generate the original system's FES is used as the initial biasing potential for the mutated system.

The reason for this is so that, if run for sufficiently long, metadynamics will gradually alter the profile of the original FES until it matches that of the mutated system's FES. Therefore even after a very short simulation which has not yet converged, it is possible to compare the original FES to the mutated system's FES and extrapolate what effect the mutation had on the FES; In particular it can be inferred whether the mutation has optimised the original system or not.

By performing many such mutations and short simulations this module also allows a rough comparison between which mutations best optimised the system by comparing which mutations caused the greatest change in the original FES in a fixed time interval. Thus, this module allows one to test many mutations and narrow down which ones will best optimise their system.

Applications

This module is particularly relevant to anywhere MD is being used to design systems through an iterative process such as chemical or biological labs. However, it can also be applied to areas where one needs to analyse many similar systems through MD.

Performance

For a simple water in salt system, when changing the charge on the salt ions a simulation time of 100ps was sufficient to analyse the changes that had occurred in the FES. By contrast a full simulation of the system required at least 4ns to converge.

Software Prerequisites

The core software requirements are:

1. **Python 3**

- Numpy

2. Plumed 2.5+

In addition, an MD engine is required needed to run the simulations. To run the example provided the following additional software is required:

3. LAMMPS (MD engine)

4. Moltemplate (To perform mutations)

5. **Additional Python**

- mpi4py
- matplotlib

Usage

All files discussed in this section can be found in the examples folder. This module mutates a system and then runs a metadynamics simulation of them using lammps. Thus the user needs to provide 3 scripts in advance.

These three files are as follows:

1. A plumed data file for performing metadynamics
2. A python file which will mutate their system
3. A python file which will simulate their system

These files should all be stored in the same location as `indicator_run.ipynb`. Example files are provided which explain how the file should be constructed. Once these three files are in place `indicator_run.ipynb` may be run. It will guide the user through any inputs required.

A brief summary of what `indicator_run.ipynb` does is provided below

1. Simulate the original system and save the metadynamics info into a file
2. Perform some mutations to the system using the user provided script.
3. Run multiple shorter simulations starting off where the initial simulation ended using the user provided script to simulate
4. Save and store the resulting outputs from each simulation in an accessible manner

Once this is complete the user may use the Analysis subfolder to analyse the output of the simulations. In this folder a single plumed data file needs to be created. An example file is provided which may also be used. Then the file `analyse.ipynb` may be run. Again, this file will guide the user through the necessary steps.

A brief summary of what `Analyse.ipynb` does is provided below

1. Reads in all the data created by `indicator_run.ipynb` (the COLVAR files mainly)
2. Creates histograms/probability densities from the restarted simulations
3. Runs a function that analyses the histograms which is user defined (e.g. the function might return the difference between the max and min value of the FES)
4. Visualises the resulting data

Examples

Examples can be found [here](#).

Source Code

Module Source Code

However, please note that the source code is currently under embargo until associated works are published, if you would like to obtain a copy of the code, please contact Dr. Donal MacKernan at donal.mackernan@ucd.ie or Ali Jaafar at ali.jaafar@ucd.ie.

1.8 European Environment for Scientific Software Installations

A number of modules related to the E-CAM support of the European Environment for Scientific Software Installations **EESSI** which is a collaboration between a number of academic and industrial partners in the HPC community.

Through the EESSI project, they want to set up a shared stack of scientific software installations to avoid not only duplicate work across HPC sites but also the execution of sub-optimal applications on HPC resources.

For end users, EESSI wants to provide a uniform user experience with respect to available scientific software, regardless of which system they use. The software stack is intended to work on laptops, personal workstations, HPC clusters and in the cloud, which means the project will need to support different CPUs, networks, GPUs, and so on. The intention is to make this work for any Linux distribution, and a wide variety of CPU architectures (Intel, AMD, ARM, POWER, RISC-V).

The pilot instance of the EESSI software stack includes GROMACS, and benchmarking is being done by E-CAM using that application, which is why we include these modules in this section.

Software Technical Information

Name European Environment for Scientific Software Installations

Language Ansible for infrastructure, Python for installation framework

Licence [GPL v2](#)

Documentation Tool Markdown (and [MkDocs](#))

Application Documentation <https://eessi.github.io/docs/>

Relevant Training Material <https://github.com/EESSI/eessi-demo>

Software Module Developed by Alan O’Cais (for contribution described here)

1.8.1 MPI support for EESSI-based containers

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The European Environment for Scientific Software Installations ([EESSI](#)) is a collaboration between a number of academic and industrial partners in the HPC community to set up a shared stack of scientific software installations to avoid the installation and execution of sub-optimal applications on HPC resources. The software stack is intended to work on laptops, personal workstations, HPC clusters and in the cloud, which means the project will need to support different CPUs, networks, GPUs, and so on.

EESSI can be used through via containers, however this requires some additional settings for MPI workloads. This module outlines the creation of an initialisation script that can facilitate this while also catering to systems which have no direct connection to the internet.

Purpose of Module

The EESSI architecture is built upon the [CernVM-FS distributed file system](#) which provides a scalable, reliable and low-maintenance software distribution service. CernVM-FS uses a cache so that a client only ever has local copies of the files it actually needs. The cache is populated over the [http](#) protocol.

If CernVM-FS is not available or configured where a user would like to use EESSI, it is still possible to use EESSI via a [Singularity](#) container. The container approach, however, requires additional configuration when considering MPI workloads.

In addition, there are many cases where worker nodes in HPC systems have no connection to the outside world, which makes it impossible for them to populate their CernVM-FS cache.

This module describes a script created to address both of these issues.

Background Information

The European Environment for Scientific Software Installations [EESSI](#) is a collaboration between a number of academic and industrial partners in the HPC community. Through the EESSI project, they want to set up a shared stack of scientific software installations to avoid not only duplicate work across HPC sites but also the execution of sub-optimal applications on HPC resources.

The software stack is intended to work on laptops, personal workstations, HPC clusters and in the cloud, which means the project will need to support different CPUs, networks, GPUs, and so on. When using singularity containers which leverage EESSI on HPC systems there are additional requirements to ensure that MPI workloads can be correctly launched and run.

Building and Testing

The script itself can be downloaded as described in the next section. It includes extensive commenting and, at the time of writing, is configured to use the 2020.12 version of the EESSI pilot software stack. You should configure settings in the script according to the system you have access to.

The script creates two layers of caching for CernVM-FS, a global one and a per-node cache. The script should be run from a location that has external internet access and access to the shared file system of the HPC resource. The script will inspect the architecture where it is run, and fully pre-populate the cache with the software stack for that architecture. The per-node cache is then dynamically populated from the global cache.

After running the script, it will tell the user to set a number of environment variables, e.g.,

```
export EESSI_CONFIG="container:cvmfs2 cvmfs-config.eessi-hpc.org /cvmfs/cvmfs-config.
↪eessi-hpc.org"
export EESSI_PILOT="container:cvmfs2 pilot.eessi-hpc.org /cvmfs/pilot.eessi-hpc.org"
export SINGULARITY_HOME="/p/project/cecam/singularity/cecam/oais1/home:/home/oais1"
export SINGULARITY_BIND="/p/project/cecam/singularity/cecam/alien_2020.12:/shared_
↪alien,/tmp:/local_alien,/p/project/cecam/singularity/cecam/oais1/home/default.
↪local:/etc/cvmfs/default.local"
export SINGULARITY_SCRATCH="/var/lib/cvmfs,/var/run/cvmfs"
```

It will also tell you how to start a shell session within the container

```
singularity shell --fusemount "$EESSI_CONFIG" --fusemount "$EESSI_PILOT" /p/project/
↪cecam/singularity/cecam/oais1/client-pilot-centos7-x86_64.sif
```

Once inside the shell you are able to initialise the EESSI computing environment, which will give you access to all the software available within EESSI, and which you can access via environment modules. You can use the modules to access the software you are interested in, and to find the path to the executables you are interested in within the container. Let's do this for GROMACS executable `gmx_mpi`.

```
Singularity> source /cvmfs/pilot.eessi-hpc.org/2020.12/init/bash
Found EESSI pilot repo @ /cvmfs/pilot.eessi-hpc.org/2020.12!
Using x86_64/intel/skylake_avx512 as software subdirectory.
```

(continues on next page)

(continued from previous page)

```

Using /cvmfs/pilot.eessi-hpc.org/2020.12/software/x86_64/intel/skylake_avx512/modules/
↳all as the directory to be added to MODULEPATH.
Found Lmod configuration file at /cvmfs/pilot.eessi-hpc.org/2020.12/software/x86_64/
↳intel/skylake_avx512/.lmod/lmodrc.lua
Initializing Lmod...
Prepending /cvmfs/pilot.eessi-hpc.org/2020.12/software/x86_64/intel/skylake_avx512/
↳modules/all to $MODULEPATH...
Environment set up to use EESSI pilot software stack, have fun!
[EESSI pilot 2020.12] $ module load GROMACS
[EESSI pilot 2020.12] $ which gmx_mpi
/cvmfs/pilot.eessi-hpc.org/2020.12/software/x86_64/intel/skylake_avx512/software/
↳GROMACS/2020.1-foss-2020a-Python-3.8.2/bin/gmx_mpi

```

Now that we know the path to the executable within the container, we can call it directly from outside the container and use it within a batch job. We show how one can execute a GROMACS benchwork using the installation found inside EESSI (on JUWELS):

```

[juwels01 ~]$ SLURM_MPI_TYPE=pspmix OMP_NUM_THREADS=2 \
    srun --time=00:05:00 --nodes=1 --ntasks-per-node=24 --cpus-per-task=2 \
    singularity exec --fusemount "$EESSI_CONFIG" --fusemount "$EESSI_PILOT" \
↳\
    /p/project/cecam/singularity/cecam/oais1/client-pilot_centos7-x86_64.
↳sif \
    /cvmfs/pilot.eessi-hpc.org/2020.12/software/x86_64/intel/skylake_avx512/
↳software/GROMACS/2020.1-foss-2020a-Python-3.8.2/bin/gmx_mpi \
    mdrun -s ion_channel.tpr -maxh 0.50 -rethway -noconfout -nsteps 10 -g_
↳logfile

```

Source Code

EESSI is still in a pilot phase, and for this reason the final version of this script cannot be created until the underlying requirements have stabilised. For the time being the script is contained in an [issue in the EESSI filesystem layer repository](#).

Software Technical Information

Name LearnHPC

Language Terraform for infrastructure

Licence [GPL v2](#)

Documentation Tool Markdown (and [MkDocs](#))

Application Documentation <https://learnhpc.eu/>

Relevant Training Material <https://learnhpc.eu/hpc-intro>

Software Module Developed by Alan O'Cais (for contribution described here)

1.8.2 EESSI and vGPU support in Magic Castle

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

In the module *MPI support for EESSI-based containers*, we introduced the European Environment for Scientific Software Installations (EESSI) which provides a shared stack of scientific software installations. That software stack is intended to work on laptops, personal workstations, HPC clusters and in the cloud. That initiative is built upon the previous efforts of [Compute Canada](#) to develop a pan-Canadian software infrastructure.

Another interesting project to come from Compute Canada, which leverages the software infrastructure, is [Magic Castle](#). Magic Castle which aims to recreate the Compute Canada user experience in public clouds, it uses the open-source software Terraform and HashiCorp Language (HCL) to define the virtual machines, volumes, and networks that are required to replicate a virtual HPC infrastructure. After deployment, the user is provided with a complete HPC cluster software environment including a Slurm scheduler, a Globus Endpoint, JupyterHub, LDAP, DNS, and over 3000 research software applications compiled by experts with EasyBuild.

Magic Castle is compatible with AWS, Microsoft Azure, Google Cloud, OpenStack, and OVH.

Purpose of Module

This module describes the inclusion and support of the EESSI software stack in Magic Castle. In addition we also include the generalisation of the virtual GPU (vGPU) support within Magic Castle for those found in the [Fenix Research Infrastructure](#).

Background Information

EU-wide requirements for HPC training are exploding as the adoption of HPC in the wider scientific community gathers pace. However, the number of topics that can be thoroughly addressed without providing access to actual HPC resources is very limited, even at the introductory level. In cases where such access is available, security concerns and the overhead of the process of provisioning accounts make the scalability of this approach questionable.

EU-wide access to HPC resources on the scale required to meet the training needs of all countries is an objective that we attempt to address with [LearnHPC](#). The proposed solution leverages Magic Castle to provision virtual HPC systems in a public cloud. This infrastructure will allow us to dynamically create temporary event-specific HPC clusters for training purposes, including a scientific software stack from EESSI.

Building and Testing

Since EESSI is now already integrated in Magic Castle, one can simply follow the standard [Magic Castle setup instructions](#) and use the [switch for the EESSI software stack](#) in the infrastructure configuration file.

If you use vGPU enabled instances for execution nodes in your virtual cluster, the vGPUs will be automatically configured and included as available resources in the SLURM environment.

Source Code

For EESSI support in Magic Castle, see

- https://github.com/ComputeCanada/magic_castle/pull/124
- https://github.com/ComputeCanada/puppet-magic_castle/pull/77

- <https://github.com/EESSI/software-layer/pull/43>
- <https://github.com/EESSI/software-layer/pull/47>

For the support of the vGPUs from the Fenix Infrastructure, see

- https://github.com/ComputeCanada/puppet-magic_castle/pull/93
- https://github.com/ComputeCanada/puppet-magic_castle/pull/95
- https://github.com/ComputeCanada/puppet-magic_castle/pull/94

Software Technical Information

Name EESSI GitHub Action

Language Yaml, bash

Licence MIT

Documentation Tool Markdown

Application Documentation <https://github.com/marketplace/actions/eessi>

Relevant Training Material None

Software Module Developed by Alan O’Cais (for contribution described here)

1.8.3 EESSI-based GitHub Action for Continuous Integration

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The European Environment for Scientific Software Installations (**EESSI**) is a collaboration between a number of academic and industrial partners in the HPC community to set up a shared stack of scientific software installations to avoid the installation and execution of sub-optimal applications on HPC resources. The software stack is intended to work on laptops, personal workstations, HPC clusters and in the cloud, which means the project will need to support different CPUs, networks, GPUs, and so on.

EESSI can be leveraged in continuous integration (CI) workflows to easily provide the dependencies of an application. With this module we are a [GitHub Action](#) for EESSI so that it can be used with CI on GitHub.

Purpose of Module

To set up the European Environment for Scientific Software Installations (EESSI) for use in GitHub Workflows.

Background Information

The European Environment for Scientific Software Installations **EESSI** is a collaboration between a number of academic and industrial partners in the HPC community. Through the EESSI project, they want to set up a shared stack of

scientific software installations to avoid not only duplicate work across HPC sites but also the execution of sub-optimal applications on HPC resources.

The software stack is intended to work on laptops, personal workstations, HPC clusters and in the cloud, which means the project will need to support different CPUs, networks, GPUs, and so on. As such the stack can also be leveraged to provide dependencies for applications within CI workflows.

Building and Testing

You can use this GitHub Action in a workflow in your own repository, see the [EESSI action in the GitHub Marketplace](#) for further details.

A minimal job example for GitHub-hosted runners of type `ubuntu-latest` is:

```
jobs:
  ubuntu-minimal:
    runs-on: ubuntu-latest
    steps:
    - uses: eessi/github-action-eessi@v1
    - name: Test EESSI
      run: |
        module avail
      shell: bash
```

This means that one can potentially load any application provided by EESSI in your workflow. A further full example that uses GROMACS from a particular version of the EESSI stack (2020.12) is:

```
name: ubuntu_gromacs
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - uses: eessi/github-action-eessi@main
      with:
        eessi_stack_version: '2020.12'
    - name: Test EESSI
      run: |
        module load GROMACS
        gmx --version
      shell: bash
```

Source Code

We link here the [GitHub repository](#) of the EESSI GitHub Action.

General Information

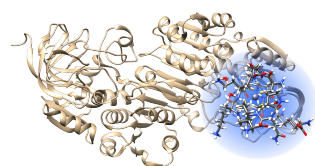
Contents

- *Electronic Structure Modules*
 - *Introduction*
 - *Extended Software Development Workshops*
 - *Pilot Projects*

- *How to contribute?*
- search

Electronic Structure Modules

2.1 Introduction



This is a collection of the modules that have been created by E-CAM community within the area of Electronic Structure. This documentation is created using ReStructured Text and the git repository for the documentation source files can be found at <https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library> which are public and open to contributions.

In the context of E-CAM, the definition of a software module is any piece of software that could be of use to the E-CAM community and that encapsulates some additional functionality, enhanced performance or improved usability for people performing computational simulations in the domain areas of interest to us.

This definition is deliberately broader than the traditional concept of a module as defined in the semantics of most high-level programming languages and is intended to capture inter alia workflow scripts, analysis tools and test suites as well as traditional subroutines and functions. Because such E-CAM modules will form a heterogeneous collection we prefer to refer to this as an E-CAM software repository rather than a library (since the word library carries a particular meaning in the programming world). The modules do however share with the traditional computer science definition the concept of hiding the internal workings of a module behind simple and well-defined interfaces. It is probable that in many cases the modules will result from the abstraction and refactoring of useful ideas from existing codes rather than being written entirely de novo.

Perhaps more important than exactly what a module is, is how it is written and used. A final E-CAM module adheres to current best-practice programming style conventions, is well documented and comes with either regression or unit tests (and any necessary associated data). E-CAM modules should be written in such a way that they can potentially take advantage of anticipated hardware developments in the near future (and this is one of the training objectives of E-CAM).

2.2 Extended Software Development Workshops

2.2.1 ESDW Zaragoza 2016

The first Electronic Structure ESDW in Zaragoza in June 2016 was the starting point for the modules below.

MatrixSwitch

Software Technical Information

The information in this section describes MatrixSwitch as a whole. Information specific to the additions in this module are in subsequent sections.

Language Fortran 2008

Documentation Tool Sphinx, ReStructuredText

Application Documentation [ESL wiki](#)

Relevant Training Material See usage examples in the `examples` directory of the source code.

Licence Simplified BSD

- *Purpose of Module*
- *Background Information*
- *Installation*
- *Testing*
- *Source Code*

Purpose of Module

MatrixSwitch is a module which acts as an intermediary interface layer between high-level routines for physics-related algorithms and low-level routines dealing with matrix storage and manipulation. This allows the high-level routines to be written in a way which is physically transparent, and enables them to switch seamlessly between different software implementations of the matrix operations.

Background Information

MatrixSwitch is a software library and module to be used within a calling code. It is developed within the same repository project as other libraries (see Source Code Section), but all are self-contained within separate directories.

Installation

The source code of the *MatrixSwitch* module is bundled in the git repository of the `omm-bundle` software which you can obtain using `git`:

```
git clone https://gitlab.e-cam2020.eu/ESL/omm.git
```

The source code of the *MatrixSwitch* module itself is contained in a subdirectory with the same name, *MatrixSwitch*.

Note: The information contained in the *Installation* and *Testing* sections are likely to work with the latest version of the source code from the repository. If this is not the case you can revert to the commit where the information is guaranteed to work:

```
git checkout 919d916f
```

1. Enter the `src` directory.
2. Copy `make.inc.example` to `make.inc` and modify it to suit your needs. Available options for `FPPFLAGS` are:
 - `-DHAVE_MPI`: enable MPI parallel routines
 - `-DHAVE_LAPACK`: enable LAPACK routines
 - `-DHAVE_SCALAPACK`: enable ScaLAPACK routines (requires MPI)
 - `-DHAVE_PSPBLAS`: enable to link to pspBLAS (requires pspBLAS installed at first)
 - `-DCONV`: enable automatic conversion of scalar types (real/complex) to agree with matrix definitions (real/complex). Note that conversions from complex to real will simply discard the imaginary part.
3. Type `make -f Makefile.manual`.
4. Type `make -f Makefile.manual install`.

Note: We provide also the possibility to build modules with Autotools. You can find the related documentation in the following files: [omm-bundle](#) and [MatrixSwitch/doc](#)

Testing

The `examples` directory contains a number of small programs that make use of *MatrixSwitch*. These can be useful both for testing the installation and for learning how to use the library. To compile them:

1. Enter the `examples` directory.
2. Copy `make.inc.example` to `make.inc` and modify it to suit your needs. Be aware that `make.inc` in the `src` directory will also be used.
3. Type `make -f Makefile.manual`.

Each example contains a header explaining what the program does and providing sample output to compare against.

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [omm-bundle](#) project. The *MatrixSwitch* directory can be found [here](#).

libOMM

Software Technical Information

Language Fortran 2008

Documentation Tool Sphinx, ReStructuredText

Application Documentation [ESL wiki](#)

Relevant Training Material See usage examples in the `examples` directory of the source code.

Licence Simplified BSD

- *Purpose of Module*
- *Background Information*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

Purpose of Module

libOMM solves the Kohn-Sham equation as a generalized eigenvalue problem for a fixed Hamiltonian. It implements the orbital minimization method (OMM), which works within a density matrix formalism. The basic strategy of the OMM is to find the set of Wannier functions (WFs) describing the occupied subspace by direct unconstrained minimization of an appropriately-constructed functional. The density matrix can then be calculated from the WFs. The solver is usually employed within an outer self-consistency (SCF) cycle. Therefore, the WFs resulting from one SCF iteration can be saved and then re-used as the initial guess for the next iteration.

Background Information

libOMM is a software library to be used within a calling code. It is built on top of the MatrixSwitch library for dealing with matrix storage and operations. Both libraries are developed within the same repository project (see [Source Code](#)), but are self-contained within separate directories.

Software Technical Information

License Simplified BSD

Language Fortran 2008

Documentation Tool Source code documentation in progress.

Application Documentation [The ESL wiki](#)

Relevant Training Material See usage examples in the `examples` directory of the source code.

Installation

The source code of the *LibOMM* module is bundled in the git repository of the `omm-bundle` software which you can obtain using `git`:

```
git clone https://gitlab.e-cam2020.eu/ESL/omm.git
```

The source code of the *LibOMM* module itself is contained in a subdirectory with the same name, `LibOMM`.

Note: The information contained in the *Installation* and *Testing* sections are likely to work with the latest version of the source code from the repository. If this is not the case you can revert to the commit where the information is guaranteed to work:

```
git checkout 7eda3275
```

1. Enter the `src` directory.
2. Copy `make.inc.example` to `make.inc` and modify it to suit your needs. `MSLIBPATH` should point to the `MatrixSwitch` directory (default in `make.inc.example` is for the version included in the distribution). `LibOMM` should be compiled with the `-DCONV` flag. Some available options for `FPPFLAGS` are:
 - `-DHAVE_MPI`: enable MPI parallel routines
 - `-DHAVE_LAPACK`: enable LAPACK routines (currently necessary for preconditioning/Cholesky factorization)
 - `-DHAVE_SCALAPACK`: enable ScaLAPACK routines (requires `-DMP`)
 - `-DNORAND`: fixed seed for the random number generator. Enable for testing purposes.
 - `-DCBIND`: use `ISO_C_BINDING` for LOGICAL inputs in the wrapper interfaces. Enable for linking to C.
3. Type `make -f Makefile.manual`.
4. Type `make -f Makefile.manual install`.

Note: We provide also the possibility to build modules with Autotools. You can find the related documentation in the following files: [omm-bundle](#) and [LibOMM/doc](#)

Testing

The `examples` directory contains a number of small programs that make use of `libOMM` with `MatrixSwitch`. These can be useful both for testing the installation and for learning how to use the library. To compile them:

1. Enter the `examples` directory.
2. Copy `make.inc.example` to `make.inc` and modify it to suit your needs. Be aware that `make.inc` in the `src` directory will also be used.
3. Type `make -f Makefile.manual`.

Each example contains a header explaining what the program does and providing sample output to compare against.

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [omm-bundle](#) project. The libOMM directory can be found [here](#).

FDF - Flexible Data Format

Software Technical Information

Language Fortran 95

Documentation Tool Sphinx, ReStructuredText

Application Documentation [ESL wiki](#)

Licence GPL

- *Purpose of Module*
- *Background Information*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

Purpose of Module

FDF (Flexible Data Format) is an input file parser that offers an easy, transferable and practical way for a Fortran program to read its input. It is text (ASCII) based, and conceived for small data (input parameters). Every input piece of data is introduced in a line of an input file (which can be standard input) by writing a name-value pair, that is, a name characterising the data, and its value. If the latter corresponds to a physical magnitude, the units can also be specified after the value. Names can be long and should be descriptive of the value it corresponds to. FDF blocks are used to input structured data, in which case, the program using FDF reads the inside of the block.

From the programming point of view, FDF allows for any data to be retrieved whenever, from any part of the code, and in any order.

If a piece of data sought by FDF is not found in the input file, FDF will return a default value, as set up in the call to the FDF routine.

Background Information

FDF is a software library and module to be used within a calling code. It is developed as part of the Siesta DFT code (see [Source Code](#)), but is self-contained within a separate directory and can be used independently of the main code.

Software Technical Information

License GPL

Language Fortran 95

Documentation Tool Source code documentation in progress.

Application Documentation [ESL wiki](#)

Relevant Training Material Creation of materials in progress.

Installation

Note: The information contained in the *Installation* and *Testing* sections are likely to work with the latest version of the source code from the Siesta website. If this is not the case you can download the [siesta-4.1-b2 release](#) where the information is guaranteed to work.

For now, FDF has to be compiled as part of Siesta; see the documentation in the `Docs` directory. Once compiled, the FDF library and module files can be found in the `fdf` subdirectory of the building directory.

1. For the sequential version installation, go to `Obj` and issue the command:

```
sh ../Src/obj_setup.sh
```

- If the intel compiler is used, do:

```
cp intel.make arch.make
```

- If the gcc compiler is used, do:

```
cp gfortran.make arch.make
```

then do:

```
make
```

2. For parallel version installation, you should follow the same procedure except of using a appropriate parallel `arch.make`. A [arch.make](#) file with gcc compiler is available in E-CAM website.

Testing

Choose one specific test under the `Obj/Tests` directory, do:

```
make
```

Compare the output files with those under `Tests/Reference`.

Source Code

The source code is available from the [Launchpad](#) under the [siesta](#) project. The FDF directory can be found [here](#).

Libpspio

Software Technical Information

Language Libpspio is written in C, with bindings in Fortran 2003.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation Provide a link to any documentation

Application Documentation [ESL wiki](#)

Licence GNU Lesser GPL

- *Purpose of Module*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

Libpspio is a pseudopotentials I/O library for Density-Functional Theory (DFT) calculations. It can both read and write pseudopotential data, which makes it suitable for use with pseudopotential generators and electronic structure codes.

Purpose of Module

The main objective of Libpspio is to let any DFT code access or produce pseudopotential information without having to care about file formats. Libpspio is a valuable alternative to most error-prone homemade implementations and is helpful in improving file format specifications.

Software Technical Information

Language Libpspio is written in C, with bindings in Fortran 2003.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation Provide a link to any documentation

Application Documentation [The ESL wiki](#)

Licence GNU Lesser GPL

Installation

A release can be download from [This link](#) Current installation and testing are done with gcc compiler. GNU Scientific Library (GSL, version>1.15) and Check (a unit test framework for C, version>0.94) is required for installation and testing.

Here are the commands for installation:

```
$ tar xfvz libpspio-0.0.0.tar.gz
$ ./configure
$ make
```

Note: We provide also the possibility to build modules with Autotools. [This](#) is a useful document.

Testing

Libpspio contains several unit tests that can be used to check the compilation and to perform regression testing. These tests can be executed by doing:

```
$ make check
```

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [pspio](#) project. The Libpspio directory can be found [here](#).

Libescdf

Software Technical Information

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [The ESL wiki](#)

Licence L-GPL v3

- *Purpose of Module*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

Libescdf is a library containing tools for reading and writing massive data structures related to electronic structure calculations, following the standards defined in the [Electronic Structure Common Data Format](#)

Purpose of Module

Libescdf is a library created to exchange electronic-structure-related data in a platform-independent and efficient manner. It is based on the Electronic Structure Common Data Format Specifications, as well as HDF5.

Software Technical Information

License L-GPL v3

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen

Application Documentation [ESL wiki](#)

Installation

A release can be download from [this link](#) Current installation and testing are done with gcc compiler. HDF5 is required for installation and testing.

Here are the commands for installation:

```
$ tar xfvz libescdf-0.1.0.tar.gz
$ ./configure
$ make
```

Note: We provide also the possibility to build modules with Autotools. [Here](#) are some useful documents.

Testing

Libescdf contains several unit tests that can be used to check the compilation and to perform regression testing. Check (version>=0.9.4) is required for installation and testing. These tests can be executed by doing:

```
$ make check
```

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [escdf](#) project. The Libescdf directory can be found [here](#).

POKE

Software Technical Information

Language Fortran 2008

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [ESL wiki](#)

Licence L-GPL v3

- *Purpose of Module*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

Poke is a solver for the Poisson equation designed for electronic structure codes

Purpose of Module

Poke is a solver for the Poisson equation designed for electronic structure codes. Similarly to the eigensolvers, the aim is to implement in a single package several different algorithms of use in different situations, providing a unified and clean interface for the user. Special attention goes to allowing different FFT back ends to be connected to the library.

Software Technical Information

License LGPL v3

Language Fortran 2008

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [The ESL wiki](#)

Installation

A release can be download from [this link](#) Current installation and testing are done with gcc compiler. FFTW is required for installation and testing.

Here are the commands for installation:

```
$ tar xfvz poke-ahi.tar.gz
$ ./configure
$ make
```

Note: We provide also the possibility to build modules with Autotools. [This](#) is a useful document.

Testing

Poke contains several unit tests that can be used to check the compilation and to perform regression testing. These tests can be executed by doing:

```
$ make check
```

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [poke](#) project. The poke directory can be found [here](#).

SQARE radial grids and functions

Software Technical Information

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [ESL wiki](#)

Licence L-GPL v3

- *Purpose of Module*
- *Background Information*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

SQARE (Solvers for quantum atomic radial equations) is a library of utilities intended for dealing with functions discretized on radial meshes, wave-equations with spherical symmetry and their corresponding quantum states. The utilities are segregated into three levels: radial grids and functions, ODE solvers, and states.

Purpose of Module

This module provides functions and structures to create radial meshes and define discretized radial functions on those meshes.

Background Information

If the modifications are to an existing code base then this would be the place to describe that codebase and how to get access to it.

Software Technical Information

License LGLP v3

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [The ESL wiki](#)

Installation

A release can be download from [this link](#) Current installation and testing are done with gcc compiler. Check (version \geq 0.9.4) is required for installation and testing.

Here are the commands for installation:

```
$ tar xfvz libsquare-0.0.0.tar.gz
$ ./configure
$ make
```

Testing

SQARE contains several unit tests that can be used to check the compilation and to perform regression testing. These tests can be executed by doing:

```
$ make check
```

Source Code

The source code is available from the [E-CAM Gitlab](#) under the `square` project. The SQARE Grids directory can be found [here](#).

SQARE ODE

Software Technical Information

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [ESL wiki](#)

Licence L-GPL v3

- *Purpose of Module*
- *Background Information*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

SQARE (Solvers for quantum atomic radial equations) is a library of utilities intended for dealing with functions discretized on radial meshes, wave-equations with spherical symmetry and their corresponding quantum states. The utilities are segregated into three levels: radial grids and functions, ODE solvers, and states.

Purpose of Module

This module provides functions and structures to solve ordinary differential equations on a radial mesh.

Background Information

If the modifications are to an existing code base then this would be the place to describe that codebase and how to get access to it.

Software Technical Information

License LGPL v3

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [The ESL wiki](#)

Installation

A release can be download from [This link](#) Current installation and testing are done with gcc compiler. Check (version \geq 0.9.4) is required for installation and testing.

Here are the commands for installation:

```
$ tar xfvz libsquare-0.0.0.tar.gz
$ ./configure
$ make
```

Testing

SQARE contains several unit tests that can be used to check the compilation and to perform regression testing. These tests can be executed by doing:

```
$ make check
```

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [square](#) project. The SQARE ODE-solvers directory can be found [here](#).

SQARE states

Software Technical Information

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [ESL wiki](#)

Licence L-GPL v3

- *Purpose of Module*
- *Background Information*
- *Software Technical Information*
- *Installation*
- *Testing*
- *Source Code*

SQARE (Solvers for quantum atomic radial equations) is a library of utilities intended for dealing with functions discretized on radial meshes, wave-equations with spherical symmetry and their corresponding quantum states. The utilities are segregated into three levels: radial grids and functions, ODE solvers, and states.

Purpose of Module

This module provides functions and structures to solve radial wave-equations in various flavors and obtain the corresponding eigenstates.

Background Information

If the modifications are to an existing code base then this would be the place to describe that codebase and how to get access to it.

Software Technical Information

License LGLP v3

Language C with Fortran 2003 bindings.

Documentation Tool Doxygen,Sphinx,ReStructuredText

Application Documentation [The ESL wiki](#)

Installation

A release can be download from [this link](#) Current installation and testing are done with gcc compiler. Check (version>=0.9.4) is required for installation and testing.

Here are the commands for installation:

```
$ tar xfvz libsquare-0.0.0.tar.gz
$ ./configure
$ make
```

Testing

SQARE contains several unit tests that can be used to check the compilation and to perform regression testing. These tests can be executed by doing:

```
$ make check
```

Source Code

The source code is available from the [E-CAM Gitlab](#) under the `sqare` project. The SQARE States directory can be found [here](#).

Software Technical Information

Name ESL Demonstrator

Language Fortran 2003

Licence [Mozilla Public License v2.0](#)

Documentation Tool [Doxygen](#)

Application Documentation *Not currently available*

Relevant Training Material *Not currently available*

The ESL Demonstrator

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The ESL Demonstrator is a basic atomic-scale simulation software illustrating how to use and bring together the various available components of the [Electronic Structure Library](#) (ESL). It is meant to be used as a concrete implementation example for both end-users and developers. For users, it evidences and explains the typical operations and building blocks of an electronic structure code. For developers, it shows how to bring together the different ESL components in a consistent way. Although it is not expected to produce production-grade results, the ESL Demonstrator can be helpful for beginners who want to discover the field of electronic-structure calculations.

Purpose of Module

Since 2014, researchers, engineers and developers from all over the world have regularly gathered to design, coordinate and develop software libraries and tools of common interest for the electronic-structure community. In 2017, the available modules reached a sufficient level of usability and completeness to be used widely within the whole community. However, documenting every single module properly so that developers of electronic-structure software can integrate them seamlessly into their own codes would have been a daunting task. The challenge was two-fold:

- How do we provide usable and comprehensive documentation and keep it accurate, while all the ESL projects are evolving asynchronously, each at its own pace?
- How do we make the process efficient enough, so that a small number of volunteers can continue focus mostly on their own projects, while the rest of the community benefits from relevant information and guidelines on how to use these projects?

The ESL Demonstrator, aka [esl-demo](#), addresses this issue by providing a concrete and evolving example of a minimalistic electronic-structure program entirely based on ESL components. It constitutes a global “executable documentation” for the ESL. It is itself documented in a standard way, using Doxygen, to provide relevant explanations about how to use each ESL component in the appropriate context. In this case, such an approach is much more suitable than traditional documentation, mainly because instead of having to document between 10 and 20 components separately, the ESL developers only have to take care of one meta-component, therefore:

- it requires less effort from less people;
- it can be put into action by anyone with a working build environment;
- it provides feedback to developers across the whole ESL about the possible side effects their changes may produce;
- ESL components are built and used together, which provides a proof that they are indeed compatible and interoperable;
- API changes are automatically detected, even if they have not been communicated or published;
- defects and incompatibilities are easily made obvious and can be discussed around a concrete occurrence of the problems and side effects they may cause.

Background Information

The [esl-demo](#) program is able to perform simple ground-state calculations using plane-wave (PW) or atom-centered (AC) basis sets, as well as norm-conserving pseudopotentials.

Its architecture is made of 3 logical blocks, spanning 3 levels of execution, as illustrated in the following table:

| Plane Waves (PW) | Atom-Centered (AC) | Basis-Independent (BI) |
|---|---------------------|---|
| Self-Consistent Field | | |
| Eigensolvers | Eigensolvers | Smearing Exchange-Correlation Poisson Solver Mixing Ion-Ion Interaction |
| HΨ | Hamiltonian Builder | |
| Plane-Wave Basis | Atom-Centered Basis | |
| I/O: FDF, ESCDF - Pseudos: Pspio, PSML - FFT Wrappers | | |

Column-wise, one block takes care of plane-wave-related data and processes, another one focuses on atom-centered aspects, and the remaining one handles everything independent from the basis sets. At the lowest level, the program interacts with the computer hardware, operating system and system libraries available, as well as imports/exports data related to the current calculation. In the middle layer, itself divided into 3 sub-levels, it implements the quantum-mechanical equations in the framework of Density-Functional Theory (DFT). At the top level, it drives the operations of the lower layers and applies completion criteria. All cells of the table but the Self-Consistent Field correspond to the use of one or more ESL components.

[esl-demo](#) is available on [Gitlab](#) and mirrored on [GitHub](#). It can be downloaded with Git. Please note that only the Gitlab version is guaranteed to be up-to-date.

Building and Testing

The `esl-demo` is based on the `esl-bundle` module, which should be installed before starting to do anything related to the `esl-demo`.

The recommended way to get started with the `esl-demo` module is first to download it from Gitlab with `Git`:

```
git clone https://gitlab.com/esl/esl-demo.git
cd esl-demo
```

Before continuing, please read the `README.rst` file of `esl-demo` carefully and make sure you have installed all the prerequisites on your computer.

The `esl-demo` module uses `Cmake` as its build system. Here is a typical sequence to follow to build the code:

```
mkdir my_build
cd my_build
cmake .. -DBUILD_TESTING=1
make -j8
```

To run `esl-demo`, you will need at least a pseudopotential and a FDF input file. Some examples are provided in the `tests/` subdirectory of the source tree (which will now also be found in your `my_build` directory). You can run the test suite in the `my_build` directory with `make test`.

Note: The information contained in the *Installation* and *Testing* sections are likely to work with the latest version of the source code from the repository. If this is not the case, you can go back to the commit where this information is guaranteed to work after the download is complete:

```
git checkout de3dac2
```

Source Code

`esl-demo` is an original ESL product created from scratch. Its source code is available from `Gitlab` under the `esl-demo` project.

2.2.2 ESDW San Sebastian 2016

The ESDW in San Sebastian in September 2016 was the starting point for the modules below.

Symmetry-Adapted-Wannier-Functions module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source Code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

This module implements the symmetry-adapted Wannier functions.

Purpose of Module

Implementation of the symmetry-adapted Wannier functions (see R. Sakuma, Phys. Rev. B 87, 235109 (2013), courtesy of R. Sakuma (Lund University, Sweden), T. Koretsune (Riken, JP), Y. Nomura (U. Tokyo, JP), Y. Nohara (Atomic-Scale Material Simulations, Co., Ltd.), R. Arita (Riken, JP))

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If 'run-custom-test-parallel', it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier90/pull/7>
<https://github.com/wannier-developers/wannier90/pull/57>
<https://github.com/wannier-developers/wannier90/pull/66>
<https://github.com/wannier-developers/wannier90/pull/81>
<https://github.com/wannier-developers/wannier90/pull/84>
<https://github.com/wannier-developers/wannier90/pull/88>
<https://github.com/wannier-developers/wannier90/pull/89>

Wannier90-TB-Interface module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

Streamlined the interface between wannier90 and tight-binding codes such as pythtb (new input variable: write_tb). Also, matrix elements of the position operator can now be printed (courtesy P. Garcia Fernandez, Unican, ES)

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of

the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [Wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If 'run-custom-test-parallel', it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier91/pull/8>
<https://github.com/wannier-developers/wannier90/pull/46>
<https://github.com/wannier-developers/wannier91/pull/56>
<https://github.com/wannier-developers/wannier91/pull/60>

Non-Collinear-Spin module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

This module implements the non-collinear spin with ultrasoft pseudos functionality.

Purpose of Module

Non-collinear spin with ultrasoft pseudos now implemented in the pw2wannier90 interface with Quantum ESPRESSO, working also in parallel (courtesy F. Thoele (ETHZ, CH), T. Koretsune (Riken, JP), L. Paulatto (UPMC Paris))

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [Wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If ‘run-custom-test-parallel’, it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier91/pull/81>

<https://github.com/wannier-developers/wannier90/pull/88>

Adaptively-Refined-Mesh module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Wannier code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

Adaptively-refined mesh is implemented correctly for even sizes (e.g., 4x4). This module contains one important bugfix (calculation of orbital magnetization), another less serious bugfix (calculation of spin-colored Fermi contours) and a number of miscellaneous smaller things noted down as TO DO since the last release. (More details in related pull-request 60, see link in source code section.)

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see ‘Wannier code history’ [<http://www.wannier.org/history.html>](http://www.wannier.org/history.html) ‘_)

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If 'run-custom-test-parallel', it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier90/pull/60>

FORD-Documentation-Tool-Integration module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source Code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

This module add the first implementation of the FORD infrastructure for code documentation (courtesy D. Gresch, ETHZ).

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [Wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If 'run-custom-test-parallel', it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier90/pull/15>

<https://github.com/wannier-developers/wannier90/pull/45>

<https://github.com/wannier-developers/wannier90/pull/67>

<https://github.com/wannier-developers/wannier90/pull/86>

Improvement-Wannier90-Z2pack-Interface module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

Improved the interface with the Z2pack code (courtesy D. Gresch, ETHZ)

This introduces the nnkpts input block, which specifies the nearest neighbours. The format is the same as in the .nnkp file, except that the number of neighbours per k-point is not needed as input. The goal of this is to have a consistent interface to ab initio codes, to be used with Z2Pack.

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [Wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If 'run-custom-test-parallel', it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier90/pull/11>

Improvements-Makefiles module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

Improvements to various Makefiles and changes to Make.inc file for Wannier90 code.

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [Wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If ‘run-custom-test-parallel’, it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier90/pull/12>

<https://github.com/wannier-developers/wannier90/pull/87>

Use_WS_Distance module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

This module add the use_ws_distance flag to improve the interpolation of band structures (courtesy of L. Paulatto, UPMC Paris).

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of

the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [Wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If 'run-custom-test-parallel', it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier91/pull/3>
<https://github.com/wannier-developers/wannier90/pull/9>
<https://github.com/wannier-developers/wannier90/pull/53>
<https://github.com/wannier-developers/wannier90/pull/65>
<https://github.com/wannier-developers/wannier90/pull/78>
<https://github.com/wannier-developers/wannier90/pull/79>

Test-Suite-Travis-CI-Integration module

Software Technical Information

Language FORTRAN

Licence GPLv2

Documentation Tool FORD for source code documentation, see [Source code documentation](#).

Application Documentation http://www.wannier.org/user_guide.html

Relevant Training Material http://www.wannier.org/user_guide.html

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

This module added a test-suite, and integrated with GitHub and Travis-CI for continuous integration. A number of tests have been added (contributed mainly by S. Ponce, Oxford). Also compilation on the buildbot test farm at the Oxford Materials Modelling Laboratory has been activated

Background Information

This module is produced during the ECAM/Wannier90-developer workshop held in San Sebastian. This coincided with the move of the Wannier90 repository to GitHub to enable easier integration of community contributions. One of the first such contributions was the ability to compute symmetry-adapted Wannier Functions. (For more background information, see [Wannier code history](#))

Installing

Installation of wannier90 code is relatively simple. Detailed installing information is given by [this link](#).

Testing

Test-Suite ([Pull-Request 5](#)) and Travis-CI integration ([Pull-Request 6](#)) are added to Wannier90 repository during this workshop.

Thus, each Pull-Request within this ECAM module passed the Travis-CI continuous integration test before being merged into the Wannier90 code. Within the Travis-CI test, a set of tests in Test-Suite are checked. Manual testing can be done through the following command:

```
$ make run-tests
```

If ‘run-custom-test-parallel’, it runs the specified test in parallel (4 process with MPI):

```
$ make run-custom-test-parallel testdir=example01
```

For more details, please see [HERE](#).

Source Code

The source code of this module can be found in the following Pull-Requests in Wannier90 repository under Github:

<https://github.com/wannier-developers/wannier90/pull/5>
<https://github.com/wannier-developers/wannier90/pull/6>
<https://github.com/wannier-developers/wannier90/pull/14>
<https://github.com/wannier-developers/wannier90/pull/17>
<https://github.com/wannier-developers/wannier90/pull/22>
<https://github.com/wannier-developers/wannier90/pull/23>
<https://github.com/wannier-developers/wannier90/pull/26>
<https://github.com/wannier-developers/wannier90/pull/49>
<https://github.com/wannier-developers/wannier90/pull/51>
<https://github.com/wannier-developers/wannier90/pull/59>
<https://github.com/wannier-developers/wannier90/pull/63>
<https://github.com/wannier-developers/wannier90/pull/79>
<https://github.com/wannier-developers/wannier90/pull/84>

2.2.3 ESDW Lausanne 2018

The ESDW in Lausanne in February 2018 was the starting point for the modules below.

ESL Bundle

Software Technical Information

Language The building framework is written in Python. For the languages used in the different modules included in the Bundle, please check the corresponding documentation.

Licence The building framework is distributed under the [GPL](#). For the licenses used in the different modules included in the Bundle, please check the corresponding documentation.

Documentation Tool ReStructuredText

Application Documentation [README](#)

Relevant Training Material Not currently available.

Software Module Developed by The ESL Bundle was created by Damien Caliste, Alin Marin Elena, Micael Oliveira, and Yann Pouillon. The building framework is based on a modified version of [JHBuild](#), which was written by James Henstridge.

- *Purpose of Module*
- *Installation*
- *Source Code*

The ESL Bundle aims at incorporating all the [CECAM Electronic Structure Library](#) modules into a single package and using a unified framework for compilation and installation.

Purpose of Module

The ESL Bundle is a collection of libraries and utilities broadly used in electronic structure calculations, put together to make their use easier by researchers and scientific software developers. It includes a building framework helping users, developers and packagers in obtaining a working installation of complex combinations of software packages without having to track the dependencies themselves.

Installation

The ESL Bundle comes with a version of **JHBuild** which has been tuned to fit the context of the ESL. **JHBuild** supports a wide variety of build systems, although it is not a build system itself. It is rather a tool designed to ease the build of collections of related source packages, that it calls “modules”. It was originally written for the **Gnome Project**, but its use has then been extended to other situations.

Most of the operations are performed by executing the `jhbuild.py` script with appropriate parameters. The command line syntax is the following:

```
jhbuild.py [global-options] command [command-arguments]
```

The following global options are available:

- f, --file config** Use an alternative configuration file instead of the default `~/config/jhbuildrc`.
- m, --moduleset moduleset** Use a module set other than the module set listed in the configuration file. This option can be a relative path if the module set is located in the **JHBuild** `moduleset` folder, or an absolute path if located elsewhere.
- no-interact** Do not prompt the user for any input. This option is useful if leaving a build unattended, in order to ensure the build is not interrupted.

In the ESL Bundle, the default module set is `esl`. This module set provides a meta-module called `esl-bundle`, which builds and installs all the packages included in the bundle. A second meta-module called `esl-bundle-mpi` is provided, that builds the packages with MPI support. Note that not all packages can be compiled with MPI support. In that case they will be built without it.

The `jhbuild.py` script does not need to be invoked from the directory where it is located.

Note: To keep the source directory clean, we highly recommended the use of a build directory.

Therefore, a typical way of installing the collection of ESL libraries is the following:

```
mkdir my_build_dir
cd my_build_dir
../jhbuild.py build
```

By default, the `build` command will compile all the modules from the `esl-bundle` meta-module and install them in the current directory. This, and a few other options, can be changed in the configuration file. Several sample configuration files are provided in the `rcfiles` directory. These files should be suitable to build the bundle in a variety of systems, but they can also be used as a starting point to write configuration files more suited to your needs.

The configuration files use Python syntax. Here is a list of some important options:

- `modules`: dictionary of modules to build.
- `prefix`: directory where the modules should be installed.
- `checkoutroot`: where to unpack the module’s sources.

Configuration options to be passed to the modules build systems can also be specified in the configuration file. Here is an example of how to do this:

```
# Set the FC variable when invoking the configure script for all modules
autogenargs="FC=gfortran"

# Run make in parallel with two threads
makeargs="-j2"

# Here the futile module requires an extra configuration option.
# Note that this will overwrite the global options set by autogenargs, so we
# have to add it here explicitly.
module_autogenargs['futile'] = "--with-ext-linalg='-lopenblas' " + autogenargs
```

Source Code

The source code is available from the [E-CAM Gitlab](#) under the `esl-bundle` project. The ESL Bundle directory can be found [here](#).

Software Technical Information

Name EasyBuild

Language Python

Licence GPL-2.0

Documentation Tool ReST

Application Documentation <https://easybuild.readthedocs.io>

Relevant Training Material See documentation

Software Module Developed by Micael Oliveira

Add ELPA easyblock to EasyBuild

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

EasyBuild is used by a number of large HPC sites and integrating targeted support for ELPA ensures that those sites use optimally built versions of ELPA.

Purpose of Module

Automate the selection of appropriate configuration flags for ELPA within EasyBuild depending on the type of CPU and available features. Include additional options as appropriate. Build single and double precision versions of ELPA and also ensure it is linked against the expected version of the linear algebra libraries.

Background Information

EasyBuild is a software build and installation framework that allows you to manage (scientific) software on High Performance Computing (HPC) systems in an efficient way. Full details on can be found in the [EasyBuild documentation](#).

EasyBuild already had limited support for ELPA, this module allows for automated hardware specific configuration and optimisations.

Building and Testing

To build the software requires EasyBuild (see [installation instructions for EasyBuild here](#)) and an example build command would be:

```
eb ELPA-2018.11.001-intel-2019a.eb
```

Source Code

There are two relevant Pull Requests in the main EasyBuild repositories:

- <https://github.com/easybuilders/easybuild-easyblocks/pull/1621>
- <https://github.com/easybuilders/easybuild-easyconfigs/pull/8360>

2.2.4 ESDW Dublin 2019

The ESDW in Dublin in January 2019 was the starting point for the modules below.

ESL Easyconfigs

Software Technical Information

Language The easyconfigs are written in Python.

Licence The building framework is distributed under the [GPL](#). For the licenses used in the different modules included in the Bundle, please check the corresponding documentation.

Documentation Tool ReStructuredText

Application Documentation [README](#)

Relevant Training Material Not currently available.

Software Module Developed by The ESL Easyconfigs was created by Micael Oliveira, Yann Pouillon and Alin Marin Elena.

- *Purpose of Module*
- *Installation*
- *Source Code*

The ESL Easyconfigs aims at providing for all the [CECAM Electronic Structure Library](#) modules and their dependencies easybuild easyconfigs to allow easy installation on supercomputers around the world that use EasyBuild package manager.

Purpose of Module

The ESL Easyconfig is a collection of Easybuild easyconfigs that allow to easily build on a supercomputer all the libraries and utilities broadly used in electronic structure calculations, put together to make their use easier by researchers and scientific software developers. It includes a set of recipes for building the libraries and their dependencies helping users, developers and packagers in obtaining a working installation of complex combinations of software packages without having to track the dependencies themselves. We are aiming at providing the recipes up to date for two of the most common toolchains foss and intel. Once considered mature enough the recipes will be upstreamed to EasyBuild official catalogue.

Installation

One needs to install firstly [Easybuild](#) by following the preferred instructions

To install the full set of ESL modules and their dependencies for foss toolchain version 2019a (latest release at time of writing) one needs to do

```
eb easyconfigs/e/esl-bundle/esl-bundle-0.3.1-foss-2019a.eb -r .
```

One shall note that in organizing the files the easyconfig recipes and their needed patches we follow the same convention as EasyBuild itself.

Source Code

The source code is available from the [Gitlab](#) under the [esl-easyconfigs](#) project. The ESL Bundle directory can be found [here](#).

Software Technical Information

Name ELSI - ELectionic Structure Infrastructure

Language ELSI is written in Fortran, with bindings in C/C++.

Licence 3-Clause [BSD License](#)

Documentation Tool Doxygen for source code documentation. LaTeX for the user manual.

Application Documentation [User Manual](#)

Relevant Training Material 'Not currently available.'

Software Module Developed by Victor Wu

ELSI - ELectionic Structure Infrastructure

- *Purpose of Module*

- [Building and Testing](#)
- [Source Code](#)

ELSI provides and enhances scalable, open-source software library solutions for electronic structure calculations in materials science, condensed matter physics, chemistry, molecular biochemistry, and many other fields. ELSI focuses on methods that solve or circumvent eigenvalue problems in electronic structure theory. The ELSI infrastructure should also be useful for other challenging eigenvalue problems.

Purpose of Module

ELSI deals with the Kohn–Sham eigenvalue problem, which is central to Kohn–Sham density-functional theory, one of the most widely used methods in electronic structure. This problem is often the bottleneck in large scale calculations by high-performance computation and many different algorithms and strategies exist to tackle it. ELSI acts as a unified software interface to access different algorithms and their corresponding implementations. This greatly simplifies the implementation and optimal use of the different strategies.

One of the key design pillars of ELSI is portability and support for various computing environments, from laptop-type computers all the way to the most efficient massively parallel supercomputers and new architectures (GPU and manycore processors).

The libraries currently supported in ELSI are:

- [ELPA](#) (massively parallel dense eigensolvers)
- [libOMM](#) (orbital minimization method)
- [PEXSI](#) (pole expansion and selected inversion)
- [EigenExa](#) (massively parallel dense eigensolvers)
- [SLEPc-SIPs](#) (sparse eigensolver based on shift-and-invert spectral transformations)
- [NTPoly](#) (density matrix purification)

ELSI is used in several electronic structure codes, such as DFTB+, DGDFT, FHI-aims, and SIESTA.

Building and Testing

ELSI can be installed as part of the ESL-Bundle. Instructions to build the ESL-Bundle are provided in the [ESL Bundle](#) module.

ELSI can also be built with [EasyBuild](#) by using the [ESL Easyconfigs](#) module.

Detailed instructions on how to build ELSI without using the above options are provided in the [User Manual](#). ELSI uses the CMake build system and the procedure to build it is fairly standard:

```
mkdir build
cd build
cmake [options] /path/to/elsi/sources
make
make install
```

A complete list of options can be found in the [User Manual](#).

ELSI also provides several test programs. To run the test programs one first needs to enable them when running CMake with the following option:

```
-DENABLE_TESTS=ON
```

The test programs can be launched with

```
make test
```

Source Code

The ELSI source code is available from the [ELSI website](#) or from the [ELSI Gitlab server](#).

ELSI was added to the *ESL Bundle* in the following Merge Request:

- https://gitlab.com/ElectronicStructureLibrary/esl-bundle/merge_requests/9

2.2.5 ESDW Lausanne 2020

The ESDW in Lausanne in February/March 2020 was the starting point for the modules below.

Software Technical Information

Name libxc_in_fhi_aims

Language Fortran, C

Licence Proprietary

Documentation Tool Manual

Application Documentation [FHI-AIMS User Manual](#)

Relevant Training Material You can download training material for the use of FHI-aims from the program of the Hands-On workshop here: [FHI-aims tutorials](#).

Software Module Developed by Libxc Developers, FHI-AIMS Developers.

Support of GGA and MGGA functionals from Libxc in FHI-AIMS

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*
 - *Implementation of a unified Libxc interface*
 - *Interface to the scalar-relativistic atomic solver*

Libxc provides hundreds of well-tested approaches to calculate interactions between electrons at the atomic scale. Implementing them individually is long and tedious. With this integration, FHI-AIMS has undergone a paradigm shift and greatly expanded its capabilities in this domain.

Purpose of Module

FHI-aims is an efficient, accurate all-electron, full-potential electronic structure code package for computational molecular and materials science (non-periodic and periodic systems) using *numeric atom-centered basis functions* (NAOs) [Blum2009]. The code supports DFT (semilocal and hybrid) and many-body perturbation theory. The numerous implemented exchange-correlation (XC) functionals in the Libxc library extend the possibilities for the users and their simulations in FHI-aims.

The purpose of the module is twofold:

1. Implementation of a unified Libxc interface in FHI-aims.
2. Enabling the use of Libxc functionals with the corresponding, properly generated minimal basis functions for GGA and meta-GGA functionals (see detailed information in the Background Information Chapter). This interfaces the scalar- relativistic atomic solver of FHI-aims (default version) with Libxc.

An interface to Libxc has been implemented before, but remained at a “proof-of-concept” stage. Libxc in FHI-aims is not only used for DFT calculations, but is needed for DFPT and the calculation magnetic and optical response properties. After full integration of Libxc, it will form an essential part of each simulation and will be used by most of the FHI-aims user.

In the long term, we hope that Libxc can extend and finally replace the internal FHI-aims XC library helping to move FHI-aims from a monolithic to a more modular software architecture. At the same time, we expect that the Libxc project will benefit from the increase of usability and visibility.

Background Information

While point one from above (Implementation of a unified Libxc interface) is straightforward to implement from the [documentation of Libxc](#) the second part needs some further explanation.

The XC potential is needed at two points of a regular DFT calculation in FHI-aims. First, during the initialization generating the minimal basis (i.e. the NAOs) and the initial density. Second, during the usual SCF iterations.

The minimal basis is calculated by solving the scalar-relativistic Schrödinger equation of the free atom for each species. In principle, due to the spherical symmetry of this problem, all contributions of the XC-potentials can be formulated in analytical terms. Thus, this requires a separate routine as during the SCF-cycle. In practice, the one-dimensional radial equations are solved on a dense logarithmic radial grid as described in [Fuchs1999]. An interface to Libxc for this atomic solver and the implementation of the corresponding expressions (the functional derivatives of the XC potential w.r.t the density) were needed. In principle, any XC functional could be used to generate the minimal basis. However, it has been empirically become evident that using the same XC functional for generating the minimal basis and during the SCF iterations guarantees a faster convergence with fewer basis functions – at least for LDA and GGA functionals. This current module only implements the Libxc interface to the various LDA and GGA functionals, but not for the meta-GGAs. Instead, still only the pw-LDA functional is used to generate the basis functions and initial density for all meta-GGA calculations. It is planned to implement a finite-difference approach for generating the meta-GGA minimal basis set in the future as the corresponding analytical expression are becoming more and more numerous (tedious to implement).

Building and Testing

The build of FHI-aims for various compiler and compiler settings is integrated in the FHI-aims Gitlab CI pipeline, where all implemented parts of this module are built, too. A test has been added to the regression test suite of FHI-aims testing the newly implemented interface and the result of a DFT simulation for diamond silicon and the PBE functional.

Source Code

Note: The source code of FHI-aims is in principle open, however, a separate license is needed to get access to it. In case you need access, please ask via: <mailto:aims-coordinators@fhi-berlin.mpg.de>. A brief overview of the needed steps are listed below.

Implementation of a unified Libxc interface

This first step was straightforward to implement. Already existing code blocks have been merged into a single module `libxc_interface.f90` and unified to have a consistent interface for all parts of the code. Some effort was needed to synchronize XC functional dependent runtime variables, which was especially tedious for the range-separated hybrid functional. All requested resources from Libxc during runtime are denoted in the main output file and citations are given for citation.

Interface to the scalar-relativistic atomic solver

Due to the rotational symmetry of the free-atom problem all terms of the XC potential v_{XC} can be express analytically. The current implementation considers all derivatives up to GGA functionals (here for spin-unpolarized case): $v = \frac{\delta E_{\text{XC}}}{\delta \rho} = \frac{\partial e}{\partial \rho} - \nabla \cdot \frac{\partial e}{\partial \nabla \rho}$

The goal is to express all terms of the energy per volume $e(\rho, \nabla \rho)$ as partial derivatives of the density or the density gradient. In case of GGA functionals, this is straightforward by using the nabla operator in spherical coordinates and using the chain rule for the appearing derivatives w.r.t to r . The final expressions have to be transformed in terms of the reduced variables $\sigma = \partial \rho^2$ as the then appearing energy derivatives can be requested from the Libxc library. The implemented routines can handle both spin-polarized and spin-unpolarized free atoms.

Software Technical Information

Name `psolver_integration`

Language Fortran 95, with YAML I/O.

Licence [GPL](#)

Documentation Tool Doxygen

Application Documentation [The Solver Package page on the BigDFT Wiki](#)

Relevant Training Material [The Solver Package page on the BigDFT Wiki](#)

Software Module Developed by BigDFT Developers, SIESTA Developers, Octopus Developers.

Integration of PSolver in SIESTA and Octopus

- *Purpose of Module*
- *Background Information*
- *Building and Testing*

- *Source Code*

A Poisson solver is an efficient tool to determine electromagnetic fields produced by an electric charge distributed in space. The integration of PSolver into SIESTA and Octopus has opened the way for these software programs to access more complex physical systems. The PSolver library allows solving the Poisson equation in much more general ways than using Fourier Transforms.

Purpose of Module

The PSolver library solves the Poisson equation using wavelets. With this approximation one can more easily take into account certain boundary conditions such as molecules (no boundaries), wires (periodic along 1 direction) and slabs (periodic along 2 directions). This is in contrast to Fourier transforms which assumes periodic boundary conditions along all lattice vectors. Additionally it allows cavities for different dielectric constants.

This implementation integrates the PSolver library into the DFT codes SIESTA and Octopus such that they may be used for end-users who require the functionalities.

Background Information

Users of the SIESTA code have always been using the Fourier transforms for solving the Poisson equation. However, a great deal of users are dealing with, in particular, slab systems given the advent of graphene, 2D materials and surface calculations. This integration allows users to control the boundaries in a very strict way without any approximations. The latest PSolver library (shipped with BigDFT 1.9.0) will work.

Additional tests have been added to SIESTA to ensure that everything works.

The OCTOPUS code has various options to solve the Poisson equations. Amongst others were the ISF library, which is a predecessor of the PSolver library. In the later OCTOPUS versions, an older version of PSolver was packaged with the OCTOPUS sources. For the recently released OCTOPUS 10, the interface to PSolver has been updated, so that both the old and the new API of PSolver can be used. This also prepares OCTOPUS to use the GPU version of PSolver, once it becomes available. The configure scripts of OCTOPUS have been adapted to correctly detect and configure an installed PSolver library, and tests using the library have been added to the OCTOPUS buildbot.

Building and Testing

To compile SIESTA with PSolver users should add this to their `arch.make`

```
LIBS += -L<build-dir>/install/lib -lPSolver-1 -latlab-1 -lfutile-1 -ldicts -lfmalloc-
↳1 -lyaml
INCFLAGS += -I<build-dir>/install/include
FPPFLAGS += -DSIESTA__PSOLVER
```

After building there are two tests, `h2o_psolver` and `si2x1h-psolver` which can be compared with `h2o` and `si2x1h`, respectively. They should be comparable.

In order to compile OCTOPUS with the PSolver library, add the options `--with-psolver-prefix` and `--with-futile-prefix` to the `configure` command of OCTOPUS:

```
./configure --with-psolver-prefix=<PSolver-top-dir> --with-futile-prefix=<Futile-top-
↳dir> ...
```

The OCTOPUS test components/`16-hartree_3d_psolver.test` is testing the correct functionality of the PSolver library in OCTOPUS.

Source Code

- [PSolver in SIESTA](#)
- [PSolver in OCTOPUS](#)

Software Technical Information

Name `elsi_rci_in_dftb+`

Language Fortran, C++.

Licence BSD 3-clause license <<https://opensource.org/licenses/BSD-3-Clause>>‘_

Documentation Tool Doxygen

Application Documentation [ELSI-RCI USer Manual](#), [DFTB+ Documentation](#)

Relevant Training Material Not currently available.

Software Module Developed by ELSI-RCI Developers, DFTB+ Developers.

Integration of ELSI-RCI in DFTB+

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

DFTB+ is a software package for carrying out fast quantum mechanical atomistic calculations based on the Density Functional Tight Binding method. ELSI-RCI provides and enhances open-source software packages which solve mathematical equations related to the simulation of materials and molecules at the atomic scale.

Purpose of Module

Integrating the ELSI library into DFTB+ makes the the additional ELPA, OMM, PEXSI and NTPoly solvers available. These solvers are particularly useful for large scale systems.

Background Information

This module is developed in connection with the Extended Software Development Workshop “[Integration of ESL modules into electronic-structure codes](#)” held in Lausanne in February 2020.

An associated paper which includes a description of the ELSI integration in DFTB+ has also been published [[DFTB](#)].

Building and Testing

ELSI support is available in the latest releases of DFTB+. Full installation and testing documentation is available in the [Install.rst](#) file of the DFTB+ release.

Specifically to enable the ELSI support, one would require the CMake option `-DWITH_ELSI` (and also `-DWITH_PEXSI` if the PEXSI solver is also to be supported).

Source Code

- [Link to a merge request containing relevant source code changes](#)

2.2.6 Other Modules

Modules not coming from ESDWs

flook

Software Technical Information

Language Fortran 1990/2003

Documentation Tool Doxygen

Application Documentation [ESL wiki API](#)

Relevant Training Material See usage examples in the `src/test` directory of the source code.

Licence MPL-2.0

- *Purpose of Module*
 - *Application*
- *Background Information*
- *Installation*
- *Testing*
- *Source Code*

Purpose of Module

The flook library is a simplified API for communicating between fortran code and the [Lua](#) scripting language. A basic method is to use flook as an interactive interpreter to pass variables back and forth between a parent fortran program. It does not only serve as a simple input engine but also allows calling specific fortran functions from within Lua. Thus by exposing fortran module procedures one can control the flow of programs as well as parameters in programs.

Application

The library is currently enabled in [Siesta](#) and [ESL-demo](#) where it can be used to change convergence parameters *on the fly* and/or being used as an MD back-end. Since it allows exchange of data between Lua and fortran basically every variable in fortran can be exposed to the user via a Lua script.

Background Information

The `flook` library is built on two libraries; 1) Lua and 2) [AOTUS](#). Both are shipped together with the software and are required when building. Lua is required to be able to run the Lua interpreter in-memory while the AOTUS library is required for the low-level communication layer.

Installation

The source code of the `flook` module is hosted on Github and can be obtained using `git` or via the release page of `flook`:

```
git clone https://github.com/ElectronicStructureLibrary/flook
cd flook
git submodule update --init --recursive
```

The source code of the `flook` module itself is contained in the `src` subdirectory. Lua depends on the readline library (with headers) to be installed. Please install this library first.

Note: The information contained in the *Downloading and installation* section are likely to work with the latest version of the source code from the repository.

1. Create an `obj` directory.
2. Create a Makefile in the `obj` directory containing:


```
TOP_DIR=.. include ../Makefile
```
3. Type `make` to compile `flook`, alternatively type `make liball` to create a unified library (with Lua, AOTUS and `flook` linked together).

Testing

The `src/test` directory contains a number of small programs that make use of `flook`. These may be useful to understand the flow of programming. You can build and test `flook` with the included shell script `quick_test.sh`.

Source Code

The source code is available from the `flook` repo on Github <<https://github.com/ElectronicStructureLibrary/flook>>‘_.

Software Technical Information

Name LibGridXC

Language Fortran

Licence [BSD 3-Clause](#)

Documentation Tool SIESTA Documentation Specifications (document available soon).

Application Documentation Source code can be browsed on [GitLab](#).

Relevant Training Material Not currently available.

Software Module Developed by J.M. Soler, A. Garcia, M. Oliveira, Y. Pouillon, C. Balbás and N. R. Papior

LibGridXC - Exchange-correlation energies and potentials in radial and 3D grids

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module allows the calculation of electronic exchange and correlation energies and potentials on simulation grids, both on serial and parallel computers.

Purpose of Module

LibGridXC provides routines to calculate the exchange and correlation energy and potential in spherical (i.e. an atom) or periodic systems, using a variety of LDA and GGA functionals, as well as a variety of van der Waals DFT functionals [DION2004] [KLIMES2009] [LEE2010] [VYDROV2010], implemented as described by Román-Pérez and Soler [ROMAN2009].

Background Information

LibGridXC was originally developed within SIESTA, under the name SiestaXC, and then extracted as a stand-alone module for the Electronic Structure Library, to be shared with other codes than SIESTA. The development efforts carried out to make it a module include the design and implementation of an Autotools-based build system compatible with the one of SIESTA, as well as the migration to Git for version control and the setting up of a Continuous Integration (CI) process.

Building and Testing

LibGridXC provides an Autotools-based build system. Its build procedure is relatively straightforward:

```
cd libgridxc-x.y.z mkdir my_build_dir cd my_build_dir ../configure --prefix=/my/install/dir make make
check make install
```

where `x.y.z` is the version of LibGridXC you want to install, `my_build_dir` is the build directory where you will compile the library, and `/my/install/dir` is the absolute path where you want to install it.

Build parameters can be adjusted by providing options to the configure script. To get a list of available options, you can use the `--help` option of the configure script, e.g. run:

```
./configure --help
```

from the top source directory of LibGridXC, or:

```
../configure --help
```

from your build directory. By using the `--enable-multiconfig` option of configure, you will be able to install both a serial and a MPI-aware version of LibGridXC with the same install prefix.

For more information about the Autotools, please consult [the Autotools Mythbuster](#).

Source Code

The source code of LibGridXC is hosted on [GitLab](#).

Software Technical Information

Name libvdwxc

Language C, with Fortran and Python interfaces.

Licence [GPL](#)

Documentation Tool Inline text comments for now. Doxygen will be used in the future.

Application Documentation [Home page of libvdwxc](#)

Relevant Training Material Not currently available.

Software Module Developed by Mikael Kuisma, Ask Hjorth Larsen

libvdwxc - A library for vdW-DF exchange-correlation functionals

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module allows the description of long-range electronic interactions between atoms and molecules.

Purpose of Module

libvdwxc is a general library for evaluating energy and potential for exchange-correlation (XC) functionals from the vdW-DF family that can be used with various density functional theory (DFT) codes. This work was inspired by the success of libXC, a library for local and semilocal XC functionals. At the moment, libvdwxc provides access to the vdW-DF1, vdW-DF2, and vdW-CX functionals. It provides interfaces for GPAW and Octopus. The library has been tested with respect to the S22 test set, various bulk properties of metals and semiconductors, and surface energies.

In a previous effort Marques et al. created a C library called LibXC. This library consists of 400 different local and semi-local functionals, and it is linked by 20 different codes. In other words, when using this library, scientists have thousands different functional and code combinations to choose from. The publication process of a functional becomes easier, because it needs just to be added into one place. In a similar spirit, libvdwxc is intended as a library that enables calculations of a particular family of functionals, which cannot be readily added to LibXC.

Background Information

All relevant information about libvdwxc, as well as downloadable packages, can be found at:

- [The home page of libvdwxc.](#)
- [The Git repository of libvdwxc.](#)

Building and Testing

libvdx provides an Autotools-based build system. Its build procedure is relatively straightforward:

```
cd libvdx-x.y.z mkdir my_build_dir cd my_build_dir ../configure --prefix=/my/install/dir make make
check make install
```

where `x.y.z` is the version of libvdx you want to install, `my_build_dir` is the build directory where you will compile the library, and `/my/install/dir` is the absolute path where you want to install it.

Build parameters can be adjusted by providing options to the configure script. To get a list of available options, you can use the `--help` option of the configure script, e.g. run:

```
../configure --help
```

from the top source directory of libvdx, or:

```
../configure --help
```

from your build directory.

For more information about the Autotools, please consult [the Autotools Mythbuster](#).

Source Code

The source code of libvdx is hosted on [Gitlab](#).

DBCSR@MatrixSwitch

Software Technical Information

The information in this section describes *DBCSR@MatrixSwitch* as a whole. Information specific to the additions in this module are in subsequent sections.

Language Fortran 2008

Documentation Tool Sphinx, ReStructuredText

Application Documentation [ESL wiki](#)

Relevant Training Material See a usage example in the `omm/MatrixSwitch/examples` directory of the source code.

Software Module Developed by Alfio Lazzaro and David López-Durán

Licence Simplified BSD

- *Purpose of Module*
- *Background Information*
- *Installation*
- *Testing*
- *Source Code*

Purpose of Module

MatrixSwitch is a module which acts as an intermediary interface layer between high-level and low-level routines dealing with matrix storage and manipulation. It allows a seamlessly switch between different software implementations of the matrix operations.

DBCSR is an optimized library to deal with sparse matrices, which appear frequently in many kind of numerical simulations. In *DBCSR@MatrixSwitch* *DBCSR* capabilities have been added to *MatrixSwitch* as an *optional* library dependency.

Background Information

MatrixSwitch, *DBCSR*, and *DBCSR@MatrixSwitch* are software libraries to be used within a calling code. *MatrixSwitch* has been developed within the same repository of other self-contained libraries, all them collected in the *omm-bundle* project (see the [Source Code](#) section below). As *DBCSR* has been added to *MatrixSwitch* in a modular way, all them can be used together or separated.

To carry out calculations in serial mode may be too slow sometimes and a parallelisation strategy is needed. In serial/parallel *MatrixSwitch* employs Lapack/ScaLapack to perform matrix operations, irrespective of their dense or sparse character. The disadvantage of the Lapack/ScaLapack schemes is that they are not optimized for sparse matrices. *DBCSR* provides the necessary algorithms to solve this problem and in addition is specially suited to work in parallel.

Installation

The source code of the *MatrixSwitch* module is contained in a subdirectory of of the *omm-bundle* package with the same name, *omm/MatrixSwitch*. ‘*omm-bundle*’ is in a git repository and can be obtained in this way:

```
git clone https://gitlab.e-cam2020.eu/esl/omm.git
```

The *DBCSR* library was originally developed as part of the *CP2K* code, it is now available as a standalone library, and can be found in the *CP2K* releases directory:

```
https://github.com/cp2k/dbcsr/releases/download/v1.0.0/dbcsr-1.0.0.tar.gz
```

Build instructions for the *DBCSR* library are available on the project page.

To enable *DBCSR@MatrixSwitch* in the *omm-bundle* package follow the steps below:

1. Enter the *omm* directory.
2. Copy `make.inc.example` to `make.inc` and modify it to suit your needs. To use *DBCSR* in *MatrixSwitch* include in your `make.inc` the path to the *DBCSR* library and add to `FPPFLAGS` the new flag `-DHAVE_DBCSR` (this requires that `-DHAVE_MPI` is also enabled).
3. Type `make -f Makefile.manual`.
4. Type `make -f Makefile.manual install`.

Testing

The `examples` directory of *MatrixSwitch* contains `example_pdcslr_pddbc.F90`. It explains the use of *DBCSR@MatrixSwitch* and how *DBCSR* works. *DBCSR* results are compared to those obtained with Scapalack to check the validity of the new procedure. If this comparison fails, the program will exit immediately. To compile it:

1. Enter the `omm/MatrixSwitch/examples` directory.

2. Copy `make.inc.example` to `make.inc` and modify it to suit your needs. Be aware that `make.inc` in the `src` directory will also be used.
3. Type `make -f Makefile.manual`.

As in the other examples in *MatrixSwitch*, `example_pdcscr_pddbc.F90` contains a header explaining what the program does and provides a sample output to compare with.

Source Code

In the [E-CAM Gitlab](#) can be found all the source codes of [MatrixSwitch](#) and [omm-bundle](#), while [DBCSR](#) itself is in the set of [CP2K Github](#) repositories.

Software Technical Information

Name MaZe for OF-DFT.

Language C.

Licence [MIT](#).

Documentation Tool [Doxygen](#).

Application Documentation [GitLab](#).

Relevant Training Material Not currently available.

Software Module Developed by Alessandro Coretti, Rodolphe Vuilleumier, Sara Bonella

Mass-Zero Constrained Dynamics for Orbital Free Density Functional Theory.

- *Purpose of Module*
- *Background Information*
- *Installation*
- *Testing*
- *Source Code*
- *References*

Purpose of Module

The program performs Orbital-Free Density Functional Theory Molecular Dynamics (OF-DFT-MD) using the Mass-Zero (MaZe) constrained molecular dynamics approach as discussed in [\[BONELLA2020\]](#). The method is based on an extended Lagrangian and the dynamics enforces, at each timestep, the Born-Oppenheimer condition that the system relaxes instantaneously to the ground state through the formalism of holonomic constraints of zero mass. The adiabatic separation between the degrees of freedom is enforced rigorously, while the numerical algorithm is exactly symplectic and time-reversible in both physical and additional set of degrees of freedom. Mathematical details about the implementation of the methods are discussed at length in Alessandro Coretti's Ph.D. thesis. The computation of the electronic density is carried on in reciprocal space through a plane-waves expansion so that the mass-zero degrees of freedom are represented by the Fourier coefficients of the electronic density field. The evolution of the

ions is performed using Velocity-Verlet algorithm, while the SHAKE algorithm is used for evolution of the additional degrees of freedom.

The code is intended for condensed matter physicists and for material scientists and it can be used for various purposes related to the subject. Even though some analysis tool is included in the package, the main goal of the software is to produce particles trajectories to be analyzed in post-production by means of external software.

In computing trajectories, MaZe is intended to stand in the middle between force-field based MD and Kohn-Sham MD in terms of efficiency and accuracy. Indeed, while the forces are computed on-the-fly at each timestep, the optimization is done on the electronic density field instead of the Kohn-Sham orbitals. This feature avoids the need for satisfying the orthonormality constraint among orbitals and allows the computational complexity of the code to scale linearly with the dimensionality of the system. On the other hand, no information on the orbitals is available. The accuracy of the simulation relies on the choice of the kinetic energy functional, which has to be provided in terms of the electronic density alone.

Background Information

The module is standalone and only relies on the libraries discussed in the next section.

Installation

The execution of the code depends on the following libraries:

- **FFTW**: a library for computing the discrete Fourier transform;
- **Libxc**: a library of exchange-correlation functionals for density-functional theory;
- **BLAS**: a library that provides standard building blocks for performing basic vector and matrix operations;
- **Argp**: an interface for parsing unix-style argument vectors;

On macOS, **Homebrew** is strongly recommended to install compiler and dependencies.

The installation is based on a Makefile. A few machine dependent variable must be defined in the file ‘./config.mk’ prior to invoking the make utility. Examples can be found in the ‘./configuration_files/’ folder. The structure of the ‘./config.mk’ file is as follows:

```
#Compiler Configuration
#Compiler command
CC =
#Compiler options
CFLAGS =
#Includefiles linkers
IFLAGS =
#Libraries linkers
LIBS =

#Test configuration
#Python command
TEST =
```

The command `make` will then build the executables. The command `make clean` cleans the files resulting from the compilation. Detailed documentation can be build using **Doxygen** through the command `make documentation`. The whole suite of regression tests can be run through the command `make tests`.

Testing

Tests for the code and for regressions are launched through a python script which can be found in `./tests/`. Move into this folder and run `python regression_tests.py -s MaZe`. The scripts can take other options in order to launch different suites of tests. Default is `'all'` which can take up to 20 minutes. Run `python regression_tests.py --help` for more information on regression tests script.

By default the script tests an MD simulation of solid Sodium using different parameters:

- **Pseudopotential:** ‘Gaussian (Gauss)’ pseudopotential and ‘Topp and Hopfield (Topp)’ pseudopotential;
- **Jacob’s ladder rung:** ‘LDA’ for Local Density Approximation and ‘GGA’ for Generalized Gradient Approximation. The approximation refers only to the kinetic functional which is ‘Thomas-Fermi (TF)’ for LDA and ‘Thomas-Fermi plus von Weiszaecker correction (TFvW)’ and ‘Perrot’ functional for GGA;
- **Kinetic functional:** As above ‘Thomas-Fermi (TF)’, ‘Thomas-Fermi plus von Weiszaecker correction (TFvW)’ and ‘Perrot’ functionals;
- **Explicit enforcing of additional constraint:** When the suffix `‘_additional_constraint’` appears in the name of the text, the conservation of the number of electrons is explicitly enforced as discussed in [BONELLA2020].

All the simulation in the tests are run using a Slater exchange functional and no correlation functional.

The subfolders inside `./tests/` can also be conveniently used as examples and references for the format of the input file `‘runtime.inpt’` and of the configuration file `‘configuration.inpt’`.

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [MaZe](#) project.

The repository contains the following directories:

- **`./source/`:** contains the source code. The subfolder `‘./source/headers/’` contains the modules’ headers, while the subfolder `‘./source/obj/’` is used for compilation file outputs;
- **`./tests/`:** contains regression tests;
- **`./scripts/`:** contains useful python scripts to run simulations over different sets of parameters;
- **`./documentation/`:** contains the documentation generated with Doxygen together with the wiki of the project;
- **`./configuration_files/`:** contains examples of configuration files to generate the executable on different machines;

References

Software Technical Information

Name MaZe for OF-DFT (HPC version).

Language C.

Licence [MIT](#).

Documentation Tool [Doxygen](#).

Application Documentation [GitLab](#).

Relevant Training Material Not currently available.

Software Module Developed by Alessandro Coretti, Marco Ferrarotti, Sergio Decherchi, Rodolphe Vuilleumier, Sara Bonella

Mass-Zero Constrained Dynamics for Orbital Free Density Functional Theory (HPC Version).

- *Purpose of Module*
- *Performance Evaluation*
- *Background Information*
- *Installation*
- *Testing*
- *Source Code*
- *References*

Purpose of Module

The program performs Orbital-Free Density Functional Theory Molecular Dynamics (OF-DFT-MD) using the Mass-Zero (MaZe) constrained molecular dynamics approach as discussed in [BONELLA2020b]. The method is based on an extended Lagrangian and the dynamics enforces, at each timestep, the Born-Oppenheimer condition that the system relaxes instantaneously to the ground state through the formalism of holonomic constraints of zero mass. The adiabatic separation between the degrees of freedom is enforced rigorously, while the numerical algorithm is exactly symplectic and time-reversible in both physical and additional set of degrees of freedom. Mathematical details about the implementation of the methods are discussed at length in Alessandro Coretti's Ph.D. thesis. The computation of the electronic density is carried on in reciprocal space through a plane-waves expansion so that the mass-zero degrees of freedom are represented by the Fourier coefficients of the electronic density field. The evolution of the ions is performed using Velocity-Verlet algorithm, while the SHAKE algorithm is used for evolution of the additional degrees of freedom.

The code is intended for condensed matter physicists and for material scientists and it can be used for various purposes related to the subject. Even though some analysis tool is included in the package, the main goal of the software is to produce particles trajectories to be analyzed in post-production by means of external software.

In computing trajectories, MaZe is intended to stand in the middle between force-field based MD and Kohn-Sham MD in terms of efficiency and accuracy. Indeed, while the forces are computed on-the-fly at each timestep, the optimization is done on the electronic density field instead of the Kohn-Sham orbitals. This feature avoids the need for satisfying the orthonormality constraint among orbitals and allows the computational complexity of the code to scale linearly with the dimensionality of the system. On the other hand, no information on the orbitals is available. The accuracy of the simulation relies on the choice of the kinetic energy functional, which has to be provided in terms of the electronic density alone.

Performance Evaluation

This version of the module has been optimized by the IIT in Genova through the following steps:

- Improved FFTW usage:
 - single plan creation (reuse of same FFTW plan for all FFT/iFFT);

- use FFTW patient planning;
- memory aligned allocation of FFT/iFFT vectors to exploit FFTW simd implementation;
- Async FFT/iFFT execution via pthread threadpool (**C-Thread-Pool**);
- ComputeForcesFromStructureFactor / ComputeStructureFactor loops parallelization through OpenMP;

The proposed optimizations on the FFT/iFFT routines allow a reduction of the execution time on all the GGA test cases by roughly 50%.

The parallelized `for` loops, tested on a compute node with 24 cores, allow to reduce by roughly 60% the execution time on the LDA test cases (with the exception of the ones using b-splines that needs further work).

Background Information

The module is standalone and only relies on the libraries discussed in the next section.

Installation

The execution of the code depends on the following libraries:

- **FFTW**: a library for computing the discrete Fourier transform;
- **Libxc**: a library of exchange-correlation functionals for density-functional theory;
- **BLAS**: a library that provides standard building blocks for performing basic vector and matrix operations;
- **Argp**: an interface for parsing unix-style argument vectors;

On macOS, **Homebrew** is strongly recommended to install compiler and dependencies.

The installation is based on a Makefile. A few machine dependent variable must be defined in the file `./config.mk` prior to invoking the `make` utility. Examples can be found in the `./configuration_files/` folder. The structure of the `./config.mk` file is as follows:

```
#Compiler Configuration
#Compiler command
CC      =
#Compiler options
CFLAGS  =
#Includefiles linkers
IFLAGS  =
#Libraries linkers
LIBS     =

#Test configuration
#Python command
TEST     =
```

The command `make` will then build the executables. The command `make clean` cleans the files resulting from the compilation. Detailed documentation can be build using **Doxygen** through the command `make documentation`. The whole suite of regression tests can be run through the command `make tests`.

Testing

Tests for the code and for regressions are launched through a python script which can be found in `./tests/`. Move into this folder and run `python regression_tests.py -s MaZe`. The scripts can take other op-

tions in order to launch different suites of tests. Default is ‘all’ which can take up to 20 minutes. Run `python regression_tests.py --help` for more information on regression tests script.

By default the script tests an MD simulation of solid Sodium using different parameters:

- **Pseudopotential:** ‘Gaussian (Gauss)’ pseudopotential and ‘Topp and Hopfield (Topp)’ pseudopotential;
- **Jacob’s ladder rung:** ‘LDA’ for Local Density Approximation and ‘GGA’ for Generalized Gradient Approximation. The approximation refers only to the kinetic functional which is ‘Thomas-Fermi (TF)’ for LDA and ‘Thomas-Fermi plus von Weiszaecker correction (TFvW)’ and ‘Perrot’ functional for GGA;
- **Kinetic functional:** As above ‘Thomas-Fermi (TF)’, ‘Thomas-Fermi plus von Weiszaecker correction (TFvW)’ and ‘Perrot’ functionals;
- **Explicit enforcing of additional constraint:** When the suffix ‘_additional_constraint’ appears in the name of the text, the conservation of the number of electrons is explicitly enforced as discussed in [BONELLA2020b].

All the simulation in the tests are run using a Slater exchange functional and no correlation functional.

The subfolders inside ‘./tests’ can also be conveniently used as examples and references for the format of the input file ‘runtime.inpt’ and of the configuration file ‘configuration.inpt’.

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [MaZe](#) project. The HPC version of the code can be found on the branch **HPC**.

The repository contains the following directories:

- **./source/:** contains the source code. The subfolder ‘./source/headers/’ contains the modules’ headers, while the subfolder ‘./source/obj/’ is used for compilation file outputs;
- **./tests/:** contains regression tests;
- **./scripts/:** contains useful python scripts to run simulations over different sets of parameters;
- **./documentation/:** contains the documentation generated with Doxygen together with the wiki of the project;
- **./configuration_files/:** contains examples of configuration files to generate the executable on different machines;

References

Software Technical Information

Name NLCG for OF-DFT.

Language C.

Licence [MIT](#).

Documentation Tool [Doxygen](#).

Application Documentation [GitLab](#).

Relevant Training Material Not currently available.

Software Module Developed by Alessandro Coretti, Rodolphe Vuilleumier, Sara Bonella

Nonlinear Conjugate Gradient for Orbital Free Density Functional Theory.

- *Purpose of Module*
- *Background Information*
- *Installation*
- *Testing*
- *Source Code*
- *References*

Purpose of Module

The program performs Orbital-Free Density Functional Theory Molecular Dynamics (OF-DFT-MD) using the Born-Oppenheimer approach. The condition that the system relaxes instantaneously to the ground state is enforced, at each time step, finding the minimum of the energy for a given nuclear configuration using a nonlinear conjugate gradient method. The results of these simulations are used as benchmarks in [BONELLA2020a] and in the simulations presented in Alessandro Coretti's Ph.D. thesis. The computation of the electronic density is carried on in reciprocal space through a plane-waves expansion so that the electronic degrees of freedom are represented by the Fourier coefficients of the electronic density field. The evolution of the ions is performed using Velocity-Verlet algorithm.

The code is intended for condensed matter physicists and for material scientists and it can be used for various purposes related to the subject. Even though some analysis tool is included in the package, the main goal of the software is to produce particles trajectories to be analyzed in post-production by means of external software.

In computing trajectories, the orbital-free DFT is intended to stand in the middle between force-field based MD and Kohn-Sham MD in terms of efficiency and accuracy. Indeed, while the forces are computed on-the-fly at each timestep, the optimization is done on the electronic density field instead of the Kohn-Sham orbitals. This feature avoids the need for satisfying the orthonormality constraint among orbitals and allows the computational complexity of the code to scale linearly with the dimensionality of the system. On the other hand, no information on the orbitals is available. The accuracy of the simulation relies on the choice of the kinetic energy functional, which has to be provided in terms of the electronic density alone.

Background Information

The module is standalone and only relies on the libraries discussed in the next section.

Installation

The execution of the code depends on the following libraries:

- **FFTW**: a library for computing the discrete Fourier transform;
- **Libxc**: a library of exchange-correlation functionals for density-functional theory;
- **BLAS**: a library that provides standard building blocks for performing basic vector and matrix operations;
- **Argp**: an interface for parsing unix-style argument vectors;

On macOS, [Homebrew](#) is strongly recommended to install compiler and dependencies.

The installation is based on a Makefile. A few machine dependent variable must be defined in the file ‘./config.mk’ prior to invoking the make utility. Examples can be found in the ‘./configuration_files/’ folder. The structure of the ‘./config.mk’ file is as follows:

```
#Compiler Configuration
#Compiler command
CC      =
#Compiler options
CFLAGS  =
#Includefiles linkers
IFLAGS  =
#Libraries linkers
LIBS    =

#Test configuration
#Python command
TEST    =
```

The command make will then build the executables. The command make clean cleans the files resulting from the compilation. Detailed documentation can be build using Doxygen through the command make documentation. The whole suite of regression tests can be run through the command make tests.

Testing

Tests for the code and for regressions are launched through a python script which can be found in ‘./tests/’. Move into this folder and run python regression_tests.py -s CG. The scripts can take other options in order to launch different suites of tests. Default is ‘all’ which can take up to 20 minutes. Run python regression_tests.py --help for more information on regression tests script.

By default the script tests an MD simulation of solid Sodium using different parameters:

- **Pseudopotential:** ‘Gaussian (Gauss)’ pseudopotential and ‘Topp and Hopfield (Topp)’ pseudopotential;
- **Jacob’s ladder rung:** ‘LDA’ for Local Density Approximation and ‘GGA’ for Generalized Gradient Approximation. The approximation refers only to the kinetic functional which is ‘Thomas-Fermi (TF)’ for LDA and ‘Thomas-Fermi plus von Weiszaecker correction (TFvW)’ and ‘Perrot’ functional for GGA;
- **Kinetic functional:** As above ‘Thomas-Fermi (TF)’, ‘Thomas-Fermi plus von Weiszaecker correction (TFvW)’ and ‘Perrot’ functionals;

All the simulation in the tests are run using a Slater exchange functional and no correlation functional.

The subfolders inside ‘./tests’ can also be conveniently used as examples and references for the format of the input file ‘runtime.inpt’ and of the configuration file ‘configuration.inpt’.

Source Code

The source code is available from the [E-CAM Gitlab](#) under the [MaZe](#) project.

The repository contains the following directories:

- **./source/:** contains the source code. The subfolder ‘./source/headers/’ contains the modules’ headers, while the subfolder ‘./source/obj/’ is used for compilation file outputs;
- **./tests/:** contains regression tests;
- **./scripts/:** contains useful python scripts to run simulations over different sets of parameters;
- **./documentation/:** contains the documentation generated with Doxygen together with the wiki of the project;

- `./configuration_files/`: contains examples of configuration files to generate the executable on different machines;

References

Additionally, a module from an ESDW of WP4 was more relevant to this Work Package and so is reported here

Software Technical Information

Name GPAW CUDA version: build instructions

Language Python, C, CUDA

Licence GPL

Documentation Tool Code comments. ReST and Sphinx for GPAW documentation.

Application Documentation <https://wiki.fysik.dtu.dk/gpaw/>

Software Module Developed by Martti Louhivuori (based on work by Samuli Hakala et al.)

GPAW CUDA version: build instructions

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

GPAW is a density-functional theory (DFT) program for ab initio electronic structure calculations using the projector augmented wave method. An experimental CUDA version is under development that supports running GPAW on GPUs. This module provides build instructions for the CUDA version and links to the development effort.

Purpose of Module

An experimental CUDA version of GPAW is under development with support for NVIDIA GPUS using CUDA, CuBLAS, and PyCUDA. This module provides build instructions for the CUDA version and links to the development effort.

Background Information

GPAW is a density-functional theory (DFT) program for ab initio electronic structure calculations using the projector augmented wave method. It uses a uniform real-space grid representation of the electronic wavefunctions that allows for excellent computational scalability and systematic converge properties.

GPAW is written mostly in Python, but includes also computational kernels written in C as well as leveraging external libraries such as NumPy, BLAS and ScaLAPACK. Parallelisation is based on message-passing using MPI.

To add support for GPUs, an experimental CUDA version of GPAW that uses CUDA, CuBLAS, and PyCUDA was developed (Samuli Hakala *et al.*, PARA 2012, https://doi.org/10.1007/978-3-642-36803-5_4). This early effort was

stalled for a while, but has now been continued in order to bring GPU support to modern GPAW versions. Current version is based on GPAW version 1.5.2, but active development is on-going.

Building and Testing

In addition to the normal software requirements of GPAW, the GPU version requires the [CUDA Toolkit](#) and the [PyCUDA](#) python module.

The only additional step to installing GPAW is that in the CUDA version one needs to build the CUDA kernels before building the rest of the GPAW. This is done in the `c/cuda/` directory that contains the CUDA kernels. There is a customisable Makefile (`make.inc`) that can be edited before running the `make` command.

So, the steps to install the CUDA version of GPAW are:

1. Edit the Makefile for the CUDA kernels (`c/cuda/make.inc`).

Modify the default options and paths to match your system. The essential parts are the include paths for libraries (MPI, CUDA) and the build options for `nvcc` to target the correct GPU architecture.

2. Build the CUDA kernels:

```
cd c/cuda
make
cd -
```

3. Edit the GPAW setup script (`customize.py`).

Add correct link and compile options for CUDA (and possibly other libraries). The relevant lines for CUDA are e.g.:

```
define_macros += [('GPAW_CUDA', '1')]
libraries += ['gpaw-cuda', 'cublas', 'cudart', 'stdc++']
library_dirs += [
    './c/cuda',
    '/path/to/cuda/lib64'
]
include_dirs += [
    '/path/to/cuda/include'
]
```

4. Build and install GPAW:

```
python setup.py install --prefix=$TARGET_DIRECTORY
```

Source Code

An up-to-date development version of GPAW with CUDA support is currently available at: <https://gitlab.com/mlouhivu/gpaw>. The current version is based on GPAW 1.5.2 and is available as [commit 111567ee](#).

Once it is merged with the upstream, the CUDA version will be available as a separate branch called ‘cuda’ in the main GPAW repository. Status of this work is tracked in [Merge Request !580](#).

To obtain the latest development version of the code, use the following command:

```
git clone -b cuda https://gitlab.com/mlouhivu/gpaw.git
```

or to get the version based on version 1.5.2, use the following commands:

```
git clone -b cuda https://gitlab.com/mlouhivu/gpaw.git
git checkout 111567ee39dd48e106b36b1aab4e6bc1b9961cae
```

2.3 Pilot Projects

One of primary activity of E-CAM is to engage with pilot projects with industrial partners. These projects are conceived together with the partner and typically are to facilitate or improve the scope of computational simulation within the partner. The related code development for the pilot projects are open source (where the licence of the underlying software allows this) and are described in the modules associated with the pilot projects.

Below is a list of the modules developed directly within the context of the pilot projects within E-CAM:

2.3.1 Geomoltools

Software Technical Information

This section describes the module Geomoltools.

Language Fortran 95

Documentation Tool Sphinx, ReStructuredText

Relevant Training Material See usage examples in the `tutorial` directory.

Licence GNU General Public License (GPL) version 2.

- *Purpose of Module*
- *Background Information*
- *Installation*
- *Testing*
- *Source Code*

Purpose of Module

Geomoltools is a set of computer codes designed to manipulate molecules. From simple changes of coordinates (Z-matrix to XYZ coordinates and *vice versa*) to more complicated operations as the generation of different stacking arrangements between molecules are quick and easy to perform.

Background Information

Geomoltools is a set of standalone codes to be employed independently or taking part in the development of a more general project.

Installation

The module does not need installation because is formed by independent pieces. The dependency of each single code with common subroutines are managed in the Makefile file, which leads an easy and straightforward compilation of the desired codes.

Testing

In the `tutorial` directory input/output files for each code are found and constitute examples of test and usage.

Source Code

The package, including all the files as well as technical help, can be found in the [E-CAM Gitlab](#) webpage.

2.3.2 GRASP Sampling

Software Technical Information

Language C

Licence MIT

Documentation Tool Doxygen

Purpose of the Module

This module performs a stratified sampling of the configurations, described by vectors, of a system to build a representative training set in a fitting procedure. Given a list of candidate configurations, and selected the size (N) of the training set required, the module executes the combinatorial optimization that maximizes the following dissimilarity score (DS) among the elements of the training set:

$$DS(N) = \frac{\frac{1}{N} \sum_l \frac{1}{M} \sum_j 2^{M-j} d_{lj}}{\sum_j 2^{M-j}}$$

In this formula, the j-th configuration in the sum is the j-th nearest one to the l-th configuration and d_{lj} is the Euclidean distance between the l-th and j-th configurations. M is the number of the nearest configurations considered in the score. The exponential weight makes the score near independent from the particular value of M, if it is larger than 4-6.

The combinatorial optimization that maximizes the dissimilarity score is performed using the GRASP algorithm. A stratified sampling can be performed without a combinatorial optimization using classical statistical techniques (for example Latin hypercube sampling), the GRASP sampling becomes useful when the selection is restricted to a predetermined set of configurations, generated or sampled with specific internal constraints. This is the case of the molecular configurations generated in a molecular dynamics simulation.

Background Information

The GRASP algorithm is illustrated in the paper “Feo T. A., Resende M. G., Greedy randomized adaptive search procedures. Journal of global optimization 6, 109-133 (1995)”. The application of the GRASP algorithm to perform a stratified sampling is described in the work “Force Field Parametrization of Metal Ions from Statistical Learning Techniques. J. Chem. Theory Comput., 2018, 14(1), pp 255-273”.

Input/Output Structure

The input file, “candidates.txt”, must have the form of an array $N_{Sample} \times N_{dim}$, where N_{Sample} is the number of the candidates and N_{dim} is the dimension of the vectors. The application of the algorithm provides the files “solution_v_A_B.txt” and “solution_i_A_B.txt” that includes the selected configurations and the corresponding indices respectively. “A” and “B” in the files of the solutions are the number of the selected configurations in the current application of the algorithm and the total number of configurations included in the solutions. A and B can differ because the algorithm can be applied in cumulative mode. The parameters of the algorithm are set in the file “input.gra” (an example is provided in the repository).

Source Code

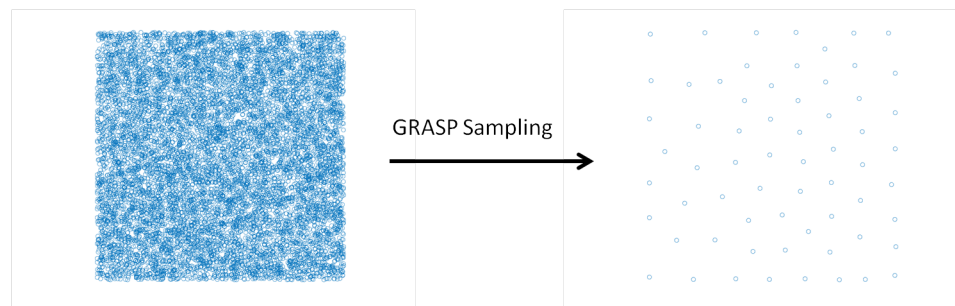
The source code of the algorithm is included in the “grasp.c” file, available from the [E-CAM Gitlab repository](#). To compile the code execute the Makefile. The GNU Scientific Library is necessary.

Testing

In the example of the “candidates.txt” file are collected 10000 two-dimensional vectors generated randomly, the application of the module produces the selection of a stratified subset that covers all the space of the basin sample. This can be appreciated observing the scatter plot of the two sets. The command to run the test is

```
$ ./grasp
```

provided that the “candidates.txt” and “input.gra” files are in the same directory of the executable file. Examples of “candidates.txt” and “input.gra” files are provided in the ./test directory. The name of “input.gra” file cannot be changed, while the name of file where the candidate configurations are defined must be indicated in the first line of the “input.gra” file.



2.3.3 GROMACS_Interface

Software Technical Information

Language Python

Licence MIT

Documentation Tool Doxygen

Purpose of the Module

This module reads the configurations of a molecular system generated by GROMACS and prepares the input for the GRASP Sampling module. The molecular system must contain a metal ion, that in the module is identified with the variable “metal_atom”. The module performs the following operations:

1. It reads the configurations stored as .gro files in the ./GROMACS_Configurations directory
2. It calculates, for each configuration, the Euclidean distances of all atoms from the metal ion.
3. It identifies the permutational equivalent atoms
4. It performs a Gaussian transformation of the distances
5. It calculates the variances of transformed distances
6. It selects the “v_len” coordinates with the higher variances
7. It prepares the input for the GRASP Sampling module as a matrix $N_{\text{conf}} \times v_{\text{len}}$ including the transformed distances for all the configurations stored in the ./GROMACS_Configurations directory.

Input/Output Structure

The “input_gromacs.txt” file must contain the following information:

1. the path where the .gro files are stored
2. the path of the file of the selected coordinates
3. the path where the output files are written
4. the number of the atoms of the system minus one
5. the size of the output vectors that describe the configurations
6. the readable format file of the input files (.gro)
7. the sigma value of the Gaussian transformation
8. the ID of the reference atom (the metal ion) in the Gromacs configurations

The module produces the “selected_coords.txt” file that identifies the v_{len} coordinates of the output matrix. The output files are: “d_store.txt” the matrix of the distances and “k_store.txt” the matrix of the transformed distances.

Source Code

The source code of the algorithm is available from the [E-CAM Gitlab repository](#). It is included in the file gromacs_interfaces.py, it is executed as follows:

```
$ python3.6 gromacs_interface.py
```

Testing

To test the module an example of the “input_gromacs.txt” file and a set of 100 configurations is provided in the ./test directory.

2.3.4 Gaussian_interface

Software Technical Information**Language** Python**Licence** MIT**Documentation Tool** Doxygen

Purpose of the Module

This module generates the Gaussian input files of clusters built by cutting of the configurations stored in .gro files. The clusters are used as model systems to perform fitting of force fields. The user must specify an atom of the configuration and a cut off distance (in nm). The module saturates the residues and prepares the input file for the calculation of the energy and the forces of the cluster generated in the Gaussian format.

Input/Output Structure

An “input.txt” file must be included in the working directory. It must contain the following information:

1. the index of the central atom of the clusters
2. the cut off distance (in nm) in the case of heterogeneous systems or the number of the solvent molecules in the case of solution systems
3. the extended path where the .gro files are stored
4. the extended path of the file of the output files
5. the level of theory for the calculation of energy and forces

The output module produces .com Gaussian input files.

Source Code

The source code of the algorithm is available from the [E-CAM Gitlab repository](#). It is included in the file Gaussian_interface.py. For the execution the following command line is required:

```
$ python3.6 Gaussian_interfaces.py
```

Testing

To test the module an example of the “input.txt” file and a set of 24 configurations is provided in the ./test directory.

2.3.5 Selectively-Localized-Wannier-Functions

Software Technical Information**Name** Selectively Localized Wannier Functions

Language Fortran90

Licence GPL

Documentation Tool Ford online link to different Wannier90 source files <http://www.wannier.org/ford/>

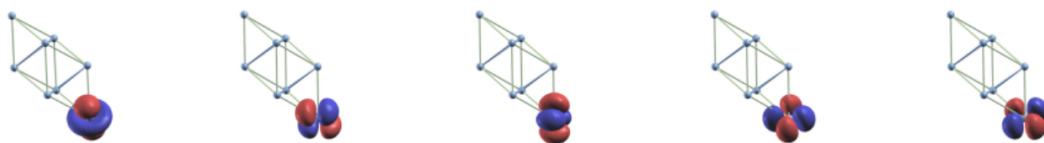
Application Documentation Wannier90 [User guide pdf](#) and [Tutorial pdf](#) and [Solution booklet pdf](#)

Relevant Training Material ‘Not currently available.’

Software Module Developed by Valerio Vitale

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Purpose of Module



This module is part of bundle

to extend the capabilities of the Wannier90 code¹. In particular, here we have implemented the algorithm from Marianetti “*et al.*”² to generate selectively localized Wannier functions which extend the method of Marzari and Vanderbilt³ in two important ways: 1) it allows the user to focus on localizing a subset of orbitals of interest and 2) to fix centres of these orbitals ensuring the preservation of the point-group symmetry. These features are very important when Wannier functions are used in beyond-density-functional-theory methods, such as DMFT, to study transport properties of novel technological relevant materials.

The module is part of the Wannier90 code.

Background Information

Wannier90 source code is available from the eponymous git-hub repository <http://github.com/wannier-developers/wannier90>, which contains the official repository. Documentation about the source code is done via FORD, an online version of this documentation is available [online](#). Instructions on how to install Wannier90 on a variety of architectures may be found in the [user guide](#). Quantum ESPRESSO source code is available from the git-hub repository <https://github.com/QEF/q-e>, and a very detailed web documentation may be found [here](#). Instruction for the installation of the python-based AiiDA workflow are available online at <http://aiida-core.readthedocs.io/en/stable/>.

¹ Comput. Phys. Commun. **185**, 2309 (2014)

² Phys. Rev. B **90**, 165125 (2014)

³ Phys. Rev. B **56**, 12847 (1997)

Building and Testing

For building the module one “simply” has to compile the Wannier90 code as explained in the online documentation. This will produce the executable `wannier90.x`, which contains the module.

Source Code

- [Link to a merge request containing my source code changes](#)
- [Link to my feature branch](#)

2.3.6 Differential Evolution (SHADE)

Software Technical Information

Language C

Licence MIT

Documentation Tool Doxygen

Software Module Developed by Francesco Fracchia

Purpose of the Module

This module performs a single-objective global optimization in a continuous domain using the metaheuristic algorithm Success-History based Adaptive Differential Evolution (SHADE). SHADE is a recent adaptive version of the differential evolution algorithm, a stochastic population-based derivative-free optimizer. The module is a component of the software tool LRR-DE, developed to parametrize force fields of metal ions. In particular, the role of the SHADE algorithm in LRR-DE is the optimization of the hyperparameters of the model. However the module has general applicability to the black-box minimization of a cost function.

The input of the module is the objective function, upper and lower limits of the domain for each dimension of the search space, and the parameters of the algorithm. The parameters of the algorithm are the size of the vector population, the maximum number of evaluations of the objective function, and the parameters of that govern the termination of the optimization. The output provides the coordinates of the identified minimum and the corresponding value of the objective function.

Background Information

The SHADE algorithm has been proposed by R. Tanabe and A. Fukunaga in the paper “Success-history based parameter adaptation for differential evolution.”, Evolutionary Computation (CEC), 2013 IEEE Congress on (pp. 71-78). It is an adaptive version of the differential evolution algorithm [Storn1997]. It is an evolutionary algorithm based on three main steps: mutation, crossover, selection.

Source Code

The source code of the algorithm is available from the [Differential Evolution repository](#). To compile the code execute the Makefile (including the demo.c file provided in the `./test` directory). The [GNU Scientific Library](#) is necessary (2.4 version tested). The `de.c` file contains the core of the code, in the `de.h` file the data types are defined.

Testing

The demo.c file in the ./test directory includes three test functions: sphere function, ellipse function, and the Michalewicz function. The command to run the test is

```
$ ./de NAMEFUNCTION
```

where NAMEFUNCTION can assume the values “sphere”, “ellips” or “michel”. The dimensionality of the functions is set equal to 20, however it can be modified in the demo.c file. The correct application of the module should identify the global minimum of the functions. They are provided in the file optima.dat in the ./test directory.

The termination of the optimization is defined by three criterions: 1) the maximum number of objective function evaluations 2) the variation of the mean value of the objective function of the population calculated in two cycle separated from a constant number of iterations 3) the difference between the mean value of the objective function of the population and the best value of the population. The criterion are tuned by three parameters defined in the demo.c file: settings.max_evaluations (default value equal to 200000), settings.step_delta is the number of the iterations that separate two checks of the mean value of the objective function of the population (default value equal to 40), settings.tolerance is the tolerance value for the criterions 2) and 3) (default value equal to 0.000001).

A further parameter is set in demo.c file, settings.size (default value equal to 100), that defines the size of the population of the vectors.

Further functions can be added in the demo.c file defining the functional form and the domain of the search space.

Software Technical Information

Name Weighted Linear Ridge Regression

Language C

Licence MIT

Documentation Tool Doxygen

Relevant Training Material Not currently available.

Software Module Developed by Francesco Fracchia

2.3.7 Weighted Linear Ridge Regression

- *Purpose of the Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Purpose of the Module

This module solves the weighted linear ridge regression problem calculating the linear parameters of a model selected by the user that minimize the deviations of the predictions from the references of the data set. Therefore, it is a supervised learning tool that optimizes the linear parameters of an analytical expression in order to fit a data set. Each element of the data set can be weighted according to the relative importance or reliability attributed by the user. The regularization provides a protection from the over-fitting, this inconvenient can occur if the flexibility of the model is

too high in relation to the available data. Moreover, the module calculates the leave-one-out cross-validation error for the employed data set. The WLRR module is a component of the LRR-DE software tool, developed to parametrize force fields of metal ions. In the LRR-DE software tool, the WLRR module is combined with the metaheuristic optimization algorithm differential evolution in order to tune the hyper-parameters of the model (the regularization parameter and the non-linear parameters of the model).

The LRR-DE module has been developed to parametrize force fields of metal ions, however the method can be applied to optimize the parameters of a general functional form with respect to reference data.

Background Information

The theoretical background of the LRR-DE procedure is illustrated in the paper [FF2018]. The LRR-DE procedure is a supervised learning methodology that combines the weighted linear ridge regression algorithm, to obtain the linear parameters of the model, with the differential evolution optimizer, to obtain the non-linear parameters of the model, using the leave-one-out cross-validation error as objective function. This module uses the GNU Scientific Library.

Building and Testing

To compile the code execute the Makefile (including the demo.c file provided in the `./test` directory). In `./test` directory a multi-objective data set is provided. The demo.c file includes an example for the definition of a model. The example is the parametrization of a force field with three components (Coulomb, Lennard-Jones 12-6) of the zinc ion in water with respect the solvation energy and the forces on the ion for a set of clusters. The linear parameters calculated by the module should be 2.40203305, 0.00001364, and -0.10986800. They appear in the third column of the output. The values of the first and second columns are the scaled parameters and the scaling factors respectively.

Source Code

The source code of the algorithm is available from the [Weighted Linear Ridge Regression repository](#). The `./source` directory includes two files: i) `wlrr.c` contains the functions that perform the scaling of the data, the operation of fitting and the calculation of the leave-one-out cross-validation error; ii) `wlrr.h` define the data types employed by `wlrr.c`.

Software Technical Information

Name Selected columns of density matrix Wannier functions

Language Fortran95

Licence [GPL](#)

Documentation Tool [Ford](#) online link to different Wannier90 source files <http://www.wannier.org/ford/>

Application Documentation Wannier90 [User guide pdf](#) and Quantum-ESPRESSO [documentation](#)

Relevant Training Material ‘Not currently available.’

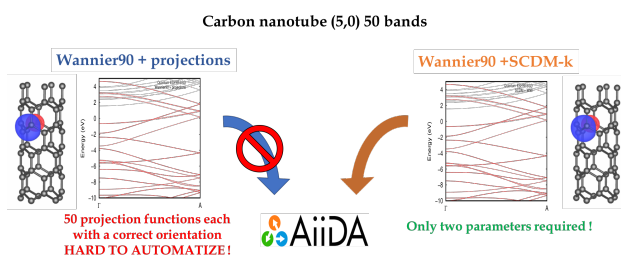
Software Module Developed by Valerio Vitale

2.3.8 SCDM Wannier Functions

- *Purpose of Module*

- [Background Information](#)
- [Building and Testing](#)
- [Source Code](#)

Purpose of Module



Wannier90¹ is a post-processing tool for the computation of the Maximally Localised Wannier Functions (MLWFs)^{2,3,4}, which have been increasingly adopted by the electronic structure community for different purposes. The reasons are manifold: **MLWFs** provide an insightful chemical analysis of the nature of bonding, and its evolution during, say, a chemical reaction. They play for solids a role similar to localized orbitals in molecular systems. In the condensed matter community, they are used in the construction of model Hamiltonians for, e.g., correlated-electron and magnetic systems. Also, they are

pivotal in first-principles tight-binding Hamiltonians, where chemically-accurate Hamiltonians are constructed directly on the Wannier basis, rather than fitted or inferred from macroscopic considerations, and many other applications, e.g. dielectric response and polarization in materials, ballistic transport, analysis of phonons, photonic crystals, cold atom lattices, and the local dielectric responses of insulators, for reference see². This module is a first step towards automation of **MLWFs**. In the original Wannier90 framework, automation of **MLWFs** is hindered by the difficult step of choosing a set of initial localized functions with the correct symmetries and centers to use as initial guess for the optimization. As a result, high throughput calculations (**HTC**) and big data analysis with **MLWFs** have proved to be problematic to implement.

The SCDM-k method⁵ removes the need for an initial guess altogether by using information contained in the single-particle density matrix. In fact, the columns of the density matrix are localised in real space, $\rho(\mathbf{r}, \mathbf{r}') \propto \exp[-\gamma|\mathbf{r} - \mathbf{r}'|]$ and can be used as a vocabulary to build the localised Wannier Functions. The SCDM-k method can be used in isolation to generate well localised WFs. More interestingly, is the possibility of coupling the SCDM-k method to Wannier90. The core idea is to use WFs generated by the SCDM-k method as initial guess in the optimisation procedure within Wannier90. This module is a big step towards automation of Wannier Functions and simplification of the use of the Wannier90 program. The module is therefore intended for all the scientists that benefit from the use of Wannier Functions in their research. Furthermore, by making the code more accessible and easier to use, this module will certainly increase the popularity of the Wannier90 code.

The module is part of the pw2wannier interface between the popular quantum ESPRESSO code [link](#) and Wannier90. It will be part of the next version of quantum ESPRESSO v.6.3 and Wannier90. Moreover, it has been successfully added in a developer branch of the [AiiDA workflow](#)⁶ to perform **HTC** on large material datasets.

Background Information

Wannier90 source code is available from the eponymous git-hub repository <http://github.com/wannier-developers/wannier90>, which contains the official repository. Documentation about the source code is done via FORD, an online version of this documentation is available [online](#). Instruction on how to install Wannier90 on a variety of architectures may be found in the [user guide](#). Quantum ESPRESSO source code is available from the git-hub repository <https://github.com/qe/qe>.

¹ Com. Phys. Comm. **178**, 685-699 (2008)

² Rev. Mod. Phys. **84**, 1419 (2012)

³ Phys. Rev. B **56**, 12847 (1997)

⁴ Phys. Rev. B **65**, 035109 (2001)

⁵ J.Comp.Phys. **334**, 1-15 (2017)

⁶ Comp. Mat. Sci. **111**, 218-230 (2016)

[//github.com/QEF/q-e](https://github.com/QEF/q-e), and a very detailed web documentation may be found [here](#). Instruction for the installation of the python-based AiiDA workflow are available online at <http://aiida-core.readthedocs.io/en/stable/>.

Building and Testing

For building the module one “simply” has to compile the quantum ESPRESSO program (v.6.2 and later), since the actual routine is inside the `pw2wannier90.f90` interface. This will produce the executable `pw2wannier90.x`. Instructions on how to achieve this are given in the quantum ESPRESSO web documentation. For testing the module one also needs the `wannier.x` executable.

Source Code

- [Link to a merge request containing my source code changes](#)
- [Link to the pull request for the documentation and examples](#)
- [Link to my feature branch](#)

Software Technical Information

Name Wannier90, AiiDA

Language Fortran90 for Wannier90, Python for AiiDA

Licence [GPL](#) for Wannier90, [MIT](#) for AiiDA

Documentation Tool [Ford](#) online link to different Wannier90 source files <http://www.wannier.org/ford/>, [ReST](#) for AiiDA

Application Documentation [Wannier90](#), [AiiDA](#)

Relevant Training Material See application documentation above

Software Module Developed by Giovanni Pizzi and Antimo Marrazzo in collaboration with Valerio Vitale

2.3.9 Automated high-throughput Wannierisation

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Maximally-localised Wannier functions (MLWFs) are routinely used to compute from first- principles advanced materials properties that require very dense Brillouin zone integration and to build accurate tight-binding models for scale-bridging simulations. At the same time, high-throughput (HT) computational materials design is an emergent field that promises to accelerate the reliable and cost-effective design and optimisation of new materials with target properties. The use of MLWFs in HT workflows has been hampered by the fact that generating MLWFs automatically and robustly without any user intervention and for arbitrary materials is, in general, very challenging. We address this problem directly by proposing a procedure for automatically generating MLWFs for HT frameworks. Our approach is based on the selected columns of the density matrix method (SCDM, see [SCDM Wannier Functions](#)) and is implemented in an AiiDA workflow.

Purpose of Module

Create a fully-automated protocol based on the SCDM algorithm for the construction of MLWFs, in which the two free parameters are determined automatically (in our HT approach the dimensionality of the disentangled space is fixed by the total number of states used to generate the pseudopotentials in the DFT calculations).

In the [paper derived from this work \[vitale2019\]](#), we apply our approach to a dataset of 200 bulk crystalline materials that span a wide structural and chemical space. We assess the quality of our MLWFs in terms of the accuracy of the band-structure interpolation that they provide as compared to the band-structure obtained via full first-principles calculations.

Background Information

This module is a collaboration between the E-CAM and [MaX](#) HPC centres of excellence.

In *SCDM Wannier Functions*, E-CAM has implemented the SCDM algorithm in the `pw2wannier90` interface code between the Quantum ESPRESSO software and the Wannier90 code. We have used this implementation as the basis for a complete computational workflow for obtaining MLWFs and electronic properties based on Wannier interpolation of the BZ, starting only from the specification of the initial crystal structure. We have implemented our workflow within the AiiDA materials informatics platform, and we used it to perform a HT study on a dataset of 200 materials.

Building and Testing

An AiiDA export file is provided with the full provenance of all simulations run in the project.

Source Code

See the [Materials Cloud entry](#). A downloadable virtual machine is provided that allows to reproduce the results of the associated paper and also to run new calculations for different materials, including all first-principles and atomistic simulations and the computational workflows.

2.3.10 W90_solution_booklet

Software Technical Information

Name Wannier90 Solution Booklet

Language LaTeX

Licence [GPL](#)

Documentation Tool

Application Documentation [Solution booklet pdf](#)

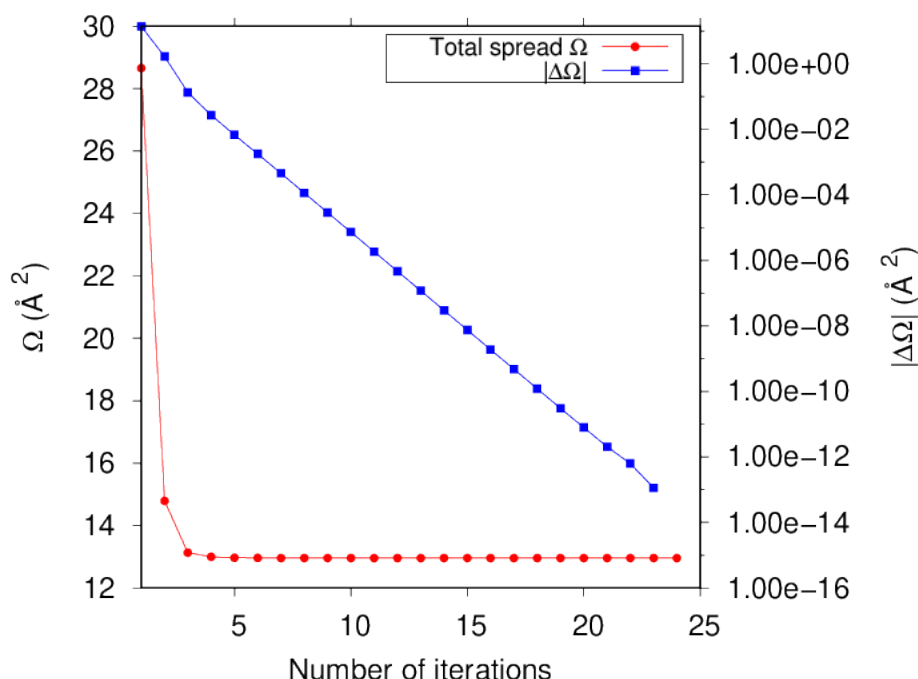
Relevant Training Material 'Not currently available.'

Software Module Developed by Valerio Vitale

- *Purpose of Module*
- *Background Information*

- *Building and Testing*
- *Source Code*

Purpose of Module



This is the solution booklet for the 21 examples in the Wannier90 (v2.1) Tutorial. It is a comprehensive LaTeX document that for each example describes in details the commands that need to be run and the

expected outputs, with texts, plots and figures. Moreover, in the text we tried to address most of the issues raised by Users in the Wannier90 mailing list. This module is mostly aimed at reducing the steepness of the learning curve for new Users.

The module is part of the Wannier90 code¹.

Background Information

Wannier90 source code is available from the following Git-hub repo <http://github.com/wannier-developers/wannier90>, which contains the official repository. Documentation about the source code is done via FORD, an online version of this documentation is available [online](#). Instructions on how to install Wannier90 on a variety of architectures may be found in the [user guide](#). Quantum ESPRESSO source code is available from the git-hub repository <https://github.com/QEF/q-e>, and a very detailed web documentation may be found [here](#).

Building and Testing

For generating the PDF file for the solution booklet one has to navigate to the `doc/solution_booklet/` folder and type “make”. This will produce the `solution_booklet.pdf` file, which contains the compiled text. You will need

¹ Comput. Phys. Commun. **185**, 2309 (2014)

pdflatex and bibtex to be installed.

Source Code

- [Link to a merge request containing my source code changes](#)
- [Link to my feature branch](#)

2.3.11 FFTXlib

Software Technical Information

Language Fortran 1995

Documentation Tool Sphinx, ReStructuredText

Application Documentation [Doc mirror](#)

Relevant Training Material See usage examples in the `examples` directory of the source code.

Licence GNU Lesser General Public License v3.0

Author of Module Emine Kucukbenli

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*
- *Further Information*

Purpose of Module

FFTXlib module is a collection of driver routines for complex 3D fast Fourier transform (FFT) libraries to be used within planewave-based electronic structure calculation software.

Generally speaking, FFT algorithm requires a data array to act on, a clear description of the input-output sequence and transform domains. In the context of planewave based electronic structure calculations, the data array may hold elements such as electronic wavefunction ψ or charge density ρ or their functions. The transform domains are direct (real) and reciprocal space, the discretization in real space is represented as a uniform grid of the unit cell and the discretization of the reciprocal space is in the basis of planewaves whose wavevectors are multiples of reciprocal space vectors (\mathbf{G}).

To understand the main motivation behind FFTXlib routines we need to clarify the differences between the representation of wavefunction and charge density in planewave based codes:

In these codes, the expansion of wavefunction in planewave basis is truncated at a cut-off wave-vector \mathbf{G}_{max} . Since density is the norm-square of the wavefunction, the expansion that is consistent with the one of wavefunctions requires a cut-off wavevector twice that of wavefunctions: $2\mathbf{G}_{max}$. Meanwhile, the real space FFT domain is often only defined by one uniform grid of the unit cell, so the array sizes of both ρ and ψ in their real space representation are the same.

Therefore, to boost optimization and to reduce numerical noise, the library implements two possible options while performing FFT: in one (‘Wave’) the wavevectors beyond \mathbf{G}_{max} are ignored, in the other (‘Rho’) no such assumption is made.

Another crucial feature of FFTXlib is that some approximations in the electronic structure calculations (such as usage of non-normconserving pseudopotentials) require that density is not just norm-square of wavefunctions, but has spatially localized extra components. In that case, these localized contributions may require higher G-vector components than the ones needed for density ($> 2\mathbf{G}_{max}$). Hence, in such systems, the density array in reciprocal space has more elements than the norm-conserving case (in other words a finer resolution or a denser grid is needed in real space) while the resolution needed to represent wavefunctions are left unchanged.

To accommodate for these different requirements of grid size, and to be able to make Fourier transforms back and forth between them, the FFTXlib routines explicitly require descriptor arguments which define the grids to be used. For example, if potential is obtained from density, the FFT operations on it should use the denser grid; while FFT on wavefunctions should use the smoother grid (corresponding to $2\mathbf{G}_{max}$ as defined before). When the Hamiltonian’s action on wavefunctions are being calculated, the potential should be brought from dense to smooth grid. But when the density is being calculated, wavefunction normsquare should be carried from smooth to dense grid.

A final important feature of FFTXlib is the index mapping. In the simple case of no parallelization, as a choice, the reciprocal space arrays are ordered in increasing order of $|\mathbf{G}|^2$ while the real space arrays are sorted in column major order. Therefore for FFT to be performed, a map between these two orders must be known. This index map is created and preserved by the FFTXlib.

In summary, FFTXlib allows the user to perform complex 3D fast Fourier transform (FFT) in the context of plane wave based electronic structure software. It contains routines to initialize the array structures, to calculate the desired grid shapes. It imposes underlying size assumptions and provides correspondence maps for indices between the two transform domains.

Once this data structure is constructed, forward or inverse in-place FFT can be performed. For this purpose FFTXlib can either use a local copy of an earlier version of FFTW (a commonly used open source FFT library), or it can also serve as a wrapper to external FFT libraries via conditional compilation using pre-processor directives. It supports both MPI and OpenMP parallelization technologies.

FFTXlib is currently employed within Quantum Espresso package, a widely used suite of codes for electronic structure calculations and materials modeling in the nanoscale, based on planewave and pseudopotentials. FFTXlib is also interfaced with “miniPWPP” module that solves the Kohn Sham equations in the basis of planewaves and soon to be released as a part of E-CAM Electronic Structure Library.

Background Information

FFTXlib is mainly a rewrite and optimization of earlier versions of FFT related routines inside Quantum ESPRESSO pre-v6; and finally their replacement. This may shed light on some of the variable name choices, as well as the default of $2\mathbf{G}_{max}$ cut-off for the expansion of the smooth part of the charge density, and the required format for lattice parameters in order to build the FFT domain descriptor. Despite many similarities, current version of FFTXlib dramatically changes the FFT strategy in the parallel execution, from 1D+2D FFT performed in QE pre v6 to a 1D+1D+1D one; to allow for greater flexibility in parallelization.

Building and Testing

A stable version of the module can be downloaded using [this link](#) .. when fftxlib has its own repo, this link can be moved there. Current installation and testing are done with gfortran compiler, version 4.4.7. The configuration uses GNU Autoconf 2.69.

The commands for installation are:

```
$ ./configure
$ make libfftx
```

As a result, the library archive “libfftx.a” is produced in `src` directory, and symbolically linked to a “lib” directory.

To see how the library works in a realistic case scenario of an electronic structure calculation, run:

```
$make FFTXexamples
```

A mini-app will be compiled in `src` directory and will be symbolically copied into `bin` directory. The mini-app simulates an FFT scenario with a test unit cell, and plane wave expansion cutoff. It creates the FFT structures and tests forward and backward transform on sample array and reports timings. Read the `README.examples` documentation in the `examples` subdirectory for further details.

Source Code

The FFTXlib bundle corresponding to the stable release can be downloaded from this [link](#). The source code itself can be found under the subdirectory `src`.

The development is ongoing.

The version that corresponds to the one of examples and tests can be obtained with SHA 31a6f4ecbb7ce474b0c87702c716713758f99a0a. This will soon be replaced with a version tag.

Further Information

This documentation can be found inside the `docs` subdirectory.

The FFTXlib is developed with the contributions of C. Cavazzoni, S. de Gironcoli, P. Giannozzi, F. Affinito, P. Bonfa’, Martin Hilgemans, Guido Roma, Pascal Thibaudeau, Stephane Lefranc, Nicolas Lacorne, Filippo Spiga, Nicola Varini, Jason Wood, Emine Kucukbenli.

2.3.12 W90_cube_format_non-orthogonal

Software Technical Information

Name Cube format files for non-orthogonal cells

Language Fortran90

Licence [GPL](#)

Documentation Tool [Ford](#) online link to different Wannier90 source files <http://www.wannier.org/ford/>

Application Documentation Wannier90 [User guide pdf](#) and [Tutorial pdf](#) and [Solution booklet pdf](#)

Relevant Training Material Training material is accessible via tests and examples as well as a tutorial and its solutions.

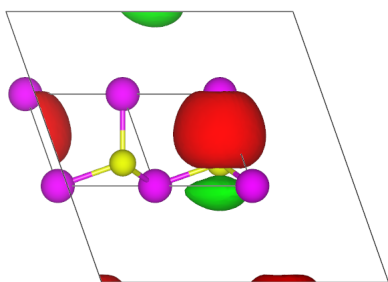
Software Module Developed by Valerio Vitale

- *Purpose of Module*

- *Background Information*
- *Building and Testing*
- *Source Code*

For many applications that rely on the detail of the electronic structure of solids, it is crucial to inspect the symmetries, centres and shapes of the Wannier functions. To this end, one needs to use a visualisation program such as VESTA¹ or VMD². One of the most popular format to encode volumetric data is the GAUSSIAN CUBE format, which is supported by almost all molecular visualization programs. This module extends the capability of Wannier90 in generating output files in the CUBE format for periodic calculations with non-orthogonal unit cells.

Purpose of Module



This module allows the User to output volumetric data, e.g. the Wannier functions on a real space grid, in the GAUSSIAN CUBE format even when the unit cell lattice vectors of the periodic calculation are non-orthogonal. The User can activate this feature by inserting the following two lines in the input file: *wannier_plot = .true.*

```
wannier_plot_format = cube
```

In addition, one can also specify the cut-off radius for the Wannier functions, i.e. the radius of the sphere that must fit inside the parallelepiped in which the Wannier function is plotted, via the *wannier_plot_radius* keyword. The number of atoms to plot with the volumetric data can be implicitly defined by the *wannier_plot_scale* keyword.

This module is part of the Wannier90 code³. The name of the subroutine that implements it is *internal_cube_format* and can be found in the *plot.F90* file within the *src* folder.

Background Information

Wannier90 source code is available from the official repository on GitHub <http://github.com/wannier-developers/wannier90>. Documentation of the source code is done via the FORD program, an online version of this documentation is available [online](#). Instructions on how to install Wannier90 on a variety of architectures may be found in the [user guide](#).

Building and Testing

For building the module one “simply” has to compile the Wannier90 code as explained in the online documentation. This will produce the executable *wannier90.x*, which contains the module. The source code can be found in the *src* folder within the *plot.F90* module.

¹

J. Appl. Crystallogr **44**, 1272-1276 (2011)

²

J. Molec. Graphics **14**, 33-38 (1996)

³

Comput. Phys. Commun. **185**, 2309 (2014)

Source Code

- [Link to a merge request containing my source code changes](#)

2.3.13 miniPWPP

Software Technical Information

Language Fortran 1995.

Documentation Tool Sphinx, ReStructuredText

Application Documentation [Doc mirror](#)

Relevant Training Material See usage examples in the `examples` directory of the source code.

Licence GNU Lesser General Public License v3.0

- *[Purpose of Module](#)*
- *[Features](#)*
- *[Building and Testing](#)*
- *[Source Code](#)*
- *[Further Information](#)*

Purpose of Module

miniPWPP is a barebone DFT code that uses plane wave basis set. Its purpose is to serve as a testbed, benchmark platform, and a demonstrator for modules and libraries that are created for pseudopotential-plane wave codes.

State of the art electronic structure packages cater to many possibilities: spin polarization, relativistic effects, several different treatments of electronic temperature etc. Because of this, their structure get more complicated over time, making it difficult to test new ideas, to benchmark core libraries, to profile different algorithms, or simply, to learn what exactly is happening under the hood of an electronic structure engine.

With miniPWPP, we present a simple, modularized electronic structure engine, with core capabilities (less than 10 main routines, each belonging to a single step of a DFT workflow) and with minimum decoration.

Due to this simple structure, miniPWPP can also serve for didactic purposes, both in physics and in information technologies. As a first example, in this module we demonstrate how the FFT interface library (FFTXlib) can be split from the rest of the electronic structure code; while retaining the matrix algebra library (LAXlib) as a subdirectory.

Features

Currently it uses pseudopotential form factors as defined by Cohen and Bergstresser in 1966. Pseudopotentials in Kleinman-Bylander form are not yet supported.

It supports solution of Kohn Sham equations at gamma or arbitrary k points. These two cases correspond to two distinct executables, in order to increase the transparency of each: `mpp_gamma.x` and `mpp.x`

It supports MPI parallelization.

It requires an FFT interface library (such as the one in E-CAM-Library: [FFTXlib module](#))

Building and Testing

A stable version of the module can be downloaded using [this link](#)

Current installation and testing are done with gfortran compiler, version 6.3.0. The configuration uses GNU Autoconf 2.69.

Here are the commands for installation:

```
$ tar -zxvf miniPWPP-1.0.tgz
$ ./configure
$ make miniPWPP
```

During configure, you can either specify the FFT interface library using the `FFTX_LIBS` and `FFTX_INCLUDE` variables:

```
$ ./configure FFTX_LIBS=/path/to/libfftx.a FFTX_INCLUDE=/path/to/fftx/modules/
```

If no library is specified, the FFTXlib module distributed from E-CAM-Library is downloaded, unpacked and used.

To test whether the module is working as expected, run:

```
.. $ make miniPWPP_ktest
.. $ make miniPWPP_gtest
```

The first tests the executable for the generic k point, while the second is for gamma point executable. The examples cover three different systems: free electrons in a periodic box, Silicon and GaAs crystals. By changing the input files in the examples directory, you can create your custom examples. Refer to `README.examples` file in the examples directory for further details.

Source Code

The miniPWPP bundle corresponding to the stable release can be downloaded from [this link](#). The source code itself can be found under the subdirectory `src`.

Further Information

This documentation can be found inside the `docs` subdirectory. Further diagonalization techniques such as Davidson and ParO will be added in the future.

The miniPWPP module is developed with the contributions of S. de Gironcoli, A. Chandran and E. Kucukbenli

2.3.14 PANNA-GVECT

Software Technical Information

Language Python 3.6.

Documentation Tool Sphinx, ReStructuredText

Application Documentation [Doc mirror](#)

Relevant Training Material See usage examples in the `doc/tutorial` directory of the source code.

Licence The MIT License (MIT)

- *Purpose of Module*
- *Features*
- *Building and Testing*
- *Usage*
- *Source Code*
- *Further Information*
- *References*

Purpose of Module

PANNA-GVECT module demonstrates how to efficiently generate Behler-Parinello and modified Behler-Parinello descriptors (see References^{1,2,3}).

These descriptors can then be used in machine learning algorithms. Even though these descriptors were originally designed for neural network models, they are equally suitable for other supervised learning schemes such as kernel methods, or unsupervised ones such as clustering techniques.

PANNA-GVECT, unlike other modules within the PANNA project, does not use TensorFlow framework.

Features

PANNA-GVECT supports periodic and aperiodic structures, multiple species, derivative of the descriptors with respect to atomic positions.

Building and Testing

A stable version of the module can be downloaded using the download button on this [page](#)

As a python module PANNA-GVECT does not require installation but it relies on numpy library version $\geq 1.15.0$.

In order to set up and test the module, run the following:

```
$ tar -zxvf panna-master.tar.gz
$ cd panna-master
$ python3 ./panna/test-gvect_calculator.py
```

¹ R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli. [arxiv:1907.03055](#). Submitted (2019).

² J. Behler and M. Parrinello, Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces, Phys. Rev. Lett. 98, 146401 (2007)

³ Justin S. Smith, Olexandr Isayev, Adrian E. Roitberg. ANI-1: An extensible neural network potential with DFT accuracy at force field computational cost. Chemical Science,(2017), DOI: 10.1039/C6SC05720A

Usage

PANNA-GVECT main script requires a configuration file that specifies the parameter of the calculation such as descriptor type, length etc. A typical command for using this module is as follows:

```
$ export PYTHONPATH=/path/to/panna/directory/panna
$ python3 gvect_calculator.py --config gvect_configuration.ini
```

A detailed tutorial about the contents of the configuration file can be found [here](#).

In this comprehensive tutorial, how use this module with other modules such as PANNA-TOOLS and PANNA-TFR is also demonstrated. Together, these modules cover all the steps necessary while going from raw data to descriptors that can be used in machine learning workflow.

Source Code

PANNA-GVECT source is currently hosted on [GitLab](#).

Further Information

The PANNA-GVECT module is developed with the contributions of R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli

References

PANNA manuscript:

and,

2.3.15 PANNA-TFR

Software Technical Information

Language Python 3.6.

Documentation Tool Sphinx, ReStructuredText

Application Documentation [Doc mirror](#)

Relevant Training Material See usage examples in the `doc/tutorial` directory of the source code.

Licence The MIT License (MIT)

- *[Purpose of Module](#)*
- *[Features](#)*
- *[Building and Testing](#)*
- *[Usage](#)*
- *[Source Code](#)*
- *[Further Information](#)*

- *References*

Purpose of Module

PANNA-TFR module demonstrates how to efficiently pack the Behler-Parinello and modified Behler-Parinello descriptor vectors (See References^{1,2,3}) written in binary format, into TensorFlow data format for efficient reading during training.

These descriptors can then be used within TensorFlow efficiently, reducing the overhead during batch creation. PANNA-TFR is built on TensorFlow.

Features

PANNA-TFR supports descriptors that change size across records, i.e. data points with different number of atoms are stored efficiently without padding.

Building and Testing

A stable version of the module can be downloaded using the download button on this [page](#)

As a python module PANNA-TFR does not require installation but it relies on numpy library version => 1.15.0 and tensorflow version => 1.13.0.

In order to set up and test the module, run the following:

```
$ tar -zxvf panna-master.tar.gz
$ cd panna-master
$ python3 ./panna/test-tfr-packer.py
```

Usage

PANNA-TFR main script requires a configuration file that specifies the parameter of the calculation such as location of descriptor files or how many descriptors to be packed in a single record file. A typical command for using this module is as follows:

```
$ export PYTHONPATH=/path/to/panna/directory/panna
$ python3 tfr_packer.py --config tfr_configuration.ini
```

A detailed tutorial about the contents of the configuration file can be found [here](#).

In this comprehensive tutorial, how use this module with other modules such as PANNA-GVECT and PANNA-TOOLS is also demonstrated. Together, these modules cover all the steps necessary while going from raw data to descriptors that can be used in machine learning workflow.

Source Code

PANNA-TFR source is currently hosted on [GitLab](#).

¹ R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli. [arxiv:1907.03055](#). Submitted (2019).

² J. Behler and M. Parrinello, Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces, Phys. Rev. Lett. 98, 146401 (2007)

³ Justin S. Smith, Olexandr Isayev, Adrian E. Roitberg. ANI-1: An extensible neural network potential with DFT accuracy at force field computational cost. Chemical Science,(2017), DOI: 10.1039/C6SC05720A

Further Information

The PANNA-TFR module is developed with the contributions of R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli

References

PANNA manuscript:

and,

2.3.16 PANNA-TRAIN

Software Technical Information

Language Python 3.6.

Documentation Tool Sphinx, ReStructuredText

Application Documentation [Doc mirror](#)

Relevant Training Material See usage examples in the `doc/tutorial` directory of the source code.

Licence The MIT License (MIT)

- *Purpose of Module*
- *Features*
- *Building and Testing*
- *Usage*
- *Source Code*
- *Further Information*
- *References*

Purpose of Module

PANNA-TRAIN is a neural network training module for atomistic data, eg. prediction of total energy and forces given a crystal structure. It implements a separate atomic network for each species, following the seminal work of Behler and Parrinello (see References^{1,2,3}) which can later be used as interatomic potential in molecular dynamics simulations.

PANNA-TRAIN uses TensorFlow framework as the underlying neural network training and data i/o engine.

¹ R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli. [arxiv:1907.03055](#). Submitted (2019).

² J. Behler and M. Parrinello, Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces, *Phys. Rev. Lett.* 98, 146401 (2007)

³ Justin S. Smith, Olexandr Isayev, Adrian E. Roitberg. ANI-1: An extensible neural network potential with DFT accuracy at force field computational cost. *Chemical Science*, (2017), DOI: 10.1039/C6SC05720A

Features

PANNA-TRAIN supports all to all connected networks for each species. Networks with different number of nodes and layers are allowed. It further supports controlling the training dynamics: eg. freeze/unfreeze layers, weight transfer, decaying learning rates etc.

Building and Testing

A stable version of the module can be downloaded using the download button on this [page](#)

As a python module PANNA-TRAIN does not require installation but it relies on numpy library version $\geq 1.15.0$, tensorflow version $\geq 1.13.0$, and tensorboard version $\geq 1.13.0$. Note that with version 2.0.0, tensorflow libraries went under substantial changes in structure, the 1.1X.X family supports the equally valid previous structure and is still being maintained. PANNA-TRAIN requires tensorflow 1.1X.X family of versions.

In order to set up and test the module, run the following:

```
$ tar -zxvf panna-master.tar.gz
$ cd panna-master
$ python3 ./panna/test-train.py
```

Usage

PANNA-TRAIN main script requires a configuration file that specifies the parameter of the calculation such as number of layers and nodes of each neural network layer, learning parameter etc. A typical command for using this module is as follows:

```
$ export PYTHONPATH=/path/to/panna/directory/panna
$ python3 train.py --config train_configuration.ini
```

A detailed tutorial about the contents of the configuration file can be found [here](#).

In this comprehensive tutorial, a neural network training scenario is demonstrated from beginning to end. Network validation is a key step in network training, hence in the tutorial how to use this module together with PANNA-EVAL module used in validation is also explained. Together, these two modules cover all the steps necessary to train an atomistic neural network, starting from a data which specifies the machine learning task in (input, target output) pair form.

Source Code

PANNA-TRAIN source is currently hosted on [GitLab](#).

Further Information

The PANNA-TRAIN module is developed with the contributions of R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli

References

PANNA manuscript:

and

2.3.17 PANNA-EVAL

Software Technical Information

Language Python 3.6

Documentation Tool Sphinx, ReStructuredText

Application Documentation [Doc mirror](#)

Relevant Training Material See usage examples in the `doc/tutorial` directory of the source code.

Licence The MIT License (MIT)

- *Purpose of Module*
- *Features*
- *Building and Testing*
- *Usage*
- *Source Code*
- *Further Information*
- *References*

Purpose of Module

PANNA-EVAL module evaluates an all to all connected neural network to predict atomistic quantities, e.g. total energy and forces of a given crystal structure.

PANNA-EVAL can be used with other modules of the PANNA project for neural network validation, but it can also serve to carry the information of the trained network to other platforms such as molecular dynamics code LAMMPS.

Although PANNA-EVAL does not need the advanced capabilities of the TensorFlow framework, it uses the ‘checkpoint’ information to automatically test the performance of a network from training data.

Features

PANNA-EVAL module has two user-end scripts: `evaluate.py` and `extract_weights.py`.

The main script of the PANNA-EVAL module, `evaluate.py` can evaluate all to all connected networks with various sizes for each species. It can also calculate the derivative of the target function, ie. forces for an energy network.

This module was primarily created to validate TensorFlow networks stored during training in checkpoint format, hence it has the functionality to look for checkpoint numbers in a training directory, and/or run several checkpoint evaluations at once.

The `extract_weights.py` script allows one to save the network parameters from TensorFlow native checkpoint format to other useful ones, such as human readable or LAMMPS potential formats. This last one allows neural networks that are trained and validated using PANNA¹ modules to be exported to LAMMPS as interatomic potentials.

¹ R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli. [arxiv:1907.03055](#). Submitted (2019).

Building and Testing

A stable version of the module can be downloaded using the download button on this [page](#)

As a python module PANNA-EVAL does not require installation but it relies on numpy library version $\geq 1.15.0$, TensorFlow version $\geq 1.13.0$. Note that with version 2.0.0, TensorFlow libraries went under substantial changes in structure, the 1.1X.X family supports the equally valid previous structure and is still being maintained. PANNA-EVAL requires TensorFlow 1.1X.X family of versions.

In order to set up and test the module, run the following:

```
$ tar -zxvf panna-master.tar.gz
$ cd panna-master
$ python3 ./panna/test-evaluate.py
```

Currently this test only assesses the `evaluate.py` script. Another test for `extract_weights.py` will be released in the near future.

Usage

PANNA-EVAL main script requires a configuration file that specifies the parameter of the calculation such as where to find the network to evaluate, which checkpoints to evaluate etc.. A typical command for using this module is as follows:

```
$ export PYTHONPATH=/path/to/panna/directory/panna
$ python3 evaluate.py --config val_config.ini
```

A detailed tutorial about the contents of the configuration file can be found [here](#).

In this comprehensive tutorial, a neural network training scenario is demonstrated from beginning to end. Network training and validation are two key steps of generating a predictive network, hence in the tutorial how to use this module together with PANNA-TRAIN module used in training is also explained. Together, these two modules cover all the steps necessary to train an atomistic neural network, starting from a data which specifies the machine learning task in (input, target output) pair form.

Source Code

PANNA-EVAL source is currently hosted on [GitLab](#).

Further Information

The PANNA-EVAL module is developed with the contributions of R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli.

References

PANNA manuscript:

2.3.18 PANNA-Charges

Software Technical Information

Language Python 3.6

Documentation Tool Sphinx, ReStructuredText

Application Documentation [Doc mirror](#)

Relevant Training Material See usage examples in the `doc/tutorial` directory of the source code.

Licence The MIT License (MIT)

- *Purpose of Module*
- *Features*
- *Building and Testing*
- *Usage*
- *Source Code*
- *Further Information*
- *References*

Purpose of Module

PANNA-Charges module demonstrates how to train a neural network to predict local atomic charges. This network can later be used to calculate the electrostatic energy density of a crystal. See Reference² for the theoretical model behind this approach.

PANNA-Charges, following other modules within the PANNA project¹, uses TensorFlow framework.

Features

PANNA-Charge supports periodic and aperiodic structures, multiple species, and a different all-to-all connected network architecture for each species. It further supports controlling the training dynamics: eg. freeze/unfreeze layers, weight transfer, decaying learning rates etc.

Building and Testing

A stable version of the module can will be released in the near future, and will be available for download using the download button on this [page](#)

As a python module PANNA-Charges does not require installation but it relies on numpy library version $\geq 1.15.0$, tensorflow version $\geq 1.13.0$, and tensorboard version $\geq 1.13.0$. Note that with version 2.0.0, tensorflow libraries went under substantial changes in structure, the 1.1X.X family supports the equally valid previous structure and is still being maintained. PANNA-TRAIN requires tensorflow 1.1X.X family of versions.

In order to set up and test the module, run the following:

² N. Artrith, T. Morawietz, J. Behler. PRB 83, 153101 (2011). High-dimensional neural-network potentials for multicomponent systems: Applications to zinc oxide. Erratum: PRB 86, 079914 (2012).

¹ R. Lot, Y. Shaidu, F. Pellegrini, E. Kucukbenli. [arxiv:1907.03055](#). Submitted (2019).

```
$ tar -zxvf panna-master.tar.gz
$ cd panna-master
$ python3 ./panna/test-charges-train.py
```

Usage

PANNA-Charges main script, `charges_train.py`, requires a configuration file that specifies the parameter of the calculation such as number of layers and nodes of each neural network layer, learning parameter etc. A typical command for using this module is as follows:

```
$ export PYTHONPATH=/path/to/panna/directory/panna
$ python3 charges_train.py --config charges_train_config.ini
```

A detailed tutorial about the contents of the configuration file will be released [here](#).

In this comprehensive tutorial, a neural network training scenario for systems with long range interactions will be demonstrated.

Source Code

PANNA-Charges source is not currently public, when it is released it will be hosted on [GitLab](#).

Further Information

The PANNA-Charges module is developed with the contributions of Y. Shaidu, R. Lot, F. Pellegrini, E. Kucukbenli.

References

PANNA manuscript:
and

2.3.19 QMCPack interfaces

The following modules related to interfaces for the QMCPack code have been produced so far in the context of an associated [Pilot Project](#):

Software Technical Information

Name ESInterfaceBase

Language C++

Licence

Documentation Tool Doxygen

Relevant Training Material Not currently available

Software Module Developed by Michele Ruggeri, Raymond C. Clay III

ESInterfaceBase

- *Purpose of Module*
- *Background Information*
- *Building and testing*
- *Source Code*

Purpose of Module

To obtain accurate results with ground state Quantum Monte Carlo methods (such as Variational and Diffusion Monte Carlo) an accurate trial wave function is essential. Such a wave function for an electron system will be typically given by the product of two factors: (1) a Jastrow term J describing electronic correlations and (2) a Slater determinant of suitable single particle orbitals ϕ_i

$$\Psi(\mathbf{R}) = J(\mathbf{R}) \cdot \text{Det}(\phi_i(\mathbf{r}_j))$$

where R is the vector containing the position of all electrons and r_i is the position of the i -th electron. While there is great freedom in the definition of the Jastrow term, that can then be variationally optimized, the single particle orbitals have to be computed in using Density Functional Theory.

The ESInterfaceBase module provides a base class for a general interface to generate single particle orbitals for QMC simulations performed using QMCPack; implementations of specific interfaces as derived classes of ESInterfaceBase are available as separate modules.

Background Information

QMCPack is available from the github repository <https://github.com/QMCPACK/qmcpack>, and the documentation can be found at the QMCPack website <https://qmcpack.org/documentation>.

Building and testing

The ESInterfaceBase module can be found in the QMCQEPack branch of the QMCPack git repository <https://github.com/michruggeri/qmcpack/tree/QMCQEPack>. After cloning and getting to the QMCQEPack branch with

```
git clone https://github.com/michruggeri/qmcpack.git
git checkout QMCQEPack
```

one can proceed to build the QMCPack software, as detailed in the official QMCPack documentation <https://qmcpack.org/documentation>, or in the manual available in the `manual` subdirectory in the main QMCPack directory.

The tests for this code are part of the deterministic unit tests for QMCPack, that can be run with the command

```
ctest -R interface
```

Note that the code is tested using the GCC compiler and OpenMPI.

Source Code

The source code is available available from <https://github.com/michruggeri/qmcpack/tree/QMCQEPack> in the QMCQEPack branch. Specifically relevant files for this module include:

- `src/Interfaces/ESInterfaceBase.cpp`
- `src/Interfaces/ESInterfaceBase.h`
- `src/Interfaces/InterfaceBase.cpp`
- `src/Interfaces/InterfaceBase.h`

Software Technical Information

Name ESHDF5Interface

Language C++

Licence

Documentation Tool Doxygen

Relevant Training Material Not currently available

Software Module Developed by Michele Ruggeri, Raymond C. Clay III

ESHDF5Interface

- *Purpose of Module*
- *Background Information*
- *Building and testing*
- *Source Code*

Purpose of Module

To obtain accurate results with ground state Quantum Monte Carlo methods (such as Variational and Diffusion Monte Carlo) an accurate trial wave function is essential. Such a wave function for an electron system will be typically given by the product of two factors: (1) a Jastrow term J describing electronic correlations and (2) a Slater determinant of suitable single particle orbitals ϕ_i

$$\Psi(\mathbf{R}) = J(\mathbf{R}) \cdot \text{Det}(\phi_i(\mathbf{r}_j))$$

where R is the vector containing the position of all electrons and r_i is the position of the i -th electron. While there is great freedom in the definition of the Jastrow term, that can then be variationally optimized, the single particle orbitals have to be computed in using Density Functional Theory.

The ESHDF5Interface module provides a derived class of ESInterfaceBase to generate single particle orbitals for QMC simulations performed using QMCPack from a suitable HDF5 file.

Background Information

QMCPack is available from the github repository <https://github.com/QMCPACK/qmcpack>, and the documentation can be found at the QMCPack website <https://qmcpack.org/documentation>.

Building and testing

The ESHDF5Interface module can be found in the QMCQEAPack branch of the QMCPack git repository <https://github.com/michruggeri/qmcpack/tree/QMCQEAPack>. After cloning and getting to the QMCQEAPack branch with

```
git clone https://github.com/michruggeri/qmcpack.git
git checkout QMCQEAPack
```

one can proceed to build the QMCPack software, as detailed in the official QMCPack documentation <https://qmcpack.org/documentation>, or in the manual available in the manual subdirectory in the main QMCPack directory.

To use the interface one must use the `interfaceh5` keyword in the `determinantset` block in a QMCPack input file; further information can be found in Section 22.5.2 of the QMCPack manual, that can be compiled with the files in the `manual` directory.

The tests for this code are part of the deterministic unit tests for QMCPack, that can be run with the command

```
ctest -R interface
```

Note that the code is tested using the GCC compiler and OpenMPI.

Source Code

The source code is available available from <https://github.com/michruggeri/qmcpack/tree/QMCQEAPack> in the QMCQEAPack branch. Specifically relevant files for this module include:

- `src/Interfaces/ESHDF5/ESHDF5Interface.cpp`
- `src/Interfaces/ESHDF5/ESHDF5Interface.h`

and for the tests:

- `src/Interfaces/tests/test_interface_HDF5.cpp`
- `src/Interfaces/tests/O.BFD.upf`

Software Technical Information

Name ESPWSCFInterface

Language C++

Licence

Documentation Tool Doxygen

Relevant Training Material Not currently available

Software Module Developed by Michele Ruggeri, Raymond C. Clay III

ESPWSCFInterface

- *Purpose of Module*
- *Background Information*
- *Building and testing*
- *Source Code*

Purpose of Module

To obtain accurate results with ground state Quantum Monte Carlo methods (such as Variational and Diffusion Monte Carlo) an accurate trial wave function is essential. Such a wave function for an electron system will be typically given by the product of two factors: (1) a Jastrow term J describing electronic correlations and (2) a Slater determinant of suitable single particle orbitals ϕ_i

$$\Psi(\mathbf{R}) = J(\mathbf{R}) \cdot \text{Det}(\phi_i(\mathbf{r}_j))$$

where R is the vector containing the position of all electrons and r_i is the position of the i -th electron. While there is great freedom in the definition of the Jastrow term, that can then be variationally optimized, the single particle orbitals have to be computed in using Density Functional Theory.

The ESPWSCFInterface module provides a derived class of ESInterfaceBase to generate single particle orbitals for QMCPack via a DFT computation performed with Quantum Espresso.

Background Information

QMCPack is available from the github repository <https://github.com/QMCPACK/qmcpack>, and the documentation can be found in the QMCPack website <https://qmcpack.org/documentation>.

Quantum Espresso can be installed using the module *QMCQEPack_gepatch*, and the documentation can be found in the Quantum Espresso website <https://www.quantum-espresso.org/resources/users-manual>.

Building and testing

The ESPWSCFInterface module can be found in the QMCQEPack branch of the QMCPack git repository <https://github.com/michruggeri/qmcpack/tree/QMCQEPack>. After cloning the repository and checking out the QMCQEPack branch with

```
git clone https://github.com/michruggeri/qmcpack.git
git checkout QMCQEPack
```

one can proceed to download Quantum Espresso and build the libpwinterface.so library using the *QMCQEPack_gepatch* module.

Once the library is built one can proceed to build and compile QMCPack, as detailed in the official QMCPack documentation <https://qmcpack.org/documentation>, or in the manual available in the manual subdirectory in the main QMCPack directory. Note that to use the Quantum Espresso interface the cmake options QE_INTERFACE must be used, typically with

```
cmake -DQE_INTERFACE=1 -DQMC_COMPLEX=1 <QMCPack base directory>
```

before compiling with `make`.

To use the interface one must use the `qmcqepack` keyword in the `determinantset` block in a QMCPack input file; further information can be found in Section 22.5.3 of the QMCPack manual, that can be compiled with the files in the `manual` directory.

The tests for this code are part of the deterministic unit tests for QMCPack, that can be run with the command

```
ctest -R interface
```

Note that the code is tested using the GCC compiler and OpenMPI.

Source Code

The source code is available available from <https://github.com/michruggeri/qmcpack/tree/QMCQEpack> in the QMCQEpack branch. Specifically relevant files for this module include:

- `src/Interfaces/PWSCF/ESPWSCFInterface.cpp`
- `src/Interfaces/PWSCF/ESPWSCFInterface.h`
- `src/Interfaces/PWSCF/pwinterface.h`

and for the tests:

- `src/Interfaces/tests/pwscf.in`
- `src/Interfaces/tests/test_interface_PWSCF.cpp`

Software Technical Information

Name QMCQEpack_gepatch

Language Fortran90

Licence

Documentation Tool Doxygen

Relevant Training Material Not currently available

Software Module Developed by Michele Ruggeri, Raymond C. Clay III

QMCQEpack_gepatch

- *Purpose of Module*
- *Background Information*
- *Building and testing*
- *Source Code*

Purpose of Module

To obtain accurate results with ground state Quantum Monte Carlo methods (such as Variational and Diffusion Monte Carlo) an accurate trial wave function is essential. Such a wave function for an electron system will be typically given

by the product of two factors: (1) a Jastrow term J describing electronic correlations and (2) a Slater determinant of suitable single particle orbitals ϕ_i

$$\Psi(\mathbf{R}) = J(\mathbf{R}) \cdot \text{Det}(\phi_i(\mathbf{r}_j))$$

where R is the vector containing the position of all electrons and r_i is the position of the i -th electron. While there is great freedom in the definition of the Jastrow term, that can then be variationally optimized, the single particle orbitals have to be computed in using Density Functional Theory.

The `QMCQEPack_qepatch` provides the files to properly patch Quantum Espresso 5.3 to build the `libpwinterface.so` library; this library is required to use the module *ESPWSCFInterface* to generate single particle orbitals during a QMCPack computation using Quantum Espresso.

Background Information

QMCPack is available from the github repository <https://github.com/QMCPACK/qmcpack>, and the documentation can be found in the QMCPack website <https://qmcpack.org/documentation>.

Quantum Espresso can be installed using this module, and the documentation can be found in the Quantum Espresso website <https://www.quantum-espresso.org/resources/users-manual>.

Building and testing

The `QMCQEPack_qepatch` module can be found in the `QMCQEPack` branch of the QMCPack git repository <https://github.com/michruggeri/qmcpack/tree/QMCQEPack>. After cloning the repository and checking out the `QMCQEPack` branch with

```
git clone https://github.com/michruggeri/qmcpack.git
git checkout QMCQEPack
```

one can proceed to build the `libpwinterface.so` library using the using the script `QMCQEPack_download_and_patch_qe.sh` in the `external_codes/quantum_espresso` directory. After patching the code one has to use the `configure` script in the resulting `q-e-qe-5.3` directory and finally compile the `libpwinterface.so` library with `make pw`. Note that when building the code it may be required to use the internal Quantum Espresso version of the FFTW libraries. In order to do so if is sufficient to change in the `DFLAGS` field of the `make.sys` file generated by the `configure` script `-D__FFTW3` with `-D__FFTW`.

To use this library to perform DFT simulations QMCPack must be suitably compiled; the relevant information can be found in the documentation of the *ESPWSCFInterface* module.

The tests for this code are part of the deterministic unit tests for QMCPack, that can be run with the command

```
cctest -R interface
```

Note that the code is tested using the GCC compiler and OpenMPI.

Source Code

The source code is available available from <https://github.com/michruggeri/qmcpack/tree/QMCQEPack> in the `QMCQEPack` branch. Specifically relevant files for this module include:

- `external_codes/quantum_espresso/QMCQEPack_qepatch.diff`
- `external_codes/quantum_espresso/QMCQEPack_download_and_patch_qe.sh`

2.3.20 Caesar

The following modules related to the development of the [Caesar](#) software package which calculates the vibrational free energy, and a number of related vibrational properties, of periodic crystals.

Software Technical Information

Name [Caesar](#)

Language [Fortran](#)

Licence [GNU Lesser General Public License version 3](#)

Documentation Tool [Ford](#)

Application Documentation See the [Caesar repository](#)

Software Module Developed by [Mark Johnson](#)

Caesar; a utility for calculating the vibrational free energy of periodic crystals

- *Purpose of Module*
- *Building and Testing*
 - *Compilation*
 - *Dependencies*
 - *Documentation and Helptext*
 - *Unit Tests*
 - *Output Visualisation*
- *Performing Calculations*
- *Source Code*

Caesar calculates the vibrational free energy, and a number of related vibrational properties, of periodic crystals.

Purpose of Module

Caesar is intended to provide a vibrational method which is more accurate than the widely-used harmonic approximation [[Hoja_ea](#)] and the more sophisticated effective harmonic approximation [[Errea_ea](#)], but which is computationally inexpensive enough to be integrated into high-throughput workflows.

Caesar can calculate vibrational properties using several vibrational methods. The *Caesar Harmonic Calculation Library* performs calculations under the harmonic approximation [[Hoja_ea](#)]. The *Caesar Anharmonic Calculation Library* performs calculations under the vibrational self-consistent harmonic approximation (VSCHA) [[Errea_ea](#)] or using vibrational self-consistent field theory (VSCF) [[Christiansen](#)].

In order to perform vibrational calculations, Caesar must interface with an electronic structure code. A wide range of electronic structure codes can be used, via the *Caesar electronic structure interface*.

Building and Testing

The details of how to build and testing Caesar are given in the [Caesar README.txt](#) file.

Details of how the documentation and unit tests were written are presented in *Caesar - Documentation and Testing*.

Compilation

An out-of-source build is recommended. For this, a clean `build` directory should be made, and then **CMake** and **Make** should be run from the `build` directory, e.g. as

```
mkdir build
cd build
cmake [options] path_to_src
make
```

where `[options]` are the desired **CMake** configuration options, and `path_to_src` is the path to the `caesar/src` directory.

Caesar has been tested using version 10.1 of the `gfortran` compiler.

Dependencies

Caesar requires the **spglib** crystal symmetry library. **CMake** will search `LIB` for **spglib**'s `lib` directory, and will search `PATH` for **spglib**'s `include` directory.

Caesar also requires the **BLAS** and **LAPACK** linear algebra libraries. These are located using **CMake**'s **FindLAPACK** utility, which searches a range of standard install locations and can be configured using additional **CMake** configuration options.

These dependencies can be suppressed by setting the **CMake** options `LINK_TO_SPGLIB` and `LINK_TO_LAPACK` to false, although this will disable many of Caesar's features.

Documentation and Helptext

The software documentation for Caesar can be generated using **Ford**. This should be generated in the `doc` directory, by calling

```
ford caesar.md
```

Documentation will be generated in the `doc/ford` directory, and `doc/ford/index.html` can be viewed by using an html reader (e.g. a web browser).

Caesar also has its own helptext system, which can be accessed through the `caesar` executable by calling `caesar --help`. This system includes helptext for each of the modes in which Caesar can be called, including details of the input settings for each mode.

Unit Tests

The unit tests for Caesar are generated using **pFUnit**. When building with tests, **pFUnit** becomes a dependency of Caesar, and **CMake** will search `PATH` for **pFUnit**'s `bin` directory.

Unit tests can be run by calling

```
ctest
```

from the `build` directory where `CMake` was run.

Unit tests are built by default, but can be suppressed by setting the `CMake` option `ENABLE_TESTS` to false.

Output Visualisation

Caesar uses `python` scripts to visualise output data. These can be run using Caesar, or can be run directly. When Caesar is built, the `python` scripts will be written to the `python` directory within the `build` directory.

Performing Calculations

Caesar is a command line utility. The behaviour of Caesar can be controlled using command line options, a configuration file, interactive input, or a combination of these. Detailed usage information can be obtained by calling

```
caesar --help
```

Source Code

The source code for Caesar is available from the [Caesar repository](#)

Software Technical Information

Name Caesar

Language Fortran

Licence GNU Lesser General Public License version 3

Documentation Tool Ford

Application Documentation See the [Caesar repository](#)

Software Module Developed by Mark Johnson

Caesar - Documentation and Testing

- *Purpose of Module*
 - *Unit Tests*
 - *Documentation*
 - *Helptext*
 - *Submodules*
- *Source Code*

Purpose of Module

The features described in this module aim to make *Caesar* easier to use, maintain and develop, by anyone who is interested in doing so.

Unit Tests

Caesar uses `pFUnit` to build and run unit tests. The unit test files used by `pFUnit` are preprocessed into fortran, and so the unit tests can be built and run alongside the rest of the code, using `CMake`.

Each module file `*.f90` has an accompanying test file, `*_test.pf`, containing the test procedures for that module. It is impractical for the unit tests to be exhaustive, and so they aim to cover several standard use cases as well as any obvious edge cases. In particular, use cases which were known to cause bugs in previous versions of *Caesar* are included in the unit tests, as a form of regression testing.

Documentation

The documentation for *Caesar* is generated using `Ford`. `Ford` generates documentation directly from the fortran source code, and so the documentation for each procedure is written in the same place as the interface for that procedure.

Each procedure has documentation describing what the procedure does, what the input and output arguments to the procedure are, and what happens in the case of an error. The details of how each procedure works are presented separately, as code comments in the implementation of each procedure.

Helptext

Caesar uses a custom system to process the input arguments for the various user-accessible procedures.

Each input argument is defined using a `KeywordData` type, which stores the name of the argument, a helptext string describing the argument, and relevant metadata including whether or not the argument is optional, and what the argument's default value is, if relevant.

Once the array of input arguments for a given procedure has been created, it is passed to the input parsing system. The input parsing system then gets the value of each argument from the user, either from command line arguments, from an input file, or from an interactive procedure which describes each argument to the user in turn, and prompts them to input the argument values.

Once the inputs are parsed, the relevant *Caesar* subroutine is called, with a dictionary-style container, containing the input arguments as key-value pairs. Each input argument can then be accessed by name, and the container returns the string representation of the argument's value, which can be parsed by the subroutine.

In addition to driving the input parser, the list of arguments is used to generate helptext. Calling `caesar [mode] --help` generates the list of arguments for the requested `mode`, and prints the helptext for each argument in turn.

Submodules

As part of the process of writing documentation and unit tests, every procedure in *Caesar* was separated into an interface and an implementation, through the use of fortran `submodules`. This has a number of advantages:

- Submodules allow for circular dependencies which break free from fortran's strict module hierarchy. This allows modules to be separated into inter-dependent libraries, grouped together based on their functionality rather than their dependencies.

- Circular dependencies make it much easier to add new implementations of abstract classes, as functions which “know about” (i.e. have dependencies on) all of the class implementations can be called by the methods of the parent class. This means that *Caesar* can easily be updated to use new potential and state representations, sampling methods, and convergence algorithms, without requiring future developers to modify the code in more than a couple of places.
- The documentation for the procedure interfaces and the procedure implementations is separated. This allows for easy browsing of interface documentation, with implementation documentation hidden until needed.
- Unit tests can be checked for code coverage by comparing the test files with the interface files, without having to consider the implementation files.
- When procedure implementations are modified, only the modified submodules need to be re-compiled. This avoids the compilation cascades which are endemic to module-only fortran projects, and dramatically reduces total compilation time for developers.

Source Code

The source code for Caesar is available from the [Caesar repository](#). The source code for the helptext system is found in the `src/common/arguments` directory of this repository.

Software Technical Information

Name Caesar electronic structure interface

Language Fortran

Licence [GNU Lesser General Public License version 3](#)

Documentation Tool [Ford](#)

Application Documentation See the [Caesar repository](#)

Software Module Developed by Mark Johnson

Caesar electronic structure interface

- *Purpose of Module*
- *The Electronic Structure Run Script*
- *Interfaces to Other Electronic Structure Interfaces*
- *Source Code*

Purpose of Module

Calculating vibrational properties requires a mapping of the nuclear potential energy surface (PES) $V(\mathbf{r})$, where \mathbf{r} is the collective coordinate describing the locations of the nuclei. *Caesar* maps the PES by sampling it at a number of nuclear configurations \mathbf{r}_i .

In software terms, each PES sample represents a single electronic structure calculation, where the electronic structure code is given the nuclear configuration \mathbf{r}_i , and calculates the value of the PES at that configuration, $V(\mathbf{r}_i)$, optionally along with other quantities such as the forces $\mathbf{f}(\mathbf{r}_i) = -\frac{\partial}{\partial \mathbf{r}} V|_{\mathbf{r}_i}$ and the Hessian matrix $H(\mathbf{r}_i) = \frac{\partial}{\partial \mathbf{r}} \frac{\partial}{\partial \mathbf{r}} V|_{\mathbf{r}_i}$.

The Caesar electronic structure interface enables *Caesar* to be used with a wide range of electronic structure codes, by treating each electronic structure calculation as a black box.

The Electronic Structure Run Script

Caesar generates a nested directory structure, with the input file for each configuration r_i written to its own calculation directory. A user-provided run script is then called repeatedly, once for each calculation directory. This script is expected to read any required parameters from the root directory, read the input file from the calculation directory, and then call the electronic structure code and write the electronic structure results to an output file in the calculation directory.

The calculation input and output files can be in a number of formats used by existing electronic structure codes, or they can be in a simple plain-text format.

The plain-text input file, `structure.dat` is formatted as

```
Lattice
L_xx L_xy L_xz
L_yx L_yy L_yz
L_zx L_zy L_zz
Reciprocal Lattice
L'_xx L'_xy L'_xz
L'_yx L'_yy L'_yz
L'_zx L'_zy L'_zz
Atoms
z_1 r_1x r_1y r_1z
z_2 r_2x r_2y r_2z
...
z_n r_nx r_ny r_nz
Supercell
S_xx S_xy S_xz
S_yx S_yy S_yz
S_zx S_zy S_zz
Reciprocal Supercell
S'_xx S'_xy S'_xz
S'_yx S'_yy S'_yz
S'_zx S'_zy S'_zz
R-vectors
R_1x R_1y R_1z
R_2x R_2y R_2z
...
R_Nx R_Ny R_Nz
G-vectors
G_1x G_1y G_1z
G_2x G_2y G_2z
...
G_Nx G_Ny G_Nz
End
```

where:

- L is the supercell lattice matrix, whose rows are the lattice vectors of the supercell in which the electronic structure calculation should be performed.
- L' is the reciprocal supercell lattice matrix of the supercell, defined as $L' = L^{-T}$.
- S is the supercell matrix, which relates the supercell lattice matrix L to the primitive cell lattice matrix L_p as $L = SL_p$.

- S' is the reciprocal supercell matrix, defined as $S' = S^{-T}$.
- z_i and \mathbf{r}_i are the species label and cartesian coordinate of the i 'th atom.
- $\{R_i\}$ are the R-vectors of the primitive cell which are contained within the supercell.
- $\{G_i\}$ are the G-vectors of the reciprocal supercell which are contained within the reciprocal primitive cell.
- Subscripts x, y and z denote cartesian components.

The plain-text output file, `electronic_structure.dat` is formatted as

```
Energy (Hartree):
V
Forces (Hartree/Bohr):
f_1x f_1y f_1z
f_2x f_2y f_2z
...
f_nx f_ny f_nz
Hessian (Hartree/Bohr^2):
Atoms: ( 1 1 )
H_11_xx H_11_xy H_11_xz
H_11_yx H_11_yy H_11_yz
H_11_zx H_11_zy H_11_zz

Atoms: ( 2 1 )
H_21_xx H_21_xy H_21_xz
H_21_yx H_21_yy H_21_yz
H_21_zx H_21_zy H_21_zz

...
Atoms: ( n n )
H_nn_xx H_nn_xy H_nn_xz
H_nn_yx H_nn_yy H_nn_yz
H_nn_zx H_nn_zy H_nn_zz
Stress (Hartree/Bohr^3):
sigma_xx sigma_xy sigma_xz
sigma_yx sigma_yy sigma_yz
sigma_zx sigma_zy sigma_zz
```

where:

- V is the energy of the supercell, normalised to energy per supercell.
- f_i is the force on the i 'th atom. The number and order of atom labels must match those in the input file (i.e. be 1 to n).
- H_{ij} is the block of the Hessian matrix corresponding to atoms i and j , i.e. $\frac{\partial}{\partial \mathbf{r}_i} \frac{\partial}{\partial \mathbf{r}_j} V$.
- σ is the stress tensor.
- Subscripts x, y and z denote cartesian components.

All sections but Energy are optional. All values must be given in atomic units.

This file is parsed by the `ElectronicStructureData` class, and the documentation for this class should be consulted for the full specification of this file. Each line of the file is split into tokens by whitespace, so the exact whitespace on each line does not matter as long as there is some whitespace between each token.

Interfaces to Other Electronic Structure Interfaces

Rather than interfacing with an electronic structure code directly, *Caesar* can instead be interfaced with an external package which in turn interfaces with the electronic structure code. *Caesar* has interfaces to two such packages: *QUIP* and *The Atomic Simulation Environment (ASE)*.

The *QUIP* interface needs to be linked at compile time, as detailed in the *Caesar* `README.txt` file. This is achieved by setting the *CMake* configuration option `LINK_TO_QUIP` to true.

The *ASE* interface uses a *python* script and does not require additional compilation. An example script is provided as `doc/input_files/example_ase_run_script.py` in the *Caesar* repository, and this script is intended to serve as a template for an interface with any of the electronic structure codes which *ASE* can interface with.

Source Code

The source code for the *Caesar* electronic structure interface is available from the `src/common/electronic_structure` directory of the *Caesar* repository

Software Technical Information

Name *Caesar*

Language Fortran

Licence GNU Lesser General Public License version 3

Documentation Tool Ford

Application Documentation See the *Caesar* repository

Software Module Developed by Mark Johnson

Caesar Harmonic Calculation Library

- *Purpose of Module*
- *Theory*
 - *The Harmonic Approximation*
 - *Calculating the Hessian Matrix*
 - *Supercells*
- *Performing Calculations*
- *Source Code*

Purpose of Module

The *Caesar* harmonic calculation library aims to provide an efficient method for calculating vibrational properties under the harmonic approximation [Hoja_ea1]. This can be done using a range of electronic structure codes, using the *Caesar electronic structure interface*.

Theory

The Harmonic Approximation

The nuclear potential energy surface V is a function of the $3n$ -dimensional nuclear coordinate \mathbf{r} . Under the harmonic approximation [Hoja_ea1], this function is approximated as quadratic in the difference between \mathbf{r} and the value of \mathbf{r} for the undisplaced structure, $\mathbf{r}^{(0)}$. Formally, this is

$$V(\mathbf{r}) = (\mathbf{r} - \mathbf{r}^{(0)}) \cdot \mathbf{H} \cdot (\mathbf{r} - \mathbf{r}^{(0)})$$

where \mathbf{H} is the Hessian matrix.

By making a coordinate transform from $\mathbf{r} = \sum_i r_i \hat{\mathbf{r}}_i$ to $\mathbf{r} = \sum_j u_j \hat{\mathbf{u}}_j$, the Hessian matrix can be diagonalised, to give

$$V(\mathbf{u}) = \sum_j \frac{1}{2} N \omega_j^2 u_j^2$$

where each u_j is a normal mode of the system, each ω_j is the corresponding frequency of that mode, and N is the size of the supercell (defined as the ratio of the volume of the supercell to the volume of the primitive cell).

Provided that every value of ω_j is real, the simple form of V means that the Hamiltonian can be diagonalised analytically, and so the free energy and other properties can be calculated analytically [Hoja_ea1].

Calculating the Hessian Matrix

Caesar calculates the Hessian matrix using a finite difference method. This involves calculating the forces on the atoms in atomic configurations where the atomic displacement $\mathbf{r} - \mathbf{r}^{(0)}$ is small. Forces must be calculated by an electronic structure code, via the *Caesar electronic structure interface*.

Caesar minimises the total computational cost of these electronic structure calculations by exploiting crystal symmetry, as calculated by the *spglib* crystal symmetry library, and by using the non-diagonal supercell method [Lloyd-Williams_Monserrat].

Supercells

Most vibrations in crystals cause atoms to move in ways which break translational symmetry. This means that it is not sufficient to calculate vibrational properties in the primitive unit cell alone. Instead, properties must be calculated in a supercell containing multiple copies of the primitive unit cell. Properties should be calculated in a number of different supercells, and the size of the supercell should be increased until the results of the calculations converge.

Performing Calculations

Running the *Caesar Harmonic Calculation Library* is a four-stage process.

- Firstly, `caesar_setup_harmonic` parses the input data, calls *spglib* to calculate the crystal symmetries, and generates a directory structure containing directories in which all the necessary electronic structure calculations must be run.
- Secondly, `caesar_run_harmonic` performs the electronic structure calculations, using the *Caesar electronic structure interface*. There is no connection between the separate electronic structure calculations, so they can be run sequentially, in parallel, or across multiple computers as desired.
- Thirdly, `caesar_calculate_normal_modes` uses the results of the electronic structure calculations to calculate the Hessian matrix and the normal modes of the crystal.

- Finally, `caesar calculate_harmonic_observables` calculates the vibrational properties of the crystal under the harmonic approximation.

The stages are separated so that the potentially computationally costly `run_harmonic` can be run on a separate computer or computers as needed, and so that `calculate_harmonic_observables` can be run repeatedly to calculate different observables as required.

The calculated properties are written to a `harmonic_observables` directory. These can be visualised using the various `caesar plot_utilities`.

Each stage of the calculation has its own helptext, which can be accessed through the `caesar` executable by calling `caesar --help`.

Source Code

The source code for the Caesar harmonic library is available from the `src/harmonic` directory of the [Caesar repository](#)

Software Technical Information

Name Caesar

Language Fortran

Licence GNU Lesser General Public License version 3

Documentation Tool Ford

Application Documentation See the [Caesar repository](#)

Software Module Developed by Mark Johnson

Caesar Anharmonic Calculation Library

- *Purpose of Module*
- *Theory*
 - *Fitting the Potential Energy Surface*
 - *The Vibrational Self-Consistent Harmonic Approximation*
 - *Vibrational Self-consistent Field Theory*
- *Performing Calculations*
- *Source Code*

Purpose of Module

The *Caesar* anharmonic calculation library aims to provide an efficient method for calculating vibrational properties beyond the harmonic approximation; under the vibrational self-consistent harmonic approximation (VSCHA) [Errea_ea1] or using vibrational self-consistent field theory (VSCF) [Christiansen1].

Theory

Fitting the Potential Energy Surface

Caesar models the nuclear potential energy surface (PES) using a truncated Taylor expansion in normal-mode coordinates. Constructing and fitting this model happens over several steps:

- Firstly, a set of symmetry-invariant basis functions are generated, using the crystal symmetries as calculated by [spglib](#).
- Secondly, a set of nuclear coordinates \mathbf{r} are generated at which the PES will be sampled.
- Thirdly, electronic structure calculations are performed at each coordinate, using the *Caesar electronic structure interface*.
- Finally, the results of the electronic structure calculations, including calculated energies, forces and other information, are used to calculate the basis function coefficients.

As with the harmonic calculation, the anharmonic calculation uses the non-diagonal supercell method [Lloyd-Williams_Monserrat1] to reduce the total computational cost of the electronic structure calculations where possible.

The Vibrational Self-Consistent Harmonic Approximation

VSCHA approximates the eigenstates of the system as those which diagonalise an effective harmonic potential $V^{\text{effective}}$. The effective harmonic potential $V^{\text{effective}}$ implemented by *Caesar* has the same functional form and normal modes $\{u_j\}$ as the harmonic potential V^{harmonic} , but has a different set of frequencies $\{\omega_j\}$.

The frequencies $\{\omega_j\}$ are calculated as those which minimise the free energy of the anharmonic PES with respect to the VSCHA eigenstates [Errea_ea1].

Vibrational Self-consistent Field Theory

Traditional VSCF separates the PES $V(\mathbf{u})$ into a sum of single-mode effective potentials $\{V_j(u_j)\}$, each of which is the expectation of V with respect to all modes other than u_j . The Hamiltonian corresponding to each mode is then constructed in the VSCHA eigenbasis, and this is diagonalised to give the single-mode VSCF eigenstates $\{|\psi_{jk}\rangle\}$. This process can be written as two equations,

$$(T + V_j)|\psi_{jk}\rangle = E_{jk}|\psi_{jk}\rangle$$

and

$$V_j = \langle V \rangle_{j' \neq j}$$

These equations are solved self-consistently, using a Pulay scheme [Pulay].

The VSCF method implemented by *Caesar* differs from traditional VSCF methods in that rather than separating the PES single-mode potentials, the PES is instead separated into single-subspace potentials, where each subspace contains a complete set of modes whose frequencies are degenerate as a result of symmetry. This implementation of VSCF is symmetry invariant, unlike the single-mode methods.

Performing Calculations

Prior to performing anharmonic calculations, a harmonic calculation must be performed. This can be done using the *Caesar Harmonic Calculation Library*, or the Hessian matrix of the undisplaced structure can be read using the *Caesar electronic structure interface*.

Like running the *Caesar Harmonic Calculation Library*, running the *Caesar Anharmonic Calculation Library* is a four-stage process.

- Firstly, `caesar_setup_anharmonic` parses the input data and reads the output of the harmonic calculation. It then generates a directory structure containing directories in which all the necessary electronic structure calculations must be run.
- Secondly, `caesar_run_anharmonic` performs the electronic structure calculations, using the *Caesar electronic structure interface*. There is no connection between the separate electronic structure calculations, so they can be run sequentially, in parallel, or across multiple computers as desired.
- Thirdly, `caesar_calculate_potential` uses the results of the electronic structure calculations to fit the anharmonic potential.
- Finally, `caesar_calculate_anharmonic_observables` calculates the vibrational properties of the crystal under VSCHA and VSCF.

The calculated properties are written to an `anharmonic_observables` directory. These can be visualised using the various `caesar_plot_` utilities.

As with the harmonic stages, each anharmonic stage has its own helptext, which can be accessed through the `caesar` executable by calling `caesar --help`.

Source Code

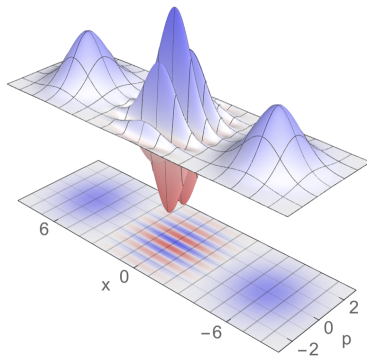
The source code for Caesar anharmonic library is available from the `src/anharmonic` directory of the *Caesar repository*

General Information

Contents

- *Quantum Dynamics Modules*
 - *Introduction*
 - *Objectives of E-CAM WP3 Quantum Dynamics*
 - *Pilot Projects*
 - *Extended Software Development Workshops*
 - *List of available Modules*
- *How to contribute?*
- search

3.1 Introduction



This is a collection of the modules that have been created by the E-CAM community within the area of Quantum Dynamics. This documentation is created using ReStructured Text and the git repository for the documentation. Source files can be found at <https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library> which are open to contributions from E-CAM members.

In the context of E-CAM, the definition of a software module is any piece of software that could be of use to the E-CAM community and that encapsulates some additional functionality, enhanced performance or improved usability for people performing computational simulations in the domain areas of interest to the project.

This definition is deliberately broader than the traditional concept of a module as defined in the semantics of most high-level programming languages and is intended to capture internal workflow scripts, analysis tools and test suites as well as traditional subroutines and functions. Because such E-CAM modules will form a heterogeneous collection we prefer to refer to this as an E-CAM software repository rather than a library (since the word library carries a particular meaning in the programming world). The modules do however share with the traditional computer science definition the concept of hiding the internal workings of a module behind simple and well-defined interfaces. It is probable that in many cases the modules will result from the abstraction and refactoring of useful ideas from existing codes rather than being written entirely de novo.

Perhaps more important than exactly what a module is, is how it is written and used. A final E-CAM module adheres to current best-practice programming style conventions, is well documented and comes with either regression or unit tests (and any necessary associated data). E-CAM modules should be written in such a way that they can potentially take advantage of anticipated hardware developments in the near future (this is one of the training objectives of E-CAM).

3.2 Objectives of E-CAM WP3 Quantum Dynamics

Software development in quantum dynamics has so far been less systematic than in other fields of modelling, such as classical molecular dynamics or electronic structure. Although some packages have been developed to implement specific methods, e.g. [Quantics](#) for wave packet dynamics, or subroutines added to electronic structure packages, e.g. Surface Hopping and Ehrenfest in [CPMD](#), these efforts are not the standard.

One of the goals of E-CAM's WP3 is then to provide an environment to stimulate the transition from in-house codes, often developed and used by single groups, to the development of modular, well documented community-based software packages capable of multiple functionalities and adopting a common set of standards and benchmarks.

To foster this development, we have initiated five parallel activities:

- Creating software for benchmarking and testing based on exact integration schemes for low dimensional systems and standard potentials.
- Creating an environment to transform in-house software to modules that adhere to the E-CAM best practices.
- Disseminating this initiative to attract coding efforts from leading groups in the field to the E-CAM repository.
- Interact with industrial partners to enrich our repository with software targeted at their needs.
- Training young code developers.

3.3 Pilot Projects

One of primary activity of E-CAM is to engage with pilot projects with industrial partners. These projects are conceived together with the partner and typically are to facilitate or improve the scope of computational simulation within the partner. The related code development for the pilot projects are open source (where the licence of the underlying software allows this) and are described in the modules associated with the pilot projects.

The [pilot project](#) of the WP3 in collaboration with [IBM](#) is related to quantum computing and improvements of the quantum computer technology. One of our main topic was development of software for construction of control pulses necessary for operating quantum logical gates between qubits in a universal quantum computer using the Local Control Theory. [[Cure](#)] More information can be found on the [pilot project](#) web site. Below are listed the pilot project modules created so far:

3.3.1 LocConQubit

Software Technical Information**Language** Python 3.5**License** MIT license (MIT)**Documentation Tool** sphinx**Software Module Developed by** Momir Mališ

- *Purpose of Module*
- *Local Control Theory (LCT)*
- *Applications of the Module*
- *Installation*
- *Testing*
- *Source Code*
- *Source Code Documentation*
- *References*

Purpose of Module

LocConQubit is a code for constructing controlled pulses on isolated qubit systems that can either drive the population between specific qubit states or work as a logical gates between qubits. LocConQubit implements the Local Control Theory (LCT) which generates the required pulses on-the-fly. The generated pulses can be further post-processed with a variety of tools accompanying the LocConQubit module in order to obtain an optimal control pulse.

Local Control Theory (LCT)

In general Local Control Theory is an on-the-fly procedure for updating the time-dependent Hamiltonian ($\hat{H}(t)$) to achieve population transfer from some initial quantum state to a designated quantum target state ($|\psi\rangle$). [LCT1] [LCT2] LCT achieves its full capacity if the time-dependent component of the full system Hamiltonian $\hat{H}(t)$ can be decomposed as an external perturbation ($V'(t) \times \hat{H}'$) acting on a system with a time-independent Hamiltonian (\hat{H}_0),

$$\hat{H}(t) = \hat{H}_0 + V'(t) \times \hat{H}',$$

and if the targeted quantum state $|\psi\rangle$ is an eigenstate of the same time-independent Hamiltonian \hat{H}_0 ,

$$\hat{H}_0|\psi\rangle = \epsilon|\psi\rangle.$$

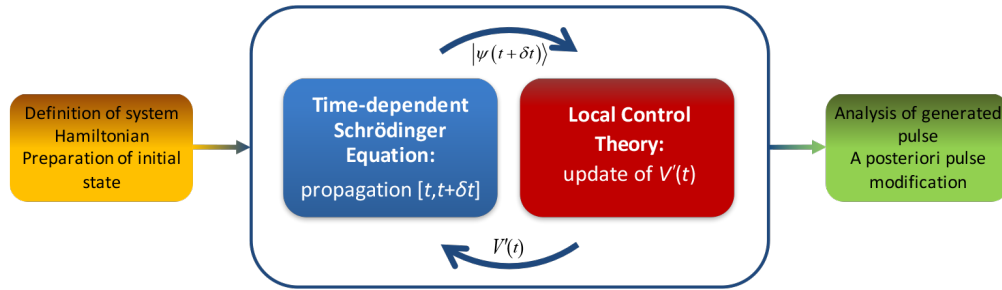
LCT in this case constructs a time-dependent perturbation $V'(t)$ from the expression

$$V'(t) = -i\langle\Psi(t)|[\hat{H}', |\psi\rangle\langle\psi|]|\Psi(t)\rangle,$$

which will achieve the required population transfer from any initial state to the $|\psi\rangle$ target state. The new time-dependent potential component $V'(t)$ updates the time-dependent component of the full Hamiltonian describing the evolution of the system via the time-dependent Schrödinger equation

$$i\frac{\partial}{\partial t}|\Psi(t)\rangle = \hat{H}(t)|\Psi(t)\rangle.$$

The LCT procedure is applied sequentially in small integration steps $[t, t+\delta t]$ within the propagation of above equation until the population has been completely transferred to the designated target state. The schematic below illustrates the LCT procedure.



Applications of the Module

Application of the LCT module can be found at the [pilot project web page](#).

Installation

The LocConQubit is a Python based code. The module requires the presence of [QuTip](#) (version 4.1 or above) program package and the modules accompanying QuTip (namely: [numpy](#) (version 1.13 or above), [scipy](#) (version 0.18 or above), [matplotlib](#) (version 2.10 or above)). A Python interpreter 3.5 or above is required, because the module has not been used with Python 2 versions. Instructions on how to install the QuTip and the accompanying program packages can be found on this [link](#). Upon the successful installation of QuTip, all other required packages will be present. It is highly recommended to verify the QuTip after its installation. Instructions for QuTip testing are provided on the [installation page](#).

Testing

Proper functionality of LocConQubit module can be verified by performing the unit tests simply by executing the below command in the directory containing all LocConQubit module files

```
python test_LCT.py
```

where *python* is an alias for a Python 3.5 version interpreter or higher. Five unit tests are executed sequentially and all must pass successfully in order to use the LocConQubit module.

Source Code

The LocConQubit module source code is located at: <https://gitlab.e-cam2020.eu:10443/Quantum-Dynamics/QC>.

Source Code Documentation

The source code is accompanied with [sphinx](#) documentation located in sub-directory `./doc`. Instructions for sphinx installation can be found [here](#). The html documentation files can be obtained by executing the following command in the `./doc` sub-directory

```
cd ./doc
make html
```

The generated documentation is located in the `./doc/_build/html/index.html`.

References

3.3.2 PerGauss: Periodic Boundary Conditions for gaussian bases

Software Technical Information

Language Fortran 90

Licence None

Documentation Tool Documentation provided as in-line comments within the source code

Application Documentation Quantics documentation can be found in [quantics](#)

Relevant Training Material Tutorial and exercises to test the code are available [here](#)

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

Purpose of Module

The module PerGauss (**Per**iodic **Gauss**ians) consists on an implementation of periodic boundary conditions for gaussian bases for the [Quantics](#) program package.

In quantum dynamics, the choice of coordinates is crucial to obtain meaningful results. While xyz or normal mode coordinates are linear and do not need a periodical treatment, particular angles, such as dihedrals, must be included to describe accurately the (photo-)chemistry of the system under consideration. In these cases, periodicity can be taken into account, since the value of the wave function and hamiltonian repeats itself after certain intervals.

This feature is already implemented for grid basis functions such as exponential-DVR and FFT to use in the framework of the MCTDH method, within the quantics package. Using as wave function ansatz a linear combination of gaussians, following the original idea of Heller, has enormous advantages: First, a gaussian that follows a classical trajectory is the exact solution of the quantum harmonic oscillator and harmonic oscillators are generally the first step into approximating potential energy surfaces. This also allows a smooth transition to dynamics methods based on classical trajectories such as Ab-Initio Multiple Spawning (AIMS) and Surface Hopping. Second, one can easily take advantage of the locality of gaussians and move towards on-the-fly methods, where the potential is calculated as the basis functions span the conformational space.

In the case of methods that use gaussian basis functions, such as G-MCTDH¹, vMCG² and its on-the-fly version DD-vMCG within the quantics set of programs, no implementation of periodic boundary conditions has been made until this contribution.

The module is expected to provide the quantum dynamics community with a more efficient way of treating large systems whose excited state driving forces involve periodic coordinates. When used on precomputed potentials (in G-MCTDH and vMCG), the model can improve the convergence since smaller grid sizes are needed. Used on-the-fly,

¹ I. Burghardt, I. H.-D. Meyer, and L. S. Cederbaum *J. Chem. Phys.* **115** (1999) 2927

² G. W. Richings, I. Polyak, K. E. Spinlove, G. A. Worth, I. Burghardt, B. Lasorne *Int. Rev. Phys. Chem.* **34** (2015) 269

it reduces considerably the amount of electronic structure computations needed compared to cartesian coordinates, since conformations that seemed far in the spanned space may be closer after applying a periodic transformation.

Background Information

Currently pergauss resides within the [Quantics](#) software package available upon request through [gitlab](#).

Testing

A test example (`pergauss.inp`) is provided to test the module and can be found in the directory `$quantics_path/inputs`, where `quantics_path` is where [Quantics](#) is located. The test can be done through the following command

```
$ quantics -mnd pergauss.inp
```

A more detailed test documentation for [Quantics](#) code developers can be found in [this link](#)

Source Code

The source code for pergauss can be found within the [Quantics](#) software which can be downloaded via [gitlab](#). The [Quantics](#) project has a private repository so you also need to be a member of the project to checkout. Then type into terminal

```
$ git clone https://gitlab.com/quantics/quantics.git DIRECTORY
```

Within the [Quantics](#) program, the explicit code is located at the source code folder in files `mctdhlib/gwplib.f90`, `geninwf/eininwfmod.f90`, `geninwf/genphil.f90`, `gendvr/einpbasmod.f90` and `include/global.f90`. Every modified line will be preceded by a comment saying `!pergauss` to help users finding the modifications.

References

LocConQubit is a code for the construction of controlled pulses on isolated qubit systems using the Local Control Theory.

3.3.3 OpenQubit

Software Technical Information

Language Python 3.5

License MIT license (MIT)

Documentation Tool sphinx

Software Module Developed by Momir Mališ

- *Purpose of Module*

- *Applications of the Module*
- *Installation*
- *Testing*
- *Source Code*
- *Source Code Documentation*

Purpose of Module

OpenQubit is a patch to the *LocConQubit* module which extends the capabilities of the latter module with functionalities to generate control pulses in a more realistic systems with dissipating effects. The module incorporates the Lindblad master equation into the system propagator upon which the Local Control Theory generates a control pulse. For more information on LocConQubit module and Local Control Theory see *LocConQubit*.

Applications of the Module

Application of the OpenQubit module can be found at this [link](#).

Installation

Before applying the patch LocConQubit code has to be installed and tested. For the installation and testing of LocConQubit code see the corresponding *documentation*. *Git* has to be also installed. The OpenQubit patch should be downloaded from the [repository](#) and made available to insert it into the directory containing the LocConQubit module. In the directory containing the branch with the LocConQubit module the installation of the OpenQubit is performed by applying the OpenQubit patch. It is advised to make a new branch from the master branch first. The installation should be made by following these instructions:

```
(Check that you are on the QC master branch,
 e.g. command 'git status' should display master in output)

git checkout -b OpenLCTCode

(Download the OpenQubit.patch file here directly or
 copy it from a directory containing the previous download:)
cp [Directory containing the OpenQubit.patch file]/OpenQubit.patch .

git apply OpenQubit.patch
```

Special care should be taken when patching the `test_5.pkl` binary file. If the above operation fails due to problems with patching a binary file, the file can be separately downloaded from the patch source code [webpage](#) and inserted into the `reference_data` subdirectory.

Testing

The application of the OpenQubit patch should be verified by executing the LocConQubit module standard test, which is performed by executing the command below in the same directory containing all of the OpenQubit module files

```
python test_LCT.py
```

where *python* is an alias for a Python 3.5 version interpreter or higher. The test executes five LocConQubit standard test and an additional OpenQubit test (`test_5.pkl`). Unit tests are sequentially executed and all must pass successfully in order to use the OpenQubit module.

Source Code

The OpenQubit patch is located at: <https://gitlab.e-cam2020.eu:10443/Quantum-Dynamics/QC/tree/OpenQubit>. This same link contains the `test_5.pkl` binary file for download.

Source Code Documentation

The source code is accompanied with `sphinx` documentation located in sub-directory `./doc`. Instruction for `sphinx` installation can be found [here](#). The html documentation files can be obtained by executing the following command in the `./doc` sub-directory

```
cd ./doc
make html
```

The generated documentation is located in the `./doc/_build/html/index.html`.

OpenQubit is an extension to the LocConQubit code for the construction of controlled pulses in a more realistic environment with dissipating effects.

3.4 Extended Software Development Workshops

3.4.1 ESDW Maison de la Simulation (Paris 2016)

The first Quantum Dynamics ESDW was held in June-July 2016 at the [Maison de la Simulation](#) near Paris. 10 students and 6 tutors, including Dr. Ivano Tavernelli representing the industrial partner of the WP3, [IBM](#), worked to develop software modules in the following areas:

- Exact quantum propagation methods for low dimensional systems to be used to provide benchmarks for approximate schemes
- Development of a library of single and multi surface potentials for benchmark systems
- Calculation of approximate quantum time correlation functions

Work was performed by teams of 2-4 students, assisted by the senior participants and by E-CAM's Software Manager, Dr. Alan O'Cais, and the Software Developer associated to WP3, Dr. Liang Liang.

In addition to the software development activities, the Workshop enjoyed lively scientific discussions centered on presentations made by the students and the senior participants. The on-line E-CAM tools for software development, including the Git repository, and tools for the documentation (Doxygen) and performance analysis were presented by E-CAM staff members and participants were instructed on their use via tutorials. The program was further enriched by the interactions with experts on software and hardware development working at Maison de la Simulation who gave talks on topics such as architectures and programming paradigms and the use of advanced visualization tools such as the Image wall hosted by the Maison de la Simulation.

3.4.2 ESDW University College Dublin (2017)

The second Quantum Dynamics ESDW was held in July 2017 (first part) and March 2018 (wrap up meeting) at [University College Dublin](#). 21 participants, including the representative of WP3's current industrial partner [IBM](#), worked to develop and upload on the E-CAM repositories software modules in the following areas:

- Calculation of approximate quantum time correlation functions via the PaPIM code;
- Mixed quantum-classical algorithms, with specific reference to Surface Hopping and Wigner-Liouville methods;
- Implementation of the factorization scheme for quantum dynamics in [CPMD](#);
- Interfacing of quantum codes with electronic structure codes;
- Grid based exact propagation schemes;
- Design and optimization of qubit control pulses.

Teams of coders assisted by senior tutors, E-CAM's Software Manager, Dr. Alan O'Cais, and WP3 Software Developer, Dr. Liang Liang, performed the work. Specific discussions on optimal parallelization strategies for the E-CAM's quantum dynamical codes (PaPIM and Quantics) were also initiated and implemented. The coding work was accompanied by scientific presentations on the themes of the workshops and by the instruction from E-CAM personnel on the CoE's tools for software production, testing, documentation and maintaining. The participants benefitted also from the proximity of software and hardware experts from the [ICHEC](#) supercomputing center that offered, in particular, a set of lectures and tutorials on OpenMP parallelization.

Modules developed in this workshop not included in other subheadings are:

Trotter Based Quantum Classical Surface Hopping Propagator - correlated sampling

Software Technical Information

Language C++ (C++11 or higher)

Licence MIT licence (MIT)

Documentation Tool Doxygen

Application Documentation [Documentation](#)

Software Module Developed by Sean Kelly, Athina Lange, Shrinath Kumar and Donal MacKernan

- *Abstract*
- *Purpose of Module*
- *Background Information*
- *Applications*
- *Algorithms and Software Implementation*
- *Compiling*
- *Checking for accuracy*
- *Testing, Performance and Scaling*
- *Source Code*

- [Source Code Documentation](#)
- [References](#)

Abstract

The present module is a highly refactored version of a code based on a highly cited algorithm published by D. Mackeran, G.Ciccotti and R. Kapral [Mackernan1]. The module software has been entirely refactored in modern C++ (GNU 2011 or higher) so as to: (a) run with high-efficiency on massively parallel platforms under OpenMP or MPI; and (b) be at the core of additional software modules aimed at addressing important issues such as improving the speed of convergence of estimates using correlated sampling, and much more realistic treatment of the classical bath, and connecting to other problems such as constant pH simulation through an effective Hamiltonian.

Purpose of Module

Quantum rate processes in condensed phase systems are often computed by combining quantum and classical descriptions of the dynamics including non-adiabatic coupling, using propagators which amount to quantum path integrals in a partial Wigner phase space representation, such as the mixed quantum-classical Dyson equation and variants thereof, or the Trotter decomposition of the quantum-classical propagator.

Background Information

An understanding of the dynamical properties of condensed phase quantum systems underlie the description of a variety of quantum phenomena in chemical and biological systems. The development of schemes for the efficient and accurate simulation of the quantum dynamics of such systems is an active area of research in chemical physics, and is essential if problems of chemical interest involving complex molecular species in the condensed phase are considered.

In investigations of the dynamical properties of quantum statistical mechanical systems, one is often interested in the average value of some operator when the system evolves from a given initially prepared distribution described by the density matrix $\hat{\rho}(0)$. In such cases the quantum mechanical average value of an operator \hat{B} is given by $\overline{B(t)} = \text{Tr} \hat{B} \hat{\rho}(t) = \text{Tr} \hat{B}(t) \hat{\rho}(0)$. Here, $\hat{B}(t)$ evolves in time through the Heisenberg equation of motion. In many applications, it is useful to partition the system into a subsystem and a bath. A phase space description of the bath can be obtained by taking a partial Wigner transform over the bath coordinate $\{Q\}$ representation of the full quantum system. In this partial Wigner representation the expectation value of $\hat{B}(t)$ takes the

$$\overline{B(t)} = \text{Tr}' \int dR dP B_W(R, P, t) \rho_W(R, P)$$

where the prime on the trace indicates a trace over the subsystem degrees of freedom.

The software module developed here is based on a Trotter-based scheme for simulating quantum-classical Liouville dynamics in terms of an ensemble of surface-hopping trajectories. The method can be used to compute the dynamics for longer times with fewer trajectories than the sequential short-time propagation (SSTP) algorithm, which is also based on surface-hopping trajectories. This module builds on the single path version of the trotter-based sampling scheme [Mackernan1] but introduces a key improvement of correlated sampling to drastically reduce the variance associated with sampling. It also builds on the C++ formulation of the single path code. For mathematical details, we refer the reader to eq.30-35 of the paper.

Applications

The applications of quantum surface hopping include, among others, non-adiabatic chemical rate processes involving electronic, vibrational or other degrees of freedom, decoherence in open quantum systems and quantum transport processes. Decoherence due to coupling with the environment is a fundamental difficulty in the development of quantum computing. The ability to predict, control and reduce decoherence requires an adequate description of the associated non-adiabatic processes taking place. Quantum effects and frequently non-adiabaticity also underlie the study of ultra-fast rate processes in solution.

Algorithms and Software Implementation

The current Single Path code has three main advantages over the original version. First it is separated into files based on function for better readability. For example the 'transition_matrix.cpp' file is where the transition matrix and associated functions are defined, etc. Secondly input parameters are read from an Input file, so the code no longer needs to be recompiled to adjust these parameters. And finally the code has been altered to run in parallel which allows for a significant reduction in runtime.

Compiling

All current versions of this code use the [GNU scientific library](#) version 2.5 for random number generation.

OpenMP version:

With the GNU compiler, gcc version 6.3.0 or greater is required.

On the Kay cluster this can be done as follows:

```
# Load modules to give us the right environment
module load gcc/8.2.0
module load gsl/gcc/2.5
```

```
# Compile command;
g++ -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_
↪matrix.cpp opt_parser.cpp -lgsl -lgslcblas -lm -fopenmp -std=c++11

# Run command:
OMP_NUM_THREADS=[number of OpenMP threads] ./run Input
```

With the Intel compiler:

```
# Compile command;
icpc -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_
↪matrix.cpp opt_parser.cpp -lgsl -lgslcblas -lm -qopenmp -std=c++11

# Run command:
OMP_NUM_THREADS=[number of OpenMP threads] ./run Input
```

MPI version:

```
# Load modules to give us the right environment
module load intel/2018u4
module load gsl/intel/2.5
module load gcc/8.2.0
```

```
# Compile command;
mpic++ -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_
↪matrix.cpp opt_parser.cpp -lgsl -lgslcblas -lm -std=c++11

# Run command:
mpirun -n [number of MPI processors] ./run Input
```

Note: A frequent error encountered while compiling is: “fatal error: gsl/gsl_rng.h: No such file or directory”

This can occur if the directory is not installed on the standard search path of the compiler. It can be fixed by adding it's location as a flag in the compile command as explained in this link: [Using the GSL Library](#).

On Kay the flags ‘-I/ichec/packages/gsl/gcc/2.5/include’ and ‘-L/ichec/packages/gsl/gcc/2.5/lib’ must be added to the compile command as:

```
g++ -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_matrix.cpp_
↪opt_parser.cpp -lgsl -lgslcblas -lm -fopenmp -std=c++11 -I/ichec/packages/gsl/gcc/2.
↪5/include -L/ichec/packages/gsl/gcc/2.5/lib
```

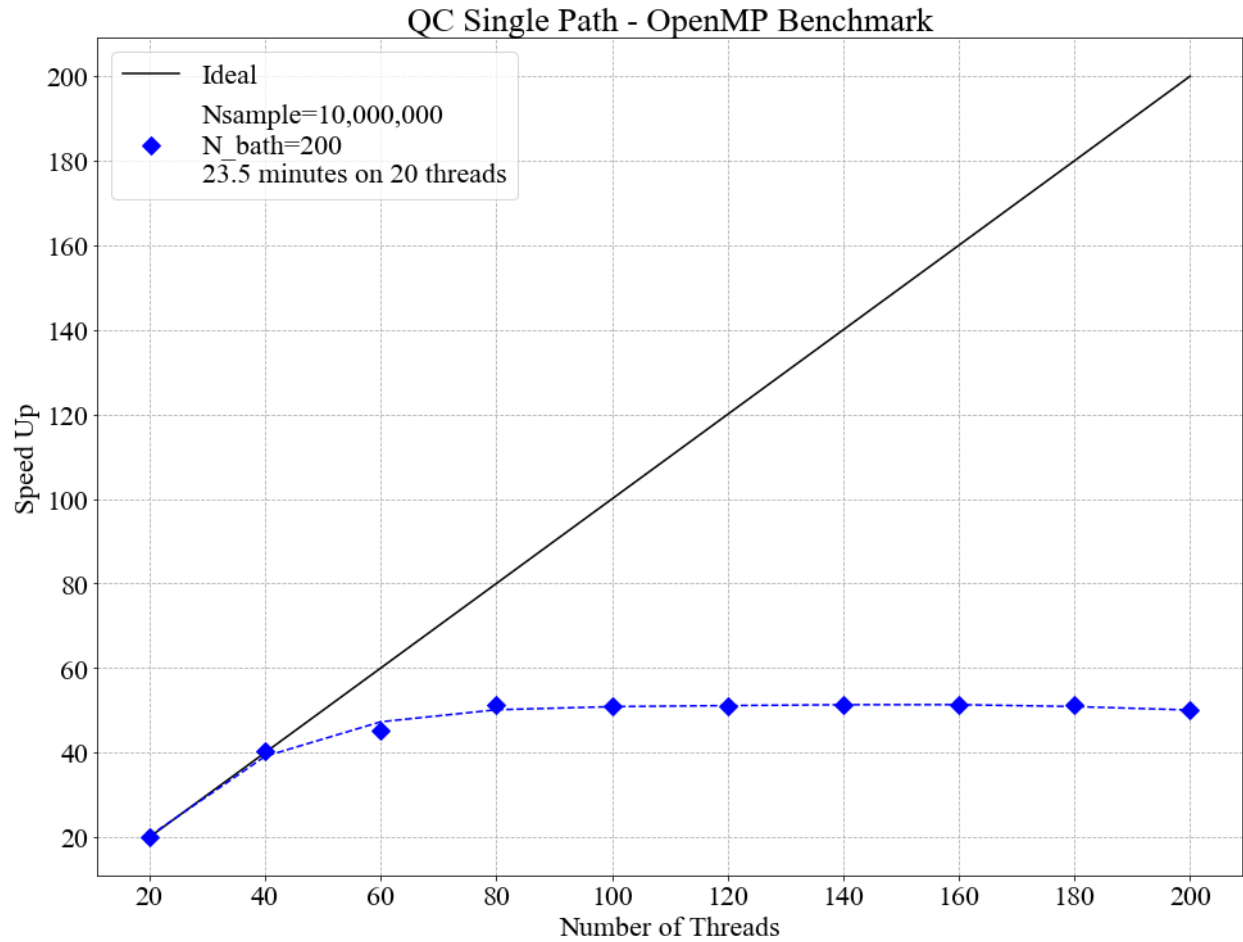
Checking for accuracy

The original serial code was run 1000 times to generate an expected output and variance. These can be found in the ./Regression_testing sub-directory. A regression test is built into both the OpenMP and MPI versions which checks if their output is within five standard deviations of the expected output (given a specific set of input parameters). If any part of the output goes outside that limit the regression test will fail. (Note: To run a test ‘Regression_test=1’ must be set in the Input file along with a standard set of parameters. All of this is specified in the Input file).

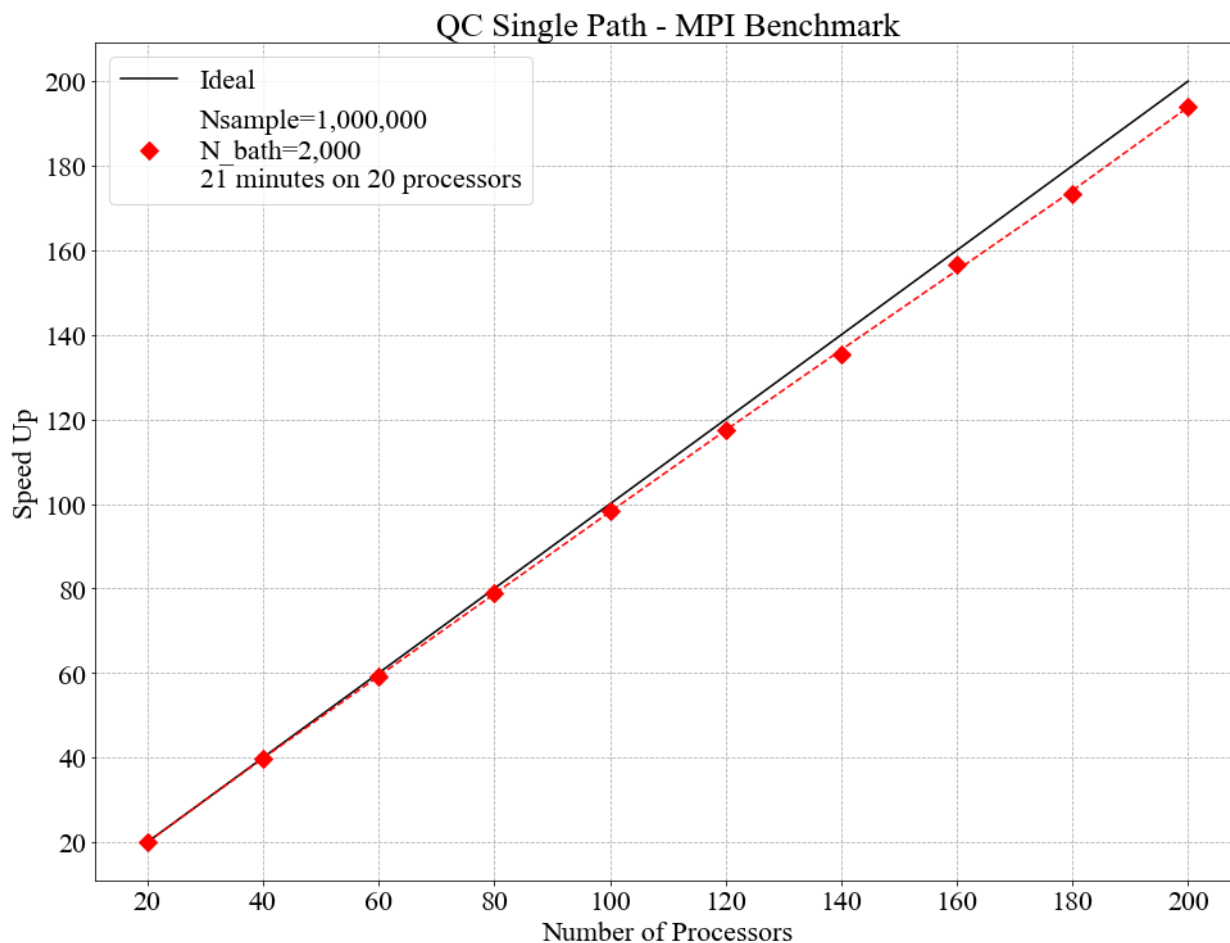
Testing, Performance and Scaling

Testing was performed on the Kay supercomputer from ICHEC. Kay is separated into nodes, each of which has 2 x (20 core) sockets. To test the parallel efficiency of both the OpenMP and MPI versions of the code they were benchmarked on 20 - 200 cores (1 - 5 nodes).

The OpenMP version was run for 10,000,000 samples (Nsample = 10,000,000) and for a bath size of 200 (N_bath = 200). As can be seen in the graph below OpenMP scales perfectly on a single node (i.e. less than 40 cores), but provides little to no benefit over multiple nodes.



The MPI version was run for 1,000,000 samples ($N_{\text{sample}} = 1,000,000$) and for a bath size of 2,000 ($N_{\text{bath}} = 2,000$). MPI scales very well over the entire benchmark (up to 200 cores), with an average efficiency of 96.3%.



Source Code

The source codes for the OpenMP and MPI versions of the code are:

[Surface Hopping - OpenMP version](#)

[Surface Hopping - MPI version](#)

Source Code Documentation

The source code documentation is given at <https://gitlab.e-cam2020.eu/Quantum-Dynamics/Surface-Hopping/tree/master/Doc>. These documentation files can be updated by executing the `make` command in the `Doc` directory.

References

3.4.3 ESDW Durham University (Durham 2019)

This modules have been developed at Durham ESDW

3.5 List of available Modules

Below are listed all the modules from the E-CAM ESDWs in Quantum Dynamic developed up-to-date:

3.5.1 CTMQC

Software Technical Information

Language Fortran 90

License GNU Lesser General Public License (LGPL)

Documentation Tool doxygen

Software Module Developed by Federica Agostini, Seung Kyu Min, Ivano Tavernelli, Graeme H. Gossel

- *Purpose of Module*
- *Coupled-Trajectory Mixed Quantum-Classical Dynamics*
- *Applications of the Module*
- *Installation*
- *Testing*
- *Source Code*
- *References*

Purpose of Module

CTMQC is a module for excited-state nonadiabatic dynamics, therefore it is used to simulate the coupled dynamics of electrons and nuclei (ideally in gas phase molecular systems) in response to, for instance, an initial electronic excitation.

The purpose of the module is to familiarize the user with a new simulation technique, i.e., the CTMQC method, for treating problems where electronic excited states are populated during the molecular dynamics. Photo-activated ultrafast processes are typical situations in which an approach like CTMQC can be used to predict molecular properties, like structures, quantum yields, or quantum coherence.

As clarified below, the CTMQC module is based on the coupled-trajectory mixed quantum classical algorithm [CTMQC1] [CTMQC2] that has been derived starting from the evolution equations in the framework the exact factorization of the electron-nuclear wavefunction [EF1] [EF3] [EF4]. The CTMQC algorithm belongs to the family of quantum-classical methods, as the time evolution of the nuclear degrees of freedom is treated within the classical approximation, whereas electronic dynamics is treated fully quantum mechanically. Basically, the nuclei evolve as point particles, following classical trajectories, while the electrons *generate* the potential inducing such time evolution.

In its current implementation, the module cannot deal with arbitrary nuclear dimensions, but it is restricted to treat 3-dimensional problems, which gives the possibility to compare quantum-classical results easily and directly with quantum wavepacket dynamics. CTMQC has been analyzed and benchmarked against exact propagation results on typical low-dimensional model systems [CTMQC3], and applied for the simulation of the photo-initiated ring-opening process of Oxirane [CTMQC4]. For this study, CTMQC has been implemented in a developer version of the CPMD

electronic structure package based on time-dependent density functional theory. Concerning electronic input properties, the CTMQC module requires a grid representation of the adiabatic potential energy surfaces and of the nonadiabatic coupling vectors, since the electronic dynamics is represented and solved in the adiabatic basis. This feature allows the algorithm to be easily adaptable, in the current form, to any quantum chemistry electronic structure package. The number of electronic states to be included is not limited, and can be specified as input.

Coupled-Trajectory Mixed Quantum-Classical Dynamics

The *exact factorization of the electron-nuclear wavefunction* [EF1] provides a prescription for decomposing the time-dependent Schrödinger equation for a system of interacting electrons and nuclei into the coupled dynamics of the subsystems, i.e., the electronic and the nuclear. The time-dependent molecular wavefunction, $\Psi(\mathbf{r}, \mathbf{R}, t)$, is the solution of the time-dependent Schrödinger equation $\hat{H}\Psi = i\partial_t\Psi$, with Hamiltonian $\hat{H}(\mathbf{r}, \mathbf{R}) = \hat{T}_n(\mathbf{R}) + \hat{H}_{BO}(\mathbf{r}, \mathbf{R})$, containing the nuclear kinetic energy, \hat{T}_n , and the electronic Born-Oppenheimer Hamiltonian, \hat{H}_{BO} , defined as the sum of the electronic kinetic energy and of the interaction potentials. Here, the symbols \mathbf{r}, \mathbf{R} indicate all electronic and nuclear coordinates, respectively. The full wavefunction can be exactly written as the product

$$\Psi(\mathbf{r}, \mathbf{R}, t) = \chi(\mathbf{R}, t)\Phi_{\mathbf{R}}(\mathbf{r}, t),$$

where $\chi(\mathbf{R}, t)$ can be considered a genuine nuclear wavefunction, yielding the exact nuclear many-body density and current density, and $\Phi_{\mathbf{R}}(\mathbf{r}, t)$, the electronic function, depends parametrically on the nuclear configuration.

Inserting the exact-factorization form of the full wavefunction into the time-dependent Schrödinger equation yields the coupled evolution equations for the two components of the molecular wavefunction, namely

$$\begin{aligned} [\hat{H}_{BO} + \hat{U}_{en}^{coup} - \epsilon] \Phi_{\mathbf{R}}(\mathbf{r}, t) &= i\hbar\partial_t \Phi_{\mathbf{R}}(\mathbf{r}, t) \\ \left[\sum_{\nu=1}^{N_n} \frac{[-i\hbar\nabla_{\nu} + \mathbf{A}_{\nu}]^2}{2M_{\nu}} + \epsilon \right] \chi(\mathbf{R}, t) &= i\hbar\partial_t \chi(\mathbf{R}, t) \end{aligned}$$

where the new quantities introduced will be discussed below. The derivation of these equations can be found in [EF2]. Nuclear masses are indicated by the symbol M_{ν} , with the index ν running over the N_n nuclei. In the electronic equation, the operator $\hat{U}_{en}^{coup}[\Phi_{\mathbf{R}}, \chi]$ couples the electronic evolution to the nuclear dynamics as it depends on the nuclear wavefunction,

$$\hat{U}_{en}^{coup}[\Phi_{\mathbf{R}}, \chi] = \sum_{\nu=1}^{N_n} \frac{1}{M_{\nu}} \left[\frac{[-i\hbar\nabla_{\nu} - \mathbf{A}_{\nu}]^2}{2} + \left(\frac{-i\hbar\nabla_{\nu}\chi}{\chi} + \mathbf{A}_{\nu} \right) \cdot (-i\hbar\nabla_{\nu} - \mathbf{A}_{\nu}) \right].$$

The scalar potential, or time-dependent potential energy surface $\epsilon(\mathbf{R}, t)$, and the time-dependent vector potential $\mathbf{A}_{\nu}(\mathbf{R}, t)$, are defined by

$$\begin{aligned} \epsilon(\mathbf{R}, t) &= \langle \Phi_{\mathbf{R}}(t) | \hat{H}_{BO} + \hat{U}_{en}^{coup} - i\hbar\partial_t | \Phi_{\mathbf{R}}(t) \rangle_{\mathbf{r}} \\ \mathbf{A}_{\nu}(\mathbf{R}, t) &= \langle \Phi_{\mathbf{R}}(t) | -i\hbar\nabla_{\nu} \Phi_{\mathbf{R}}(t) \rangle_{\mathbf{r}}, \end{aligned}$$

respectively, where $\langle \cdot \rangle_{\mathbf{r}}$ stands for an integration over the electronic coordinates. In the nuclear time-dependent Schrödinger equation, the time-dependent potentials fully account for electronic nonadiabatic effects, i.e., excited-state effects, on nuclear motion.

Approximating the nuclear time-dependent Schrödinger equation classically, the force generating the trajectory along which the ν -th nucleus evolve is determined

$$\mathbf{F}_{\nu} = \mathbf{F}_{\nu}^{\text{Eh.}} + \mathbf{F}_{\nu}^{\text{qm.}}$$

In this expression the classical force is decomposed in a – more standard – Ehrenfest-like contribution

$$\mathbf{F}_{\nu}^{\text{Eh.}} = - \sum_k |C_k(t)|^2 \nabla_{\nu} \epsilon_{BO}^{(k)} - \sum_{k,l} C_l^*(t) C_k(t) \left(\epsilon_{BO}^{(k)} - \epsilon_{BO}^{(l)} \right) \mathbf{d}_{lk,\nu}$$

and a new *coupled-trajectory* contribution, depending on the quantum momentum,

$$\mathbf{F}_\nu^{\text{qm}} \sum_k |C_k(t)|^2 \left(\sum_{\nu'=1}^{N_n} \frac{2\mathbf{Q}_{\nu'}}{\hbar M_{\nu'}} \cdot \mathbf{f}_{l,\nu'} \right) \left[\mathbf{f}_{k,\nu} - \sum_l |C_l(t)|^2 \mathbf{f}_{l,\nu} \right].$$

Several new symbols have been introduced in these expressions: $C_k(t)$ represents the k -th coefficient of the expansion of the electronic wavefunction on the adiabatic basis, thus the index k runs over the n states that are included in the expansion; $\epsilon_{BO}^{(k)}$ is the energy eigenvalue of the Hamiltonian \hat{H}_{BO} on the k -th eigenstate; $\mathbf{d}_{lk,\nu}$ stands for the nonadiabatic coupling vector between the electronic adiabatic states l and k and calculated from the displacement of the nucleus ν ; $\mathbf{f}_{k,\nu}(t) = \int^t [-\nabla_\nu \epsilon_{BO}^{(k)}(\mathbf{R}^{cl}(t'))] dt'$ is the adiabatic force integrated over time along the trajectory (indicated here as the multi-dimensional vector $\mathbf{R}^{cl}(t)$); $\mathbf{Q}_\nu(t)$ is the *quantum momentum*, whose expression will be given below. It is worth underlying at this point that all quantities depending on nuclear positions, such as the adiabatic energies or the nonadiabatic coupling vectors, become, in the quantum-classical picture, functions of the trajectory.

Expressing the electronic evolution equation in the adiabatic basis (formed by the set of eigenstates of the Born-Oppenheimer Hamiltonian \hat{H}_{BO}), one gets a set of n coupled evolution equations for the coefficients $C_k(t)$ of such expansion, namely

$$\dot{C}_k(t) = \dot{C}_k^{\text{Eh.}}(t) + \dot{C}_k^{\text{qm}}(t)$$

where, once again, the first term is a standard Ehrenfest-like contribution

$$\dot{C}_k^{\text{Eh.}}(t) = -\frac{i}{\hbar} \epsilon_{BO}^{(k)} C_k(t) - \sum_{\nu=1}^{N_n} \dot{\mathbf{R}}_\nu^{cl}(t) \cdot \sum_l \mathbf{d}_{kl,\nu} C_l(t),$$

whereas the second term is a *coupled-trajectory* contribution, depending on the quantum momentum,

$$\dot{C}_k^{\text{qm}}(t) = \sum_{\nu=1}^{N_n} \frac{\mathbf{Q}_\nu}{\hbar M_\nu} \cdot \left[\mathbf{f}_{k,\nu} - \sum_l |C_l(t)|^2 \mathbf{f}_{l,\nu} \right] C_k(t).$$

The quantum momentum is a function of nuclear positions, thus as consequence of the classical treatment of the nuclei, it becomes a function of the trajectory, namely

$$\mathbf{Q}_\nu(\mathbf{R}^{cl}(t), t) = -\frac{\hbar}{2} \frac{\nabla_\nu |\chi(\mathbf{R}^{cl}(t), t)|^2}{|\chi(\mathbf{R}^{cl}(t), t)|^2}.$$

Notice that the quantum momentum tracks the spatial variation of the nuclear density, as it contains its spatial derivative. At each time step, the nuclear density has to be reconstructed, for instance by computing a histogram from the distribution of classical trajectories. Such calculation requires that at the end of each step of dynamics, the trajectories *communicate* – all at the same time – information about their positions, in order to compute the quantum momentum. Once $\mathbf{Q}_\nu(\mathbf{R}^{cl}(t), t)$ is known, the trajectories can perform a new step of dynamics. On-the-fly calculation of the quantum momentum is possible only if the trajectories are propagated all at the same time, that is why the underlying algorithm has been dubbed “coupled-trajectory”-MQC.

Applications of the Module

The module is designed to apply the CTMQC procedure to one-, two-, and three-dimensional model systems where an arbitrary number of electronic states are coupled via the nuclear dynamics. Tully model systems [Tully] are within the class of problems that can be treated by the module, as well as a wide class of multidimensional problems involving, for instance, ultrafast radiationless relaxation of photo-excited molecules [CI1] through conical intersections.

Installation

The CTMQC is a fortran90 based code. Compilation of the code requires the gfortran compiler, and Lapack libraries. Tests have been performed with GCC 4.x 5.x and 6.x, 7.x, and confirmed that consistent results are obtained with these three versions of the gfortran compiler.

Once the main directory CTMQC has been downloaded, go to the directory and

```
cd ./src
make
```

Running the command *make* will compile the source code and generate the executable *main.x*. Go back to the CTMQC directory with the command

```
cd ../
```

and run the script

```
./create_dirs.sh
```

that creates the directory *output* where all output files will be generated. Notice that you should run this script in each new directory where you run the executable. The program generates a series of output files that are saved in different directories. Therefore, in order not to obtain errors during the execution of the program, the directories have to be created.

Testing

CREATE THE OUTPUT DIRECTORY

The directory *output* contains several subdirectories. After successful execution of the program, those subdirectories will contain $N_{\text{files}} = N_{\text{steps}}/N_{\text{dump}}$ files, with N_{steps} the number of total time steps and N_{dump} the number of time steps after which a new output file is generated. In each subdirectory, the files are labelled with an index increasing with time, from 0 to N_{files} . In the current version of the code, up to 999 files can be created.

The following subdirectories of the directory *output* will be created.

```
coeff: [only for one-dimensional calculations]
```

Each file (named *coeff.xxx.dat*) in this directory contains the coefficients of the expansion of the electronic wavefunction in the adiabatic basis as a function of the position of the corresponding trajectory. Each file is in the form: *first column* the position of the trajectory; *following $n \times n$ columns* the real part of $C_k^* C_l$ with $k, l = 1, n$; *following $n \times n$ columns* the imaginary part of $C_k^* C_l$ with $k, l = 1, n$.

```
density: [only for one-dimensional calculations]
```

Each file (named *density.xxx.dat*) in this directory contains the nuclear density reconstructed as the sum of N_{traj} normalized Gaussian functions centered at the position of the trajectories, with N_{traj} the total number of trajectories. The data listed in the file have the form: *first column* the grid in nuclear space, that is read as input from the files containing the potential energy surfaces and nonadiabatic coupling vectors (see section INFORMATION ABOUT THE INPUT FILES below); *second column* the nuclear density. Similarly to this set of files containing the density, additional files are created (named *smooth_density.xxx.dat*) where the density is smoothed by convoluting the density with a Gaussian function of fixed variance.

```
histo: [only for one-dimensional calculations]
```

Each file (named *histo.xxx.dat*) in this directory contains the nuclear density approximated as a histogram that is constructed from the distribution of classical trajectories. The data listed in the file have the form: *first column* the position along the nuclear coordinated (coarser than the original grid, but defined in the same domain); *second column* the normalized histogram.

```
trajectories
```

Each file (named *RPE.xxx.dat*) in this directory contains the values of the phase-space variables and the value of the gauge-invariant part of the time-dependent potential energy surface $\epsilon(\mathbf{R}, t)$, that is the first two terms of its expression (see for instance [EF3]). The data listed in the file have the form: *first* $n_{\text{d.o.f.}}$ *columns* the positions of the trajectories, with $n_{\text{d.o.f.}}$ the number of nuclear degrees of freedom, therefore ranging from 1 to 3; *following* $n_{\text{d.o.f.}}$ *columns* the momenta of the trajectories; *last column* the gauge-invariant part of the time-dependent potential energy surface.

Additionally, the files *BO_population.dat* and *BO_coherences.dat* are created, containing the population of the adiabatic states and the indicator of coherence as functions of time (the first columns is the time). They are defined as

$$\rho_k(t) = \frac{1}{N_{\text{traj}}} \sum_{I=1}^{N_{\text{traj}}} \left| C_k^{(I)}(t) \right|^2$$

and

$$\eta_{kl}(t) = \frac{1}{N_{\text{traj}}} \sum_{I=1}^{N_{\text{traj}}} \left| C_k^{(I)}(t) C_l^{(I)}(t) \right|^2$$

respectively, with $k = 1, \dots, n$.

INFORMATION ABOUT THE INPUT FILES

The directory tests contains input files and input data, i.e. potential energy surfaces and nonadiabatic coupling vectors on a grid, for the one-dimensional model systems known as Tully's models. They are

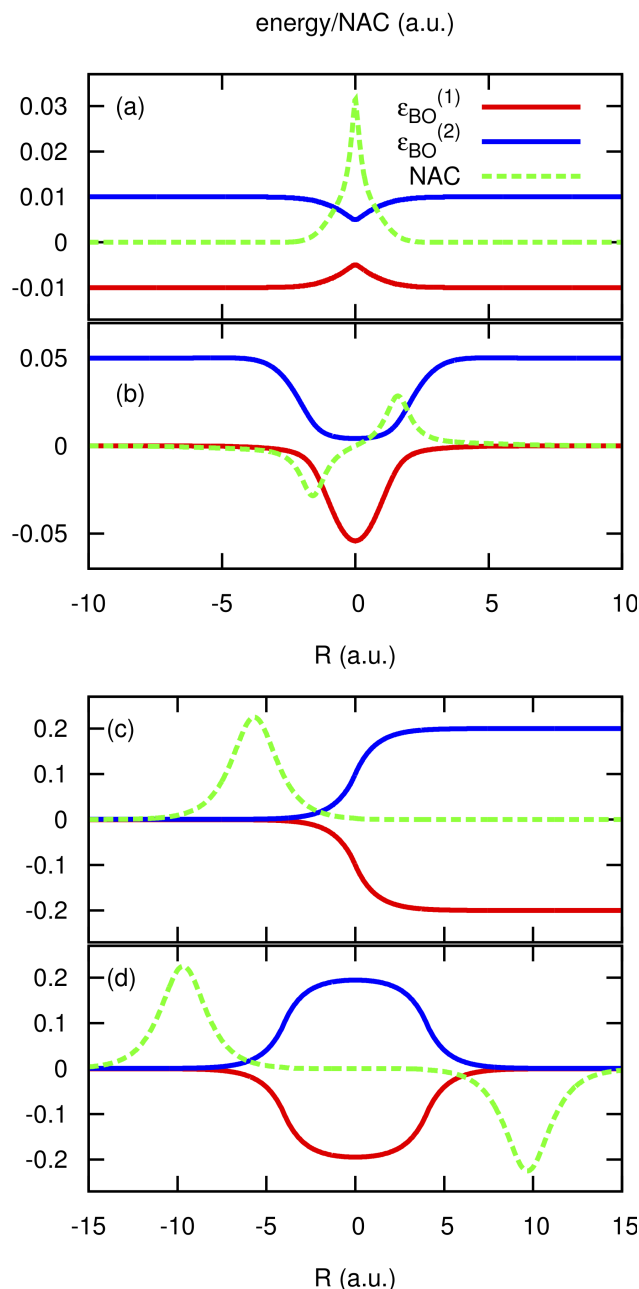
Tully #1: single avoided crossing [panel (a) of the figure below]

Tully #2: dual avoided crossing [panel (b) of the figure below]

Tully #3: extended coupling with reflection [panel (c) of the figure below]

Tully #4: double arch [panel (d) of the figure below]

Analytical expressions of these models can be found in [Tully] [CTMQC2] [CTMQC3], and they are shown in the figure below.



In the directory *tests*, the subdirectories are *tully_1* (containing subdirectories *k0_10au* and *k0_25au*), *tully_2* (containing subdirectories *k0_25au* and *k0_30au*), *tully_3* (containing subdirectories *k0_10au* and *k0_30au*), and *tully_4* (containing subdirectories *k0_20au* and *k0_40au*). For the model **Tully #1** examples are provided for initial momenta of $k_0 = 10, 25$ a.u. as clearly indicated by the name of the subdirectories; for the model **Tully #2** examples are provided for initial momenta of $k_0 = 25, 30$ a.u.; for the model **Tully #3** examples are provided for initial momenta of $k_0 = 10, 30$ a.u.; for the model **Tully #4** examples are provided for initial momenta of $k_0 = 20, 40$ a.u.. The benchmark data provided here are the output files *BO_population.dat* and *BO_coherences.dat*; each subdirectory contains examples of input files.

The directories *tully_1*, *tully_2*, *tully_3*, and *tully_4* contain as well input data: the adiabatic potential energy surfaces *k_bopes.dat* with $k = 1, \dots, n$ labelling the corresponding eigenstate (in the form: *first column* value of the energy; *following* $n_{d.o.f.}$ *columns* the spatial grid in the x, y, z directions); the nonadiabatic coupling vectors *nac1-kl_x*, *nac1-kl_y*, *nac1-kl_z* between states k and l (the form is the same as for the potential energy surfaces), computed as spatial derivatives along the x, y, z directions, respectively.

EXECUTING THE PROGRAM

To run the executable from the chosen directory (after having run the script *create_directories.sh*), write the command

```
./src/main.x < path_to_input
```

where *path_to_input* is the path to the input file. As discussed above, examples of input files are provided in the tests directory, e.g., *tully_1/k0_10au/input.in*.

After the run is completed, and from the main directory, you can run the script

```
./comparison.sh
```

and follow the indications shown in the terminal, to automatically plot the adiabatic populations and the indicator of decoherence that you have generated, and to compare them with the reference results provided.

Source Code

The CTMQC source code and test files can be found at [CTMQC](#).

References

3.5.2 EFAC

Software Technical Information

Language Fortran 90

License GNU Lesser General Public License (LGPL)

Software Module Developed by Hugo Bessone, Lea-Maria Ibele, Emanuele Marsili, Francesco Talotta, David Lauvergnat, Basile F. E. Curchod, Federica Agostini

- *Purpose of Module*
- *Short description*
- *Practical application and exploitation of the code*
- *Installation*
- *Testing*
- *Source Code*
- *References*

Purpose of Module

This module is an analysis tool to be employed for excited-state, nonadiabatic dynamics simulations. The physical situation to be studied is, for instance, the sub-picosecond response of a molecule to an UV/visible ultrashort laser pulse, that excites the molecule electronically. The photo-excited molecule can relax via, so-called, radiationless channels, i.e., internal conversion processes, towards the electronic ground state. To describe such ultrafast processes,

it is essential to account for (i) electronic transitions, thus changes of electronic states, that are induced by nuclear motion, and (ii) the quantum mechanical nature of the nuclei. Various numerical approaches exist nowadays to perform simulations of nonadiabatic processes, based on the – standard, and thus widely used – Born-Huang representation, and on the Exact Factorization. The ultimate goal here is to provide the quantum dynamics community with an easy-to-use analysis tool able to make the link between Born-Huang and Exact Factorization.

Short description

The Exact Factorization Analysis Code provides a post processing tool to transform the result of a molecular quantum dynamics simulation from the Born-Huang representation to the one of the Exact Factorization.

Using the output provided by the grid-based quantum dynamics ElVibRot code [ElVibRot], this module calculates the two key quantities of the Exact Factorization: the *time-dependent potential energy surface* and *time-dependent vector potential* that have been used to offer new perspectives on numerous nonadiabatic processes (see Refs [Gross_PRL2010] , [Gross_JCP2012] , [Agostini_JPCL2017] , [Gross_JCP2015] , [Gross_MP2013]).

The purpose of **EFAC** is to familiarize the user with the framework of the Exact Factorization and to connect it with the more commonly used quantum dynamics methods. Hence, the central purpose of this module is to make the Exact Factorization of the electron-nuclear wavefunction easily accessible to the broad quantum dynamics community.

The two time-dependent potentials of the Exact Factorization can be easily recovered by expressing them in a diabatic or adiabatic basis. This connection offers a bridge between quantum dynamics simulation conducted in the Born-Huang representation and the Exact Factorization; a bridge exploited in this module.

In the framework of the Exact Factorization, the nuclear and electronic wavefunctions are unique up to gauge transformation. While this gauge can in principle be chosen arbitrarily, the current implementation enforces two specific gauges that have proved useful in previous studies (see references above). For one-dimensional simulations, the gauge is fixed by making the time-dependent vector potential equal to zero. While being convenient, this gauge cannot be generalized to higher dimensions. As such, the tool uses a different gauge for problems in higher dimensions where the phase of the nuclear wavefunction is zero, i.e., the nuclear wavefunction will be real and non-negative at all times.

The module uses the result of a grid-based quantum dynamics calculation – (timedependent) nuclear wavefunctions in a diabatic basis and corresponding diagonal and off-diagonal potential surfaces – to construct the time-dependent potential energy surface (*TD PES*). The *TD PES* can be split into three components, two gauge-independent and one gauge-dependent. It outputs separately the two gauge-independent contributions as well as the gauge-dependent one. Additionally, the time dependent vector potential is obtained (*TDVP*).

Practical application and exploitation of the code

This code is intended to provide an easy access to the time-dependent potential energy surface and vector potential of the Exact Factorization from the result of a quantum dynamics simulation in any arbitrary number of dimensions and electronic states. This allows the user to study simple nonadiabatic model systems in low dimensions, but also the more complex nonadiabatic dynamics of molecules through conical intersections.

Installation

The **EFAC** is a fortran90 based code. Compilation of the code requires the `gfortran` compiler.

Once the tarball `EFAC.tar.gz` from the [EFAC repository](#) has been downloaded, create a new directory and untar the file typing

```
tar -zxvf EFAC.tar.gz
```

Compile the source code and generate the executable *EF.x*.


```
make
```

Testing

Go in the directory `test/1d` and copy there the executable `EF.x`. Run the compiled code.

```
./EF.x
```

The executable reads and analyses two files already present there: `EF_parameter_dpsi` and `EF_parameter_gV`. It will generate, for each time-step, three files: `Epsilon`, `Density` and `potential.dat`. In the `Epsilon` file is printed, for each grid point value, the TDPEs and TDVP. In the `Density` file is printed, for each grid point value, the modulus of the wavefunction and the diabatic densities.

`processing.x` is an additional tool provided whenever the number of degrees of freedom are more than 1. Copy the executable in the test directory and run the code. It reads the output files, created with the `EF.x` executable, and it prints the results along a specific degree of freedom. Therefore, it will generate two additional output files called `Epsilon-cut.out` and `Density-cut.out`. In the current version, the program makes the one-dimensional cuts only. In addition, when `processing.x` is run, it requires some parameters that have to be prompted in the following order:

- the time at which the cut has to be performed,
- the number of states,
- the degree of freedom treated as the independent variable and
- the values at which all the other degrees of freedom are fixed.

```
./processing.x 1 2 2 0.2
```

In this 2d case, the 2nd degrees of freedom is the independent variable while the first degrees of freedom is fixed at the value of 0.2. The `processing.x` will use `Density001.dat` and `Epsilon001.dat` files, containing the information after 1 fs of propagation.

Source Code

The EFAC source code and test files can be found at [EFAC](#).

References

The **CTMQC** module allows to simulate excited-state dynamics in model systems of one to three spatial (nuclear) dimensions, with an arbitrary number of electronic states. The algorithm is based on the quantum-classical approximation of the equations of motion derived in the framework of the exact factorization of the electron-nuclear wavefunction. In practice, trajectories are used to mimic the nuclear evolution, that is, in turn, coupled to the quantum evolution of the electronic degrees of freedom.

3.5.3 Trotter Based Quantum Classical Surface Hopping Propagator - Single Path

Software Technical Information

Language C++ (C++11 or higher)

Licence MIT licence (MIT)

Documentation Tool Doxygen

Application Documentation [Documentation](#)

Software Module Developed by Sean Kelly, Athina Lange, Philip McGrath, Shrinath Kumar and Donal MacKernan

- *Abstract*
- *Purpose of Module*
- *Background Information*
- *Applications*
- *Algorithms and Software Implementation*
- *Compiling*
- *Checking for accuracy*
- *Testing, Performance and Scaling*
- *Source Code*
- *Source Code Documentation*
- *References*

Abstract

The present module is a highly refactored version of a code based on a highly cited algorithm published by D. Mackeran, G.Ciccotti and R. Kapral [Mackernan]. The module software has been entirely refactored in modern C++ (GNU 2011 or higher) so as to: (a) run with high-efficiency on massively parallel platforms under OpenMP or MPI; and (b) be at the core of additional software modules aimed at addressing important issues such as improving the speed of convergence of estimates using correlated sampling, and much more realistic treatment of the classical bath, and connecting to other problems such as constant pH simulation through an effective Hamiltonian.

Purpose of Module

Quantum rate processes in condensed phase systems are often computed by combining quantum and classical descriptions of the dynamics including non-adiabatic coupling, using propagators which amount to quantum path integrals in a partial Wigner phase space representation, such as the mixed quantum-classical Dyson equation and variants thereof, or the Trotter decomposition of the quantum-classical propagator.

Background Information

An understanding of the dynamical properties of condensed phase quantum systems underlie the description of a variety of quantum phenomena in chemical and biological systems. The development of schemes for the efficient and accurate simulation of the quantum dynamics of such systems is an active area of research in chemical physics, and is essential if problems of chemical interest involving complex molecular species in the condensed phase are considered.

In investigations of the dynamical properties of quantum statistical mechanical systems, one is often interested in the average value of some operator when the system evolves from a given initially prepared distribution described by the density matrix $\hat{\rho}(0)$. In such cases the quantum mechanical average value of an operator \hat{B} is given by $\overline{B(t)} = \text{Tr} \hat{B} \hat{\rho}(t) = \text{Tr} \hat{B}(t) \hat{\rho}(0)$. Here, $\hat{B}(t)$ evolves in time through the Heisenberg equation of motion. In many applications, it is useful to partition the system into a subsystem and a bath. A phase space description of the bath can be obtained by taking a partial Wigner transform over the bath coordinate $\{Q\}$ representation of the full quantum system. In this partial Wigner representation the expectation value of $\hat{B}(t)$ takes the

$$\overline{B(t)} = \text{Tr}' \int dR dP B_W(R, P, t) \rho_W(R, P)$$

where the prime on the trace indicates a trace over the subsystem degrees of freedom.

The software module developed here is based on a Trotter-based scheme for simulating quantum-classical Liouville dynamics in terms of an ensemble of surface-hopping trajectories. The method can be used to compute the dynamics for longer times with fewer trajectories than the sequential short-time propagation (SSTP) algorithm, which is also based on surface-hopping trajectories. The full derivation of the algorithm is given in [Mackernan]. Here the software focus is to refactor the original code which until now was a purely serial so that it can be used efficiently on massively parallel machines. For mathematical details, we refer the reader to eq.30-35 of the paper.

Applications

The applications of quantum surface hopping include, among others, non-adiabatic chemical rate processes involving electronic, vibrational or other degrees of freedom, decoherence in open quantum systems and quantum transport processes. Decoherence due to coupling with the environment is a fundamental difficulty in the development of quantum computing. The ability to predict, control and reduce decoherence requires an adequate description of the associated non-adiabatic processes taking place. Quantum effects and frequently non-adiabaticity also underlie the study of ultra-fast rate processes in solution.

Algorithms and Software Implementation

The current Single Path code has three main advantages over the original version. First it is separated into files based on function for better readability. For example the ‘transition_matrix.cpp’ file is where the transition matrix and associated functions are defined, etc. Secondly input parameters are read from an Input file, so the code no longer needs to be recompiled to adjust these parameters. And finally the code has been altered to run in parallel which allows for a significant reduction in runtime.

Compiling

All current versions of this code use the [GNU scientific library](#) version 2.5 for random number generation.

OpenMP version:

With the GNU compiler, gcc version 6.3.0 or greater is required.

On the Kay cluster this can be done as follows:

```
module load gcc/8.2.0
module load gsl/gcc/2.5
```

```
Compile command;
g++ -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_matrix.cpp_
↳ opt_parser.cpp -lgsl -lgslcblas -lm -fopenmp -std=c++11
```

(continues on next page)

(continued from previous page)

```
Run command:
export OMP_NUM_THREADS=[number of OpenMP threads]; ./run Input
```

With the Intel compiler:

```
Compile command;
icpc -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_matrix.cpp_
↳opt_parser.cpp -lgsl -lgslcblas -lm -qopenmp -std=c++11

Run command:
export OMP_NUM_THREADS=[number of OpenMP threads]; ./run Input
```

MPI version:

```
module load intel/2018u4
module load gsl/intel/2.5
module load gcc/8.2.0
```

```
Compile command;
mpic++ -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_matrix.
↳cpp opt_parser.cpp -lgsl -lgslcblas -lm -std=c++11

Run command:
mpirun -n [number of MPI processors] ./run Input
```

Errors:

A frequent error encountered while compiling is: “fatal error: gsl/gsl_rng.h: No such file or directory”

This can occur if the directory is not installed on the standard search path of the compiler. It can be fixed by adding it's location as a flag in the compile command as explained in this link: [Using the GSL Library](#).

On Kay the flags ‘-I/ichec/packages/gsl/gcc/2.5/include’ and ‘-L/ichec/packages/gsl/gcc/2.5/lib’ must be added to the compile command as:

```
g++ -o run main.cpp bath_setup.cpp density.cpp propagation.cpp transition_matrix.cpp_
↳opt_parser.cpp -lgsl -lgslcblas -lm -fopenmp -std=c++11 -I/ichec/packages/gsl/gcc/2.
↳5/include -L/ichec/packages/gsl/gcc/2.5/lib
```

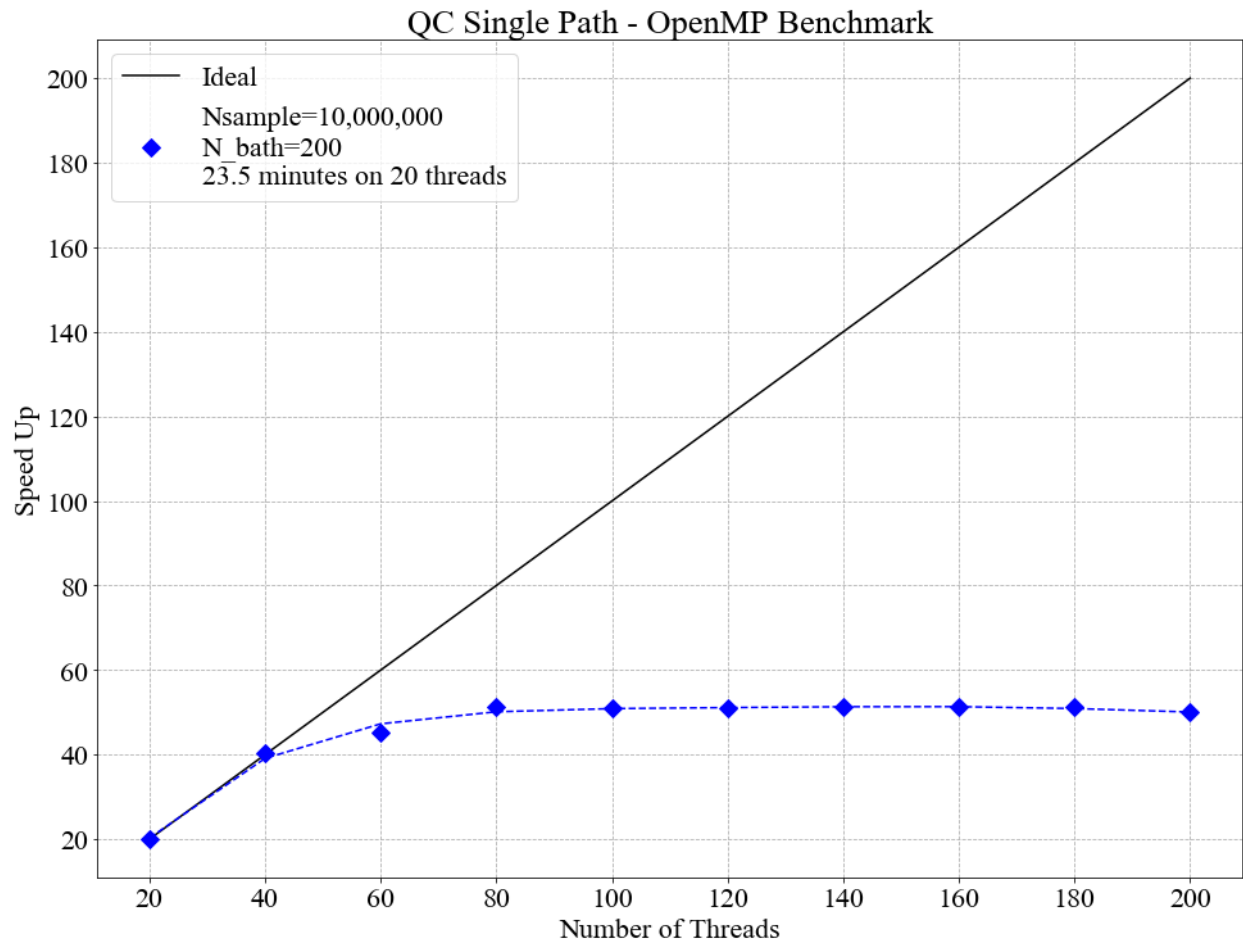
Checking for accuracy

The original serial code was run 1000 times to generate an expected output and variance. These can be found in the ./Regression_testing sub-directory. A regression test is built into both the OpenMP and MPI versions which checks if their output is within five standard deviations of the expected output (given a specific set of input parameters). If any part of the output goes outside that limit the regression test will fail. (Note: To run a test ‘Regression_test=1’ must be set in the Input file along with a standard set of parameters. All of this is specified in the Input file).

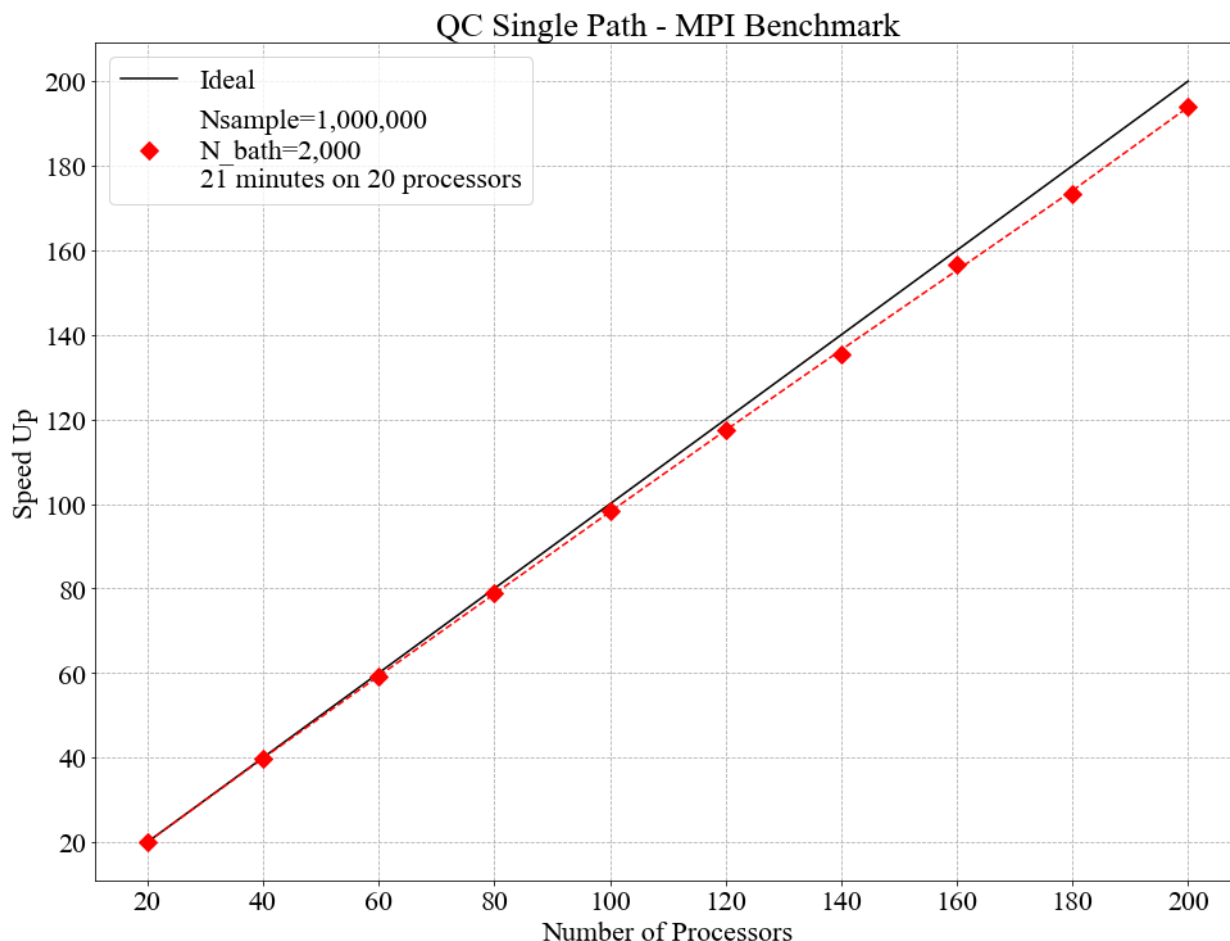
Testing, Performance and Scaling

Testing was performed on the Kay supercomputer from ICHEC. Kay is separated into nodes, each of which has 2 x (20 core) sockets. To test the parallel efficiency of both the OpenMP and MPI versions of the code they were benchmarked on 20 - 200 cores (1 - 5 nodes).

The OpenMP version was run for 10,000,000 samples ($N_{\text{sample}} = 10,000,000$) and for a bath size of 200 ($N_{\text{bath}} = 200$). As can be seen in the graph below OpenMP scales perfectly on a single node (i.e. less than 40 cores), but provides little to no benefit over multiple nodes.



The MPI version was run for 1,000,000 samples ($N_{\text{sample}} = 1,000,000$) and for a bath size of 2,000 ($N_{\text{bath}} = 2,000$). MPI scales very well over the entire benchmark (up to 200 cores), with an average efficiency of 96.3%.



Source Code

The source codes for the OpenMP and MPI versions of the code are:

[Surface Hopping - OpenMP version](#)

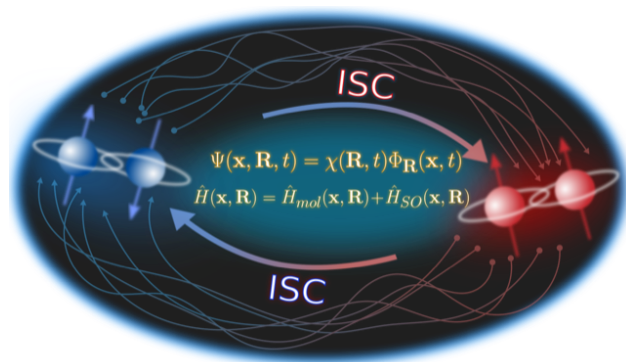
[Surface Hopping - MPI version](#)

Source Code Documentation

The source code documentation is given at <https://gitlab.e-cam2020.eu/Quantum-Dynamics/Surface-Hopping/tree/master/Doc>. These documentation files can be updated by executing the `make` command in the `Doc` directory.

References

The **SinglePath** module uses combined quantum and classical descriptions of the dynamics to compute quantum rate processes in condensed phase systems. The main purpose of this module is to act as the core of additional software modules aimed at addressing important issues such as improving the speed of convergence of estimates using correlated sampling, and much more realistic treatment of the classical bath, and connecting to other problems such as constant pH simulation through an effective Hamiltonian.

Software Technical Information**Name** G-CTMQC**Language** Fortran 90**Licence** GNU Lesser General Public License (LGPL)**Documentation Tool** doxygen**Software Module Developed by** Federica Agostini, Emanuele Marsili, Francesco Talotta**3.5.4 G-CTMQC**

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*
- *References*

Studies in the domain of photochemistry and photophysics strongly rely on simulation methods able to describe strong coupling between electronic and nuclear motion on the femtosecond time scale, usually called nonadiabatic coupling. Simulations give access to microscopic information in terms of molecular structures, electronic populations, vibrational energies that can be easily compared to experiments, for instance in the domains of time-resolved spectroscopy or 2D spectroscopy. In addition to this, molecular dynamics simulations allow to follow in real time the evolution of molecular systems, thus providing support to interpret and even predict the outcome of experiments.

Photochemical and photophysical reactions are ubiquitous in nature, from photosynthesis to vision, and are more and more exploited for technological advances, as for the photo-current production in organic photovoltaic devices. In addition, it is becoming clear the importance to consider spin-orbit coupling even in those systems composed of light elements, such as oxygen and carbon, to be able to describe processes such as intersystem crossings in organic light-emitting diodes.

G-CTMQC module provides numerical tools to perform simulations of internal conversion (spin-allowed) and intersystem crossing (spin-forbidden) phenomena underlying photochemical and photophysical reactions. G-CTMQC gives the user the flexibility of employing different approaches and, thus, various approximation schemes, to achieve dynamical information as accurate as possible, as well as ample flexibility in the choice of systems that be studied thanks to the interface of G-CTMQC with QuantumModelLib (E-CAM module).

Purpose of Module

G-CTMQC is a module for excited-state molecular dynamics simulations with various trajectory-based algorithms, including nonadiabatic coupling and spin-orbit coupling. Nuclear dynamics can be performed based on the quantum-classical algorithm derived from the exact factorization of the electron-nuclear wavefunction [EF], dubbed CT-MQC [CT-MQC]. Recently, the extension of the exact-factorization theory has been proposed to include spin-orbit coupling [SOC]. Therefore, the “generalized” algorithm is now able to treat (i) standard nonadiabatic situations, where spin-allowed electronic transitions among states with the same spin multiplicity are mediated by the coupling to nuclear motion, and (ii) spin-orbit interactions, where spin-forbidden electronic transitions among states of different spin multiplicity are induced by the spin-orbit coupling.

Electronic evolution is carried out in the adiabatic basis for standard nonadiabatic problems. In the case of spin-orbit interactions, **G-CTMQC** offers the options to use the spin-diabatic or the spin-adiabatic representations. Information about electronic-structure properties, ie, energies, gradients and couplings, is calculated and read on-the-fly at the positions of the trajectories at each time step based on the QuantumModelLib library [4] of potentials (which **G-CTMQC** is interfaced to).

In addition, the code offers the possibility of performing calculations with the trajectory surface hopping algorithm [TSH] and the Ehrenfest approach [EH]. Concerning the trajectory surface hopping method, the fewest switches scheme is implemented, along with the energy decoherence corrections to fix the overcoherence issue of surface hopping [TSH-EDC]. For surface hopping and Ehrenfest, only nonadiabatic couplings are currently implemented.

Background Information

Detailed information about the exact factorization and CT-MQC [EF] can be found in **CTMQC** where the original version of the module is described. The generalized CTMQC, **G-CTMQC**, includes various new features to original module:

- spin-allowed, between electronic states of the same spin multiplicity, and spin-forbidden, between electronic states of different spin multiplicity, transitions can be simulated; the former are mediated by the kinetic, also called nonadiabatic, coupling between electronic and nuclear motion, whereas the latter are induced by spin-orbit coupling;
- G-CT-MQC calculations, based on the generalized coupled-trajectory mixed quantum-classical algorithm, can be performed in the spin-diabatic and spin-adiabatic basis for the electronic subsystem;
- nonadiabatic calculations based on trajectory surface hopping [TSH] and on the Ehrenfest approach [EH] can be carried out, including energy decoherence corrections in surface hopping [TSH-EDC]; the fewest switches scheme is used for surface hopping;
- on-the-fly dynamics can be performed based on the calculation of electronic structure information, namely energies, gradients and couplings, along the trajectories via the interface to the QuantumModelLib library.

The new features introduced in **G-CTMQC** are documented in Refs. [SOC] and [G-CT-MQC] concerning the inclusion of spin-orbit coupling in the exact factorization and in **G-CTMQC**, in Refs. [PSB3] and [IC] concerning the inclusion of trajectory surface hopping, Ehrenfest dynamics, and different possibilities of sampling the initial conditions.

Building and Testing

G-CTMQC is a fortran90 based code. Compilation of the code requires the gfortran compiler, and Lapack libraries. Tests have been performed with GCC 7.x. Note that, before compiling **G-CTMQC** it is necessary to compile the potential library available [here](#) and copy the file *libpot.a* into the *src* directory of **G-CTMQC**.

Once the main directory CTMQC has been downloaded, go to the directory and


```
cd ./src
make
```

Running the command *make* will compile the source code and generate the executable *main.x*. Go back to the CTMQC directory with the command

```
cd ../
```

and run the script

```
./create_dirs.sh
```

that creates the directory output where all output files will be generated. Notice that you should run this script in each new directory where you run the executable. The program generates a series of output files that are saved in different directories. Therefore, in order not to obtain errors during the execution of the program, the directories have to be created.

CREATE THE OUTPUT DIRECTORY

The directory output contains several subdirectories. After successful execution of the program, those subdirectories will contain $N_{\text{files}} = N_{\text{steps}}/N_{\text{dump}}$ files, with N_{steps} the number of total time steps and N_{dump} the number of time steps after which a new output file is generated. In each subdirectory, the files are labelled with an index increasing with time, from 0 to N_{files} . In the current version of the code, up to 999 files can be created.

The following subdirectories of the directory *output* will be created.

```
coeff
```

Each file (named *coeff.xxx.dat*) in this directory contains the coefficients $C_k^{(I)}(t)$ of the expansion of the electronic wavefunction in the used electronic basis as a function of the position of the corresponding trajectory I . Each file is in the form: the *first* N_{dof} *columns* are the positions of the trajectories for each of the N_{dof} nuclear degrees of freedom; the *following* $n \times n$ *columns* are the real parts of $[C_k^{(I)}(t)]^* [C_l^{(I)}(t)]$ with $k, l = 1, n$ and n the number of electronic states considered in the expansion; the *following* $n \times n$ *columns* are the imaginary parts of $[C_k^{(I)}(t)]^* [C_l^{(I)}(t)]$ with $k, l = 1, n$.

```
histo: [only for one-dimensional calculations]
```

Each file (named *histo.xxx.dat*) in this directory contains the nuclear density approximated as a histogram that is constructed from the distribution of classical trajectories. The data listed in the file have the form: *first column* the position along the nuclear coordinated (coarser than the original grid, but defined in the same domain); *second column* the normalized histogram.

```
trajectories
```

Each file (named *RPE.xxx.dat*) in this directory contains the values of the phase-space variables and the value of the gauge-invariant part of the time-dependent potential energy surface. The data listed in the file have the form: the *first* N_{dof} *columns* are the positions of the trajectories for each of the N_{dof} nuclear degrees of freedom; the *following* N_{dof} *columns* are the momenta of the trajectories for each of the N_{dof} nuclear degrees of freedom; the *following column* is the gauge-invariant part of the time-dependent potential energy surface; the *following* n *columns* are the adiabatic energies.

Additionally, the files *BO_population.dat* and *BO_coherences.dat* are created, containing the population of the adiabatic states and the indicator of coherence as functions of time (the first columns is the time in atomic units). They are defined as

$$\rho_k(t) = \frac{1}{N_{\text{traj}}} \sum_{I=1}^{N_{\text{traj}}} |C_k^{(I)}(t)|^2$$

and

$$\eta_{kl}(t) = \frac{1}{N_{traj}} \sum_{I=1}^{N_{traj}} \left| C_k^{(I)}(t) C_l^{(I)}(t) \right|^2$$

respectively, with $k = 1, \dots, n$.

PROVIDED TESTS AND INPUT FILE

In the main CTMQC directory the

```
tests
```

directory provides examples of input files to run one-dimensional calculations with CT-MQC, surface hopping and Ehrenfest on Tully model #3 [TSH] and some reference calculations.

```
&SYSTEM
  TYP_CAL           = "XX"           !*character* XX = CT (CT-MQC calculations), EH_
↳ (Ehrenfest calculations), SH (surface hopping calculations)
  SPIN_DIA          = X              !*logical* X = T only for calculations with spin-
↳ orbit coupling in the spin-diabatic basis, otherwise X = F
  NRG_CHECK          = X              !*logical* X = T to switch off the spin-orbit_
↳ coupling when the energy between states is larger than NRG_GAP
  NRG_GAP            = X              !*real* only for calculations with spin-orbit_
↳ coupling in the spin-diabatic basis
  MODEL_POTENTIAL    = "XXXXX"       !*character* XXXXX = definition of the model as it_
↳ appears in QuantumModelLib
  OPTION             = X              !*integer* X = 1, 2, 3 for Tully's models #1, #2,
↳ #3 (only used for Tully's models calculations)
  N_DOF              = X              !*integer* X = number of nuclear degrees of freedom
  PERIODIC_VARIABLE  = X,X,X...      !*logical* one value for each nuclear degree of_
↳ freedom with X = T (periodic coordinate) or F
  PERIODICITY        = X,X,X...      !*real* one value for each nuclear degree of_
↳ freedom with X = the period in units of PI
  NSTATES            = X              !*integer* X = number of electronic states
  M_PARAMETER        = X,X,X...      !*real* one value for each nuclear degree of_
↳ freedom with X = typical distance to tune the coupling among the trajectories in CT_
↳ calculations
  QMOM_FORCE         = X              !*logical* X = F to switch off the force from the_
↳ quantum momentum (only) in CT calculations
  DECOHERENCE        = X              !*logical* X = F for surface hopping or T for_
↳ surface hopping with energy decoherence corrections
  C_PARAMETER        = X              !*real* energy parameter for the energy_
↳ decoherence correction in surface hopping
  JUMP_SEED          = X              !*integer* seed for random number generator for_
↳ the hopping algorithm in SH calculation
/

&DYNAMICS
  FINAL_TIME         = X              !*real* X = length of the simulation in atomic_
↳ units
  DT                 = X              !*real* X = integration time step in atomic units
  DUMP                = X              !*integer* X = number of time steps after which_
↳ the output is written
  INITIAL_BOSTATE     = X              !*integer* X = initial electronic state
  NTRAJ               = X              !*integer* X = number of classical trajectories
  R_INIT              = X,X,X...      !*real* one value for each nuclear degree of_
↳ freedom with X = average position of the initial nuclear distribution
  K_INIT              = X,X,X...      !*real* one value for each nuclear degree of_
↳ freedom with X = average momentum of the initial nuclear distribution
```

(continues on next page)

(continued from previous page)

```

    SIGMAR_INIT      = X,X,X...      !*real* one value for each nuclear degree of_
    ↪freedom with X = variance in position space of the initial nuclear distribution
    SIGMAP_INIT      = X,X,X...      !*real* one value for each nuclear degree of_
    ↪freedom with X = variance in momentum space of the initial nuclear distribution
    MASS_INPUT       = X,X,X...      !*real* one value for each nuclear degree of_
    ↪freedom with X = the nuclear mass
/

&EXTERNAL_FILES
    POSITIONS_FILE   = "XXXXX"      !*character* XXXXX = file containing the list of_
    ↪initial positions for the trajectories; if the field is empty, positions are_
    ↪sampled according to R_INIT and SIGMAR_INIT
    MOMENTA_FILE     = "XXXXX"      !*character* XXXXX = file containing the list of_
    ↪initial momenta for the trajectories; if the field is empty, momenta are sampled_
    ↪according to K_INIT and SIGMAP_INIT
    OUTPUT_FOLDER    = "XXXXX"      !*character* XXXXX = path to the location where_
    ↪the output is written
/

```

Source Code

The **G-CTMQC** source code and test files can be found following this [link](#).

References

The **G-CTMQC** module extends the previous **CTMQC** module, introducing new methodological and technical features. G-CTMQC is interfaced with the **QuantumModelLib** library of potentials, which gives more flexibility in the choice of systems that can be studied. The present implementation allows to perform surface hopping calculations, also with inclusion of energy decoherence corrections, and Ehrenfest dynamics, as well as CT-MQC calculations. Finally, spin-orbit coupling is included in CT-MQC (G-CT-MQC algorithm).

3.5.5 E-CAM Physical Constant module

Software Technical Information

Language Fortran 95

Compiler gfortran, ifort

Licence GNU Lesser General Public License (LGPL)

Documentation Tool Doxygen

- *Purpose of Module*
- *Background Information*
- *Installing*
- *Testing*

- *Source Code*

Purpose of Module

This module enables to use fundamental physical constants (speed of light in vacuum, Planck constant ... and isotopic masses). Two versions can be selected:

- The CODATA 2006 ones, downloaded from [NIST](#) and the NIST masses downloaded in 2012 (default).
- Constants and masses from the 70th edition of the Handbook of Chemistry and Physics.

From these fundamental constants, some conversion factors are calculated automatically and can be used easily.

Remark: the actual mass values of the NIST web page differ slightly from the module ones.

Background Information

This module has been extracted and modified from the ElVibRot-Tnum-Tnum code (ElVibRot_f90-v80.13-Tnum28.9-Tana5.1). This pre-E-CAM version can be downloaded [here](#).

Installing

Dependencies: this module needs the fortran modules in the `Source_Lib/sub_system` directory.

Build the module (with dependencies):

```
make PhysConst
```

Build the module documentation (with doxygen):

```
make doxy
```

Testing

Two example data/script files:

```
Examples/exa_PhysicalConstants/dat_PhysConst  
Examples/exa_PhysicalConstants/dat_PhysConst_HandBook70ed
```

To test the installation, you can run both test examples.

```
cd Examples/exa_PhysicalConstants ; ./run_tests
```

The results will be compared to previous results in `Examples/exa_PhysicalConstants/output_17dec2016` file.

Source Code

The source code can be downloaded from the [E-CAM GitLab service](#).

The **PhysConst** enables the use of physical constants and the correct isotopic masses.

3.5.6 Quantum Model Library module

Software Technical Information

Language Fortran 2003

Compiler

- gfortran (version 6.0.3 linux and OSX)
- ifort (version: 14.0.2, 16.0.3, 17.0.1 linux)

Licence GNU Lesser General Public License (LGPL)

Documentation Tool Doxygen

- *Purpose of Module*
- *Applications of the Module*
- *Installing*
- *Testing*
- *Source Code*

Purpose of Module

This module enables to use potential energy surfaces extracted from the literature. It has the following features:

- One or several degrees of freedom
- One or several electronic states
- For each electronic state, the energy, gradient and hessian can be obtained in the diabatic or adiabatic representations
- The gradient and the hessian can be computed analytically (even for the adiabatic representations) or numerically (with finite differences)

Applications of the Module

In standard quantum dynamics approaches (when the wave function is expanded on a grid and/or on a basis set), it is essential to use potential energy surfaces in an analytical form which have to be linked to the quantum dynamics code.

This module contains several model potentials from the literature, which can be called using simple fortran subroutines.

For instance, the phenol potential [1] (2 coordinates, 3 electronic surfaces) is called with the following fortran line:

```
CALL sub_model1_V(V,Q,ndim,nsurf,pot_name,option)
```

or

```
CALL sub_model1_V(V,Q,2,3,'phenol',option)
```

with pot_name='phenol' and where V is a 3x3 matrix representing the adiabatic potential (nsurf=3), Q a vector with 2 components (ndim=2) associated to the 2 coordinates.

The diabatic potential can be obtained by calling the “sub_model1_DiaV” subroutine with the same arguments.

[1] Z. Lan, W. Domcke, V. Vallet, A.L. Sobolewski, S. Mahapatra, J. Chem. Phys. 122 (2005) 224315

Installing

Dependencies: none

Build the library (with dependencies):

```
make lib
```

=> it creates a “libpot.a” library, where the subroutine “sub_model1_V” and others are present.

Build a driver to show how to call subroutines from an external program:

```
make driver
```

=> it creates a “Driver.x” executable file.

Build the module documentation (with doxygen):

```
make doxy
```

Testing

To test the installation, you can run the script “run_tests” in Tests directory:

```
cd Tests ; ./run_tests
```

The script tests two aspects:

- The “ModLib” library with the implemented potentials
- The “dnSLib” library in which the value, first, second and third derivatives of intrinsic fortran functions (sin, cos ...) are implemented as generic functions. Furthermore, the usual operations (+, -, *, /, **) and comparison operators (==, > ...) are also implemented.

The results will be compared to previous results in Tests/RES_old

Source Code

The source code can be downloaded from the [E-CAM gitlab](#).

The **QuantumModelLib** use potential energy surfaces extracted from the literature and can be linked to quantum dynamics codes.

Software Technical Information

Name FBTS-MPI

Language Fortran

Licence [MIT](#)

3.5.7 FBTS_MPI

- *Purpose of Module*
- *Background Information*
- *Applications*
- *Building and Testing*
- *Source Code*
- *References*

Purpose of Module

The FBTS-MPI module implements the Forward-Backward Trajectory Solution (FBTS) to the quantum-classical Liouville equation [KapralCiccotti1999] developed by Hsieh and Kapral [HsiehKapral2012], [HsiehKapral2013].

In the case of a many-body system that can be partitioned into a quantum subsystem and classical environment, this module can be used in the calculation of time-dependent observables. The purpose of this module is to provide an efficient and approximate method to study the nonadiabatic dynamics of these systems.

Background Information

In this approximate quantum dynamics method both the quantum subsystem and classical-like environment are transformed into a continuous phase space representation. This is achieved through a partial Wigner transform over the environmental degrees of freedom and a mapping representation for the quantum subsystem, wherein the subsystem degrees of freedom are represented by coherent state variables: $z_\lambda = (q_\lambda + ip_\lambda)/\sqrt{2\hbar}$.

Classical-like equations of motion are then used to evolve an ensemble of Monte Carlo sampled trajectories through time and the matrix elements of the average value of a time-dependent operator, (having undergone the Wigner transform):

$$\langle B(t) \rangle = \sum_{\lambda\lambda'} \int dX B_W^{\lambda\lambda'}(X, t) \rho_W^{\lambda'\lambda}(X)$$

is calculated by the FBTS method using,

$$B_W^{\lambda\lambda'}(X, t) = \sum_{\mu\mu'} \int dx dx' \phi(x) \phi(x') \frac{1}{\sqrt{2\hbar}} * (q_\lambda + ip_\lambda)(q'_\lambda - ip'_\lambda) B_W^{\mu\mu'}(X_t) \frac{1}{\sqrt{2\hbar}} * (q_\mu(t) - ip_\mu(t))(q'_\mu(t) + ip'_\mu(t))$$

where $(X, x, x') = (R, P, q, q', p, p')$ and $\phi = (2\pi\hbar^{-N})e^{-\sum(q_v^2 + p_v^2/2\hbar)}$.

Applications

The particular system that this FBTS-MPI module has been built for is in the study of excitation energy transfer in biological light harvesting systems, so-called protein-pigment complexes, through the use of the Frenkel exciton model. [IshizakiFleming2009PNAS] The total Hamiltonian of this system is: $\hat{H}_{total} = \hat{H}_s + \hat{H}_b + \hat{H}_{sb}$.

In this model the quantum subsystem of interest, \hat{H}_s , is the electronic excited states of the pigment molecules, the surrounding vibrational environment, \hat{H}_b , is represented as a collection of harmonic oscillators and the interaction between the two, \hat{H}_{sb} , is characterized by the spectral density.

Specifically, the subsystem Hamiltonian is built such that the diagonal elements is the site energy, ϵ_j of a particular pigment, j , with the coupling between the pigments on the diagonals, Δ_{kj} :

$$\hat{H}_s = \sum_{j=1}^J \epsilon_j |j\rangle\langle j| + \sum_{k \neq j} \Delta_{kj} |k\rangle\langle j|$$

The Hamiltonian of the bath is written as, where N is the total number of bath oscillators:

$$\hat{H}_b = \frac{1}{2} \sum_{j,n}^{J,N} \left(\hat{P}_{j,n}^2 + \omega_{j,n}^2 \hat{Q}_{j,n}^2 \right)$$

Lastly, the coupling Hamiltonian:

$$\hat{H}_{sb} = - \sum_{j,n} c_{j,n} \hat{Q}_{j,n} |j\rangle \langle j|$$

In this module an approximate form of the spectral density is used, known as the Debye spectral density given below:

$$J_D(\omega) = \frac{2\lambda\omega_c\omega}{\omega^2 + \omega_c^2}$$

The initial application for this module is in examining the mechanisms of exciton transport, which can be studied through the time-dependent exciton site populations for a given light-harvesting complex. The approximate nature of this dynamics method combined with the parallelization of the trajectory ensemble allows one to model exciton transport in large systems with many pigments that would otherwise be prohibitively expensive to simulate.

Building and Testing

In order to compile this module, two files are required, `FBTS_MPI.f90` and `luxury.f90`, one contains the FBTS method and the other returns a random number. Both of these files are located in the `./source` sub-directory and can be compiled using:

```
mpifort FBTS_MPI.f90 luxury.f90 -o FBTS_MPI.x
```

Upon successful compilation of the code execution of the code requires two input files, one containing relevant information concerning the simulation and the subsystem Hamiltonian matrix in units of wavenumbers.

The file `Input_Data.dat` contains the simulation parameters and can be easily modified. The number of states of the system, the state in which the initial excitation will occur and the number of trajectories this module will complete can be changed. The influence of the bath can also be adjusted through the parameters that will define the Debye spectral density, the characteristic frequency of the bath, ω_c , the reorganization energy and the number of bath oscillators.

There are three parameters that concern the time length of the simulation, `num_timestep`, `timestep` and `timestep_block`. The total time length of the simulation is determined by: `num_timestep * timestep`. The parameter `timestep_block` determines at what interval the time-dependent observables will be calculated and collected.

An example of this `Input_Data.dat` file and subsystem Hamiltonian matrix can be found in the `./tests/Dimer_Model` sub-directory. In order to test the code move the executable to the this sub-directory and compare the output site populations against the exact results from [IshizakiFleming2009] Figure 4(b). Remember that the output provided by the module is given in atomic units of time and must be converted to femtoseconds to compare.

Another model is provided for testing, the light harvesting complex known as the Fenna-Matthews-Olson (FMO) complex that contains 7 states, the exact results are from [WilkinsDattani2015].

The output from the `FBTS_MPI` module should be in good agreement to the exact results.

Source Code

The `FBTS_MPI` module source code is located at: [FBTS_MPI](#).

References

The `FBTS_MPI` module implements the Forward-Backward Trajectory Solution (FBTS) to the quantum-classical Liouville equation developed by Hsieh and Kapral.

3.5.8 PaPIM

PaPIM is a code for calculation of equilibrated system properties (observables). Some properties can be directly obtained from the distribution function of the system, while properties that depends on the exact dynamics of the system, such as the structure factor, [Mon2] infrared spectrum [Beu] or reaction rates, can be obtained from the evolution of appropriate time correlation functions. PaPIM samples either the quantum (Wigner) or classical (Boltzmann) density functions and computes approximate quantum and classical correlation functions.

The code is highly parallelized and suitable for use on large HPC machines. The code's modular structure enables an easy update/change of any of its modules. Furthermore the coded functionalities can be used independently of each other. The code is specifically design with simplicity and readability in mind to enable any user to easily implement its own functionalities. The code has been extensively used for the calculation of the infrared spectrum of the CH_5^+ cation in gas phase, while recently new calculations on the water dimer, and protonated water dimer systems were started.

PaPIM

Software Technical Information

Language Fortran 90/95

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Momir Mališ, Ari P. Seitsonen

- *Purpose of Module*
- *Phase Integration Method (PIM)*
- *Applications of the Module*
- *Compiling*
- *Testing*
 - *Performance and Benchmarking*
- *Source Code*
- *Source Code Documentation*
- *References*

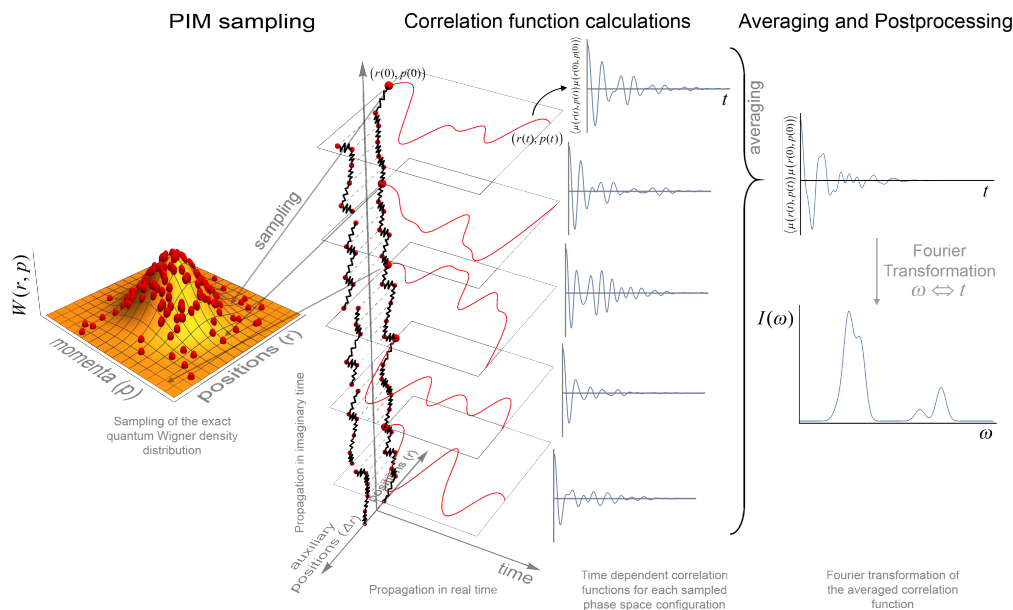
Purpose of Module

PaPIM is a code for computing time-dependent correlation functions and sampling of the phase space. It samples the phase space either classically or quantum mechanically. For the classical sampling of the phase space a Monte Carlo algorithm samples the Boltzmann distribution function, while for the quantum sampling a Phase Integration Method (PIM) [PMon1] [PMon2] is utilized for an exact sampling of the quantum Wigner density distribution. From the sampled phase space points trajectories are propagated in time using classical molecular dynamics in order to obtain the appropriate time-dependent correlation functions. The code is designed so the user can easily couple it with its own external potential energy code/library and/or correlation functions subroutines. Examples for the OH, CH_4 and CH_5^+ systems are provided, where for the CH_5^+ system an external subroutine for calculation of CH_5^+ system

potential energy and electric dipole moment, based on fitted values, [PJin] is also provided. The first two systems use Morse and harmonic potential, respectively. PaPIM comes with a direct interface to the CP2K program package for calculation of system's electronic structure properties. The CP2K package is coupled to PaPIM as a library, thus avoiding the exchange of information process via writing and reading to an external file. For more information see the corresponding *PaPIM-CP2K_Interface* module. .. Example external subroutines are provided for the OH and CH₄ systems, respectively, .. with potential energies described by the harmonic potential, .. and the electric dipole moments by point charge approximation. .. An external subroutine for calculation of .. CH₅⁺ system potential energy and electric dipole moment, based on fitted values, [PJin] is also given. The electric dipole moment operator is currently implemented into the code for calculation of the electric dipole moment autocorrelation function from which system IR spectra can be directly obtained.

Phase Integration Method (PIM)

The Phase Integration Method (PIM) is a novel approximate quantum dynamical technique developed for computing systems time dependent observables. [PMon1] [PMon2] [PBeu] PIM employs an algorithm in which the exact sampling of the quantum thermal Wigner density is combined with a linearized approximation of the quantum time propagators represented in the path integral formalism that reduces the evolution to classical dynamics. The quantities of interest can then be computed by combining classical molecular dynamics algorithms with a generalized Monte Carlo sampling scheme for sampling of the quantum initial conditions.



Applications of the Module

The PaPIM code has been extensively used for the calculation of the CH₅⁺ system infrared absorption spectrum in the gas phase. These calculations also provided the benchmark of the PIM method as well as for the code performance analysis. The results obtained on the CH₅⁺ system are currently under preparation for publication. One master thesis was completed by applying the code. Investigations of the processes shaping the infrared spectrum of small water cluster systems and a protonated water dimer system are currently being investigated using the PaPIM code.

Compiling

Fortran compiler with a MPI wrapper together with lapack libraries have to be available to successfully compile the code. The user is advised to examine the Makefile in the `./source` sub-directory prior to code compilation in

order to select an appropriate compiler and to check or adapt the compiler options to his local environment, or to generally modify the compiler options to his requirements.

Upon adapting the `Makefile`, the code compilation is executed by command `make` in the `./source` sub-directory. The executable `PaPIM.exe` is created at the same location upon successful compilation.

For module's testing purposes the user is advise to have `numdiff` package installed before running the tests. More details on the `numdiff` program package and its installation is available [here](#).

For compiling the PaPIM with an interface to the CP2K package [see here](#).

The PaPIM documentation is generated by executing the `make` command in the `./doc` sub-directory.

Testing

Testing check and validates the successful compilation of the PaPIM code. Tests and all corresponding reference files are located in sub-directory `./tests`. If using the automated testing script all tests have to be performed in this sub-directory.

The tests are performed on three systems, the OH, CH₄ and CH₅⁺. They are located in their corresponding sub-directories, `oh`, `ch4` and `ch5`, where each sub-directory contains corresponding classical and quantum input files located in `CLASSICAL` and `QUANTUM` sub-directories, respectively.

Tests are performed automatically in the `./tests` sub-directory by executing the command:

```
./test.sh -n [number of cores]
```

Flag `-n [number of cores]` controls the number of processor cores used in the tests. By default the tests are performed on two processor cores, which can be changed by setting a different number of required processor cores. Because the number of trajectories for sampling in certain tests are limited to 20, the number of processor cores should not exceed 20.

For testing of the PaPIM code linked with the CP2K package [see here](#).

For comparison of generated output values with reference data the test script uses `numdiff` command in order to compensate for small numerical differences. By default the script looks first for the `numdiff` command on the system, and in case it fails to locate it, the standard `diff` command will be used instead. However, the user is warned that due to small numerical differences between generated output and corresponding reference values the automated tests are most likely to fail. A local `numdiff` package copy can be included in the test by specifying its absolute path. For this and other options of the test script list them with the command `./test.sh -h`.

Performance and Benchmarking

PaPIM is designed as a highly scalable code. Its performance was extensively tested.

PaPIM code performance analysis

Two independent performance analyses of the PaPIM code are reported below. They were conducted by Dr. Liang Liang ([Maison de la Simulation](#) and [IDRIS](#)), and by Dr. Alan O'Cais ([Juelich Supercomputing Center](#)), respectively.

A strong scaling analysis of the PaPIM code using the [Scalasca](#) analysis tools, and the internal PaPIM code calculation time outputs was conducted on the CH₅⁺ system and performed on the [JURECA](#) cluster at the [JSC](#). [Figure 1](#) displays the results of the performed tests.

A parallel efficiency test was made on the [JUQUEEN](#) cluster at [JSC](#) also using the CH₅⁺ system. Results are displayed in [Figure 2](#).

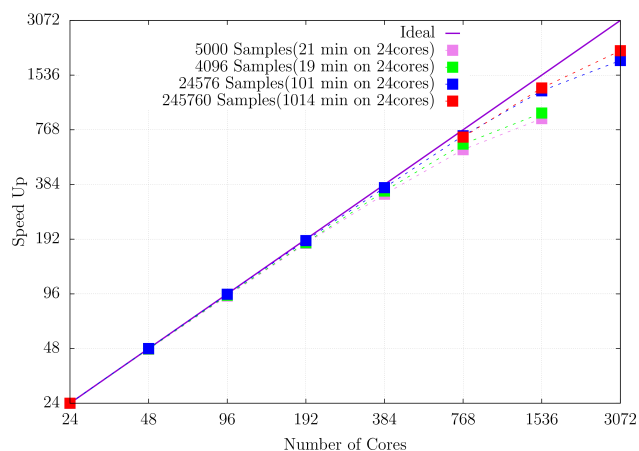


Fig. 3.1: Figure 1: PaPIM strong scaling test.

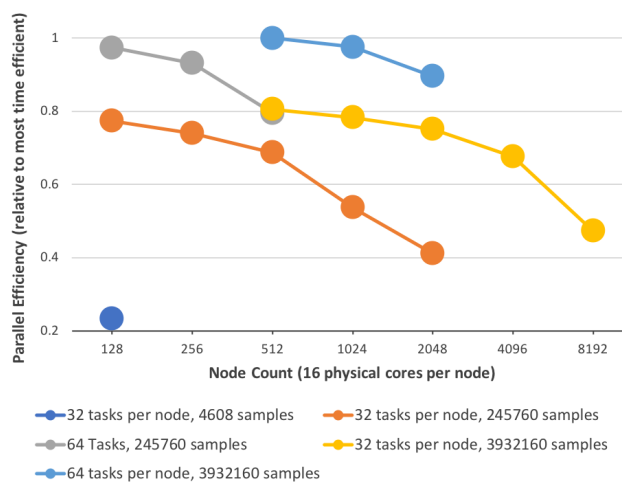


Fig. 3.2: Figure 2: PaPIM parallel efficiency test.

PaPIM scaling performance increases by increasing the number of sampling points.

Up to date the PaPIM code has been successfully run on 131,072 processor cores at JUQUEEN.

Source Code

The PaPIM module source code can be obtained from: <https://gitlab.e-cam2020.eu:10443/Quantum-Dynamics/PIM/tree/master/source>.

Source Code Documentation

The source code documentation is located in the `./doc` sub-directory. The documentation files (html and latex format) are generated by executing the `make` command in the `./doc` sub-directory.

References

PaPIM is the current version of the code, including all available functionalities.

The following modules make up the PaPIM code and can be used as stand-alone software libraries for e.g. sampling of the Wigner distribution, sampling of the classical Boltzmann distribution, or building MPI parallelized Fortran codes. Such libraries are rarely available to the community in a Fortran program format. Some of the functionalities within the code are specifically designed for computation of infrared spectra, and serve as a template for the user to implement its own functionalities.

PIM_wd

Software Technical Information

Language Fortran 90/95

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Momir Mališ

- *Purpose of Module*
- *Applications of the Module*
- *Compiling*
- *Testing*
- *Source Code*
- *Source Code Documentation*
- *References*

Purpose of Module

Module **PIM_wd** implements the Phase Integration Method (PIM) [Mon1] [Mon2] for the exact sampling of the quantum Wigner distribution in phase space representation. The PIM samples the thermal Wigner density using a generalized Monte Carlo scheme for sampling phase space points. The scheme combines the Penalty [Pen] and Kennedy [Ken] algorithms to sample noisy probability densities. This is necessary because the estimator of the quantum thermal density is not known analytically but must be computed via a statistical average affected by uncertainty. The sampled points are the basis for the calculation of time-dependent correlation function with the PIM algorithm via the module *PaPIM*. The user is required to provide the potential energy of the system by incorporating an external potential energy subroutine into the *PotMod* potential energy library.

Applications of the Module

This module forms the basis for computing the time-dependent cross- and auto-correlation functions with the PIM algorithm. It has been used in the calculation of CH_5^+ infrared spectrum and in the gas phase as well as for the computation of infrared spectrum of small water molecule clusters and protonated water dimer system.

Compiling

Fortran compiler with a MPI wrapper together with `lapack` libraries have to be available to successfully compile the code. The user is advised to examine the `Makefile` in the `./source`` sub-directory prior to code compilation in order to select an appropriate compiler and to check or adapt the compiler options to his local environment, or to generally modify the compiler options to his requirements.

```
cd source  
  
make
```

Upon adapting the `Makefile`, the code compilation is executed by command `make` in the `./source` sub-directory. An executable `PaPIM.exe` is created upon successful compilation.

Testing

For `PIM_wd` test purposes the `numdiff` package is used for automatic comparison purposes and should be made available before running the tests, otherwise the `diff` command will be used automatically instead but the user is warned that the test might fail due to numerical differences. The user is advised to download and install `numdiff` from [here](#). Tests and corresponding reference values are located in sub-directories `./tests/`. The tests are performed over three systems, the OH, CH_4 and CH_5^+ . They are located in their corresponding `oh`, `ch4` and `ch5`, where each sub-directory contains corresponding classical and quantum input files located in `CLASSICAL` and `QUANTUM` sub-directories, respectively. Before running the tests the code has to be properly compiled by running the `make` command in the `./source` sub-directory. The tests are performed automatically by executing the command `./test.sh` in the `./tests` sub-directory for all three systems:

```
cd tests  
  
./test.sh [number of cores]
```

Tests are by default performed using two processor cores, which can be changed by setting the value of required cores as an integer number after the command `./test.sh` (example `./test.sh 20`, for the use of 20 processor cores in the test). The number of processor cores should not exceed 20. Due to small numerical discrepancies between generated outputs and reference values which can cause the tests to fail, the user is advised to manually examine the numerical differences between generated output and the corresponding reference values in case the tests fail.

Source Code

The PIM_qcf module source code is located at: https://gitlab.e-cam2020.eu:10443/Quantum-Dynamics/PIM/tree/PIM_wd.

Source Code Documentation

The source code documentation can be generated automatically in `./doc` sub-directory, html and latex format, by executing the following command in the `./doc` directory:

```
doxygen PIMwd_doxygen_settings
```

References

PIM_wd samples, via the Phase Integration Method, [Mon1] the system's quantum Wigner density function. The function is given in the phase-space representation and is the basis for any further calculation of system's quantum observables.

PIM_qcf

Software Technical Information

Language Fortran 90/95

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Momir Mališ

- *Purpose of Module*
- *Applications of the module*
- *Compiling*
- *Testing*
- *Source Code*
- *Source Code Documentation*

Purpose of Module

Module **PIM_qcf** is a library of quantum cross- and auto-correlation functions used for computation of quantum time-dependent correlation functions within the Phase Integration Method (PIM). Two auto-correlation functions are currently implemented, the quantum position-position point charge dipole moment correlation function, and the velocity-velocity point charge dipole moment correlation function, all in the Kubo representation of the correlation functions. The user can follow the two examples to construct his/her own quantum correlation function.

Applications of the module

This module has been used in the calculation of CH_5^+ infrared spectrum in the gas phase as well as for the computation of infrared spectrum of small water molecule clusters and protonated water dimer system.

Compiling

A Fortran 90/95 compiler with MPI wrapper is required for successful compilation of the code. Although the correlation function subroutines are serial, the remaining code is parallelized so MPI wrappers have to be used. Quantum correlation subroutines within PIM_qcf modules are compiled by executing the command `make` in the `./source` directory. The same make command generates a `RunPIMqcf.exe` executable for testing of the correlation functions.

Testing

For PIM_qcf test purposes the `numdiff` package is used for automatic comparison purposes and should be made available before running the tests, otherwise the `diff` command will be used automatically instead but the user is warned that the test might fail due to numerical differences. The user is advised to download and install `numdiff` from [here](#). Tests and corresponding reference values are located in sub-directories `./tests/xxx`, where `xxx` stands for `oh`, `ch4`, and `ch5` systems. Before running the tests the code has to be properly compiled by running the `make` command in the `./source` sub-directory:

```
cd tests
./test.sh
```

Tests can be executed automatically by running the command `./test.sh` in the `./tests` sub-directory for all three systems, or separately for each system by running the command `./test.sh` within the corresponding sub-directory. All test are executed on one processor core. Due to small numerical discrepancies between generated outputs and reference values which can cause the tests to fail, the user is advised to manually examine the numerical differences between generated output and the corresponding reference values in case the tests fail.

Source Code

The PIM_qcf module source code is located at: <https://gitlab.e-cam2020.eu:10443/Quantum-Dynamics/PIM/tree/PIMqcf>.

Source Code Documentation

The source code documentation can be generated automatically in `./doc` sub-directory, html and latex format, by executing the following command in the `./doc` directory:

```
doxygen PIMqcf_doxygen_settings
```

PIM_qcf is a library of quantum correlation functions for computing system's time-dependent properties.

PIM_qtb

Software Technical Information**Language** Fortran 90/95**Licence** MIT license (MIT)**Documentation Tool** Sphinx Doxygen

- *Purpose of Module*
- *Description of the module*
 - *Classical Langevin dynamics*
 - *Quantum Thermal Bath (QTB)*
 - * *Adaptive Quantum Thermal Bath (adQTB-r/f)*
- *Input file*
- *Output files*
 - *Langevin trajectories*
 - *QTB analysis files*
- *Tests on implemented potentials*
 - *OH anharmonic potential*
 - *Lennard-Jones Ne₁₃ cluster*
- *Implementation*
 - *Source files*
 - *Other modifications*
- *Compiling*
- *Testing*
- *Source Code*
- *Source Code Documentation*
- *References*

Purpose of Module

Module **PIM_qtb** generates trajectories according to one of the following stochastic methods:

- Classical Langevin dynamics
- Quantum Thermal Bath [[Dam](#)]
- Adaptive Quantum Thermal Bath [[Man](#)]

These trajectories can be used to sample initial conditions for Linearized Semi-Classical Initial Value Representation (LSC-IVR) calculations.

Description of the module

The module implements different methods based on Langevin dynamics. The trajectories generated can be exploited directly or used to sample initial conditions for LSC-IVR calculations. The methods implemented are: classical Langevin dynamics, Quantum Thermal Bath (QTB) and two variants of adaptive QTB (adQTB-r and adQTB-f).

Classical Langevin dynamics

Classical Langevin dynamics is described by a stochastic differential equation :

$$\dot{p} = -\nabla U - \gamma p + F(t) \quad (3.1)$$

where p is the momentum vector of the set of atoms, interacting via the potential U , γ is the damping coefficient and $F(t)$ is the stochastic force. The random force $F(t)$ is a Gaussian white noise: to enforce the classical fluctuation-dissipation theorem, its autocorrelation spectrum is given by :

$$C_{FF}(\omega) = \int_{-\infty}^{+\infty} dt \langle F(t)F(t+\tau) \rangle e^{-i\omega t} = 2m\gamma k_B T$$

where k_B is the Boltzmann constant and T the temperature.

Quantum Thermal Bath (QTB)

The Quantum Thermal Bath uses a generalized Langevin equation in order to approximate nuclear quantum effects [Dam]. In QTB dynamics, the stochastic force is no longer a white noise but is colored according to the following formula :

$$C_{FF}(\omega) = 2m\gamma\theta(\omega, T) \quad (3.2)$$

with

$$\theta(\omega, T) = \hbar\omega \left[\frac{1}{2} + \frac{1}{e^{\hbar\beta\omega} - 1} \right]$$

where $\beta = \frac{1}{k_B T}$ and $2\pi\hbar$ is the Planck constant. The function $\theta(\omega, T)$ describes the energy of a quantum harmonic oscillators of angular frequency ω at a temperature T . The colored random force allows approximating zero-point energy contributions to the equilibrium properties of the system. The QTB method is known to lead to qualitatively good results [Bri] but as many semi-classical methods, it suffers from zero-point energy leakage (ZPEL) [Hern].

Adaptive Quantum Thermal Bath (adQTB-r/f)

The Adaptive Quantum Thermal Bath is an extension of the QTB method, designed to eliminate the zero-point energy leakage by enforcing the energy distribution prescribed by the quantum fluctuation-dissipation theorem for each degree of freedom and each frequency, all along the trajectories [Man].

In practice, this is done by minimizing the fluctuation-dissipation spectrum Δ_{FDT} defined as:

$$\Delta_{FDT}(\omega) = \text{Re} [C_{vF}(\omega)] - m\gamma_r(\omega)C_{vv}(\omega) \quad (3.3)$$

where C_{vF} is the velocity random force cross-correlation spectrum, C_{vv} the velocity autocorrelation spectrum and γ_r a set of damping coefficients dependent (or not) on the frequency.

This minimization is carried out on the fly during the QTB simulation by dissymetrizing the system-bath coupling coefficients corresponding to the damping force (dissipation) and to the random force (energy injection). This can be done either by directly modifying the random force spectrum $F(t)$ with frequency dependent damping term $\gamma_r(\omega)$ (adQTB-r variant) or by modifying the memory kernel of the dissipative force $\gamma_f(\omega)$ within the framework of a non-Markovian generalized Langevin equation (adQTB-f variant).

The coefficients γ_r or γ_f are slowly adjusted with a first-order differential equation and an adaptation coefficient A_γ :

$$\frac{d}{dt}\gamma_{r/f}(\omega) \propto A_\gamma \gamma \Delta_{FDT,r/f}(\omega, t) \quad (3.4)$$

during a preliminary “adaptation time” until they reach convergence. Observables are then computed while the adaptive process is kept active. Further information and precise implementation details can be found in ref. [Man].

Two implementations are currently available in PaPIM:

1. Random force adaptive QTB (adQTB-r):

In this variant, the dissipation kernel is left unchanged, i.e. $\gamma_f(\omega) = \gamma$ while the random force is modified according to a frequency-dependent set of damping coefficients $\gamma_r(\omega)$ to satisfy $\Delta_{FDT} = 0$ (eq. (3.3)):

$$C_{FF}(\omega) = 2m\gamma_r(\omega)\theta(\omega, T) \quad (3.5)$$

This method is applicable only if the initial damping coefficient γ is large enough to compensate effects of a possible zero-point energy leakage.

2. Dissipative kernel adaptive QTB (adQTB-f)

In this approach, the random force is not modified, i.e. $\gamma_r(\omega) = \gamma$ and remains the same as in the standard QTB method (eq. (3.2)) but the dissipation term is not described by a viscous damping term anymore ($-m\gamma v$) but corresponds to a non-Markovian dissipative force. This leads to the following generalized Langevin equation:

$$\dot{p} = -\nabla U - \int_0^\infty \gamma_f(\tau)p(t-\tau) d\tau + F(t) \quad (3.6)$$

In order to avoid solving with brute force this integro-differential equation, the dissipative memory kernel is expressed as a sum of equally spaced ($\Delta\omega$) lorentzian functions of width α :

$$\gamma_f(\omega) = \frac{\Delta\omega}{\pi} \sum_{j=0}^{n_\omega} \frac{\gamma_{f,j}}{\alpha + i(\omega - \omega_j)} + \frac{\gamma_{f,j}}{\alpha + i(\omega + \omega_j)} \quad (3.7)$$

The parameter $\gamma_{f,j}$ are then modified to satisfy $\Delta_{FDT} = 0$ (eq. (3.3)).

Input file

To run PaPIM using one of the Langevin methods, one must set the parameter *sampling_type* in the *sampling* section to one of the following values:

- classical_langevin
- qtb
- adqtr
- adqtf

In this case the parameters *n_equilibration_steps* and *n_mc_steps* are ignored and the section *langevin* is read.

The section *langevin* must specify the following parameters:

- *dt* : time step of the Langevin dynamics (REAL)
- *lgv_nsteps* : number of Langevin steps between each IVR sample (INTEGER)
- *lgv_nsteps_therm* : number of thermalization steps (INTEGER)
- *integrator* : integration method (two splitting methods are currently implemented: BAOAB, ABOBA (see reference [Lei])) (STRING, default="ABOBA")
- *damping* : base damping coefficient for production runs (γ in eq. :eq:eqLGV) (REAL)
- *damping_therm* : base damping coefficient for thermalization (γ in eq. :eq:eqLGV) (REAL)
- *qtb_frequency_cutoff* : cutoff frequency for the QTB kernel (REAL)
- *adqtb_agammas* : (Only for adqtbr and adqtb) adaptation speed coefficient for adQTB (A_γ in eq. (3.4))(REAL)
- *adqtb_alpha* : (Only for adqtb) Width of the lorentzian used to represent the dissipative kernel $\gamma_f(\omega)$ (α in eq. (3.7)) (REAL)
- *write_spectra* : write average random force autocorrelation function ff, velocity autocorrelation function vv and velocity random force cross-correlation function vf spectra (LOGICAL, default=.FALSE.)
- *write_trajectories* : write Langevin trajectories in x,y,z,px,py,pz format (LOGICAL, default=.FALSE.)

Remark: all physical quantities are specified in Hartree atomic units.

Output files

The Langevin module is plugged to the IVR subroutines and thus can output the same correlation functions as the classical MC sampling. Additionally, it can write the Langevin trajectories and spectra obtained directly from them.

Langevin trajectories

If the parameter *write_trajectories* of the *langevin* section of the input file is set to TRUE, Langevin trajectories are saved. Trajectory files follow the following format:

```
num_of_atoms
  At_symbol(1)  X   Y   Z   Px  Py  Pz
  At_symbol(2)  X   Y   Z   Px  Py  Pz
  .
  .
  At_symbol(n)  X   Y   Z   Px  Py  Pz
num_of_atoms
  At_symbol(1)  X   Y   Z   Px  Py  Pz
  At_symbol(2)  X   Y   Z   Px  Py  Pz
  .
  .
  At_symbol(n)  X   Y   Z   Px  Py  Pz
.
.
.
```

This corresponds to an extended XYZ format with information on momenta. It is readable by visualization software such as VMD to display the trajectories.

The module outputs multiple trajectory files depending on the number of independent trajectories (blocks) and the number of MPI processes. The naming follows the rules:

- `xp.traj.xyz` for 1 block and 1 process
- `xp_proci.traj.xyz` for 1 block and multiple processes
- `xp_proci_blockj.traj.xyz` for multiple blocks and processes

QTB analysis files

In addition to the trajectories, several files can be edited during the simulations. They are useful to carefully check the convergence of the adaptive QTB, notably by calculating $\Delta_{FDT}(\omega)$ (eq. (3.3)).

- `ff_vv_vf_spectra.out` spectra of random force and velocity autocorrelation and random force velocity cross-correlation functions (in atomic units)
 $\omega \ C_{FF}(\omega) \ 2m\gamma\theta(\omega, T) \ C_{vv}(\omega) \ m\gamma C_{vv}(\omega) \ C_{vF}(\omega)$
- `gamas.out` (for adQTB-r and adQTB-f only) final set of $\gamma_{r/f}(\omega)$ optimized during the adaptive procedure (in atomic units)
 $\omega \ \gamma_{r/f}(\omega) \ \gamma$

Tests on implemented potentials

OH anharmonic potential

The classical Langevin has been tested on the OH anharmonic potential. The left panel of Figure Fig. 3.3 shows time correlation functions obtained with IVR using initial conditions sampled from classical (Boltzmann) Monte Carlo and from classical Langevin. Its right panel shows the corresponding spectra obtained by Fourier transform.

Lennard-Jones Ne_{13} cluster

A Lennard-Jones potential has been implemented in `LennardJonesPot.f90` with the following pair potential:

$$V(r_{ij}) = \sum_{i=1}^N \sum_{j>i}^N 4\epsilon \left(\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 \right) \quad (3.8)$$

A confining pair potential (useful in the cases of small clusters) can be added to eq. (3.8). A 4th order polynomial is used for distances greater than a chosen distance r_{cont} :

$$V_{conf}(r_{ij}) = \sum_{i=1}^N \sum_{j>i}^N \epsilon (r_{ij} - r_{cont})^4 \quad (3.9)$$

Parameters for this potential are specified in an external text file. The file name is given in the input file using the parameter `lennard_jones_parameters` in section `system`. The parameters to specify are:

- `epsil` : depth of the potential well ϵ (in Kelvin) (eq. (3.8))
- `sigma` : distance for which the potential cancels σ (in Å) (eq. (3.8))
- `r_cont` : minimum distance for which a confining potential r_{cont} defined in eq. (3.9) is applied (in Å)

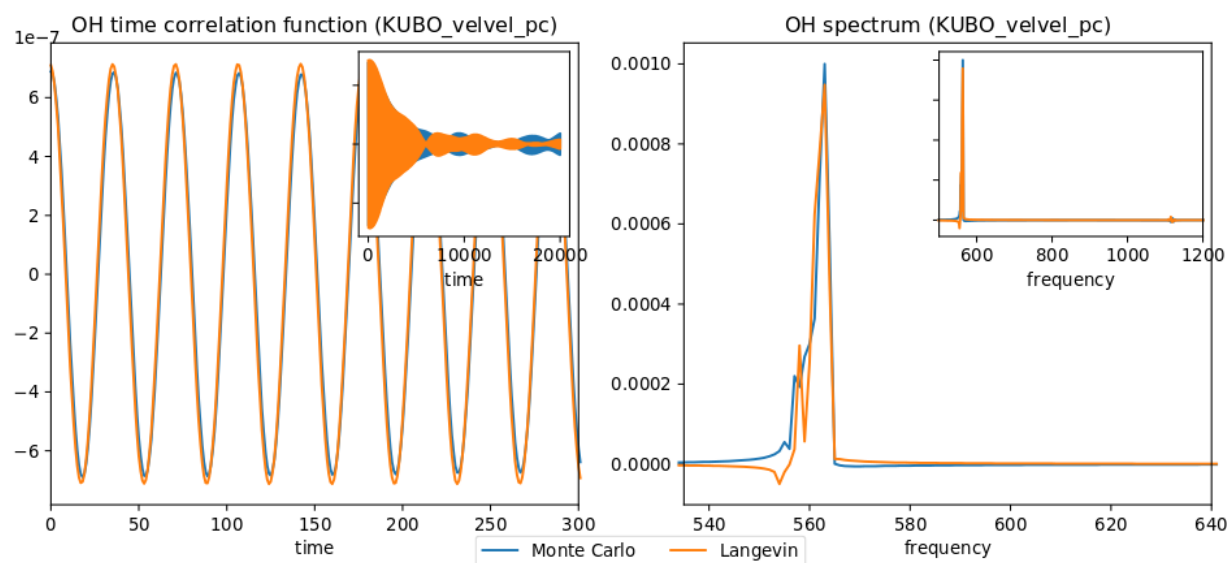


Fig. 3.3: Left panel: OH time correlation function using IVR with initial conditions sampled from MC and from Langevin. Right panel: corresponding spectra obtained by FFT.

The QTB and both adaptive methods were tested on a Ne_{13} cluster in order to reproduce results from reference [Man]. The Lennard-Jones parameters which have been used are $\epsilon = 34.9$, $\sigma = 2.78$ and $r_{\text{cont}} = 10$. 5 runs of 8000 steps with 16000 initial time steps are used with all four methods (Langevin, QTB, adQTB-r, adQTB-f). Damping term is set to 5.0×10^{-5} atomic units and adaptive coefficients A_γ and α for adQTB-f to 5.0×10^{-6} atomic units. Pair correlation function is then computed from the trajectories output with a Python script `compute_g2r.py`. Results are shown in figure Fig. 3.4 and are in agreement with the ones of ref. [Man].

In this particular case, adaptive QTB leads to significantly better results than both classical Langevin and QTB when comparing them to the reference results obtained with PIMD (Path Integral Molecular Dynamics).

Implementation

Langevin module is built with the fewest modifications possible in the main and previous code of PaPIM. The main program of the sampler is in the file `langevin.f90`. It is structured in the same fashion as the existing samplers (`PIM.f90` and `ClassMC.f90`) and only provides the subroutine `langevin_sampling` to the main program.

Source files

The Langevin module is divided in multiple files:

- `langevin.f90`: contains the Langevin sampler and links the main code with the other files of the module
- `langevin_integrator.f90`: subroutines to integrate Langevin equations
- `langevin_analysis.f90`: spectral analysis tools for Langevin and (ad)QTB trajectories
- `qtb_random.f90`: generation of QTB colored noise and adaptation subroutines for adQTB

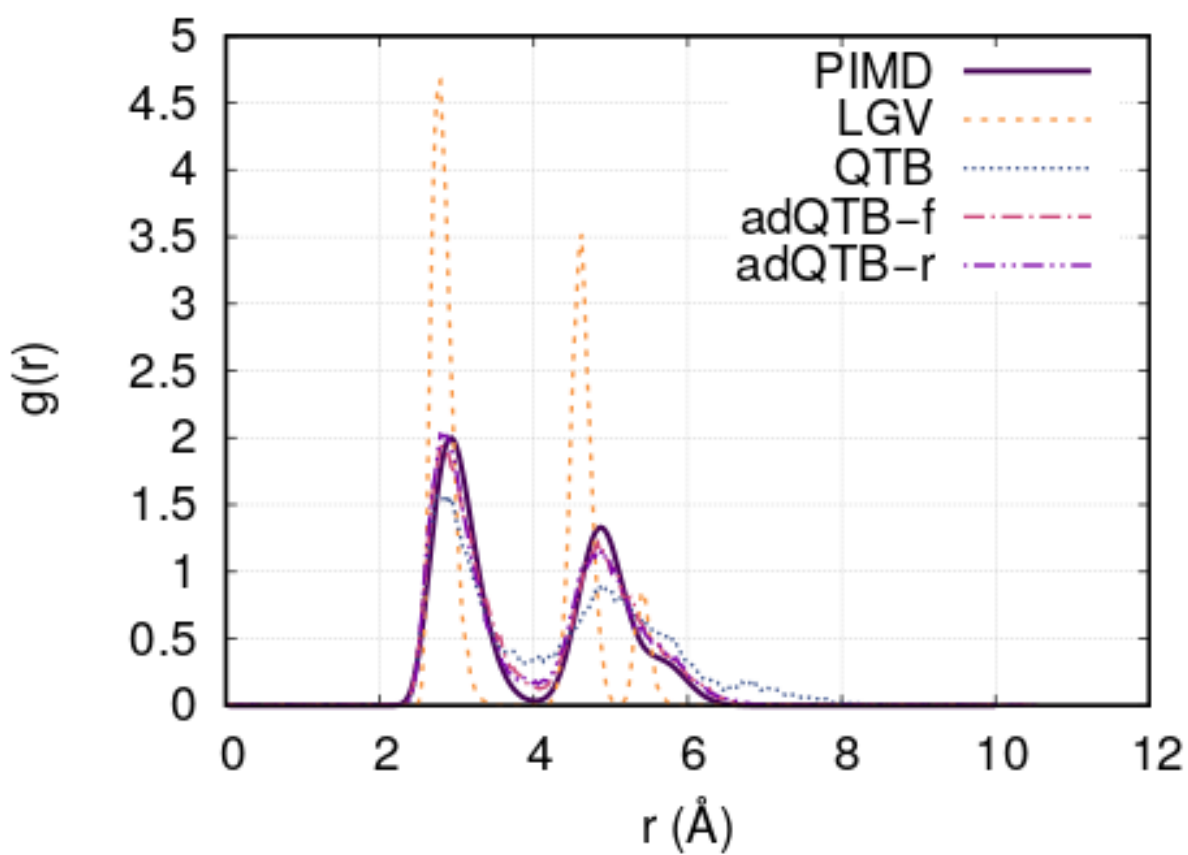


Fig. 3.4: Pair correlation function of Ne_{13} cluster obtained with Langevin, QTB, adQTB-r and adQTB-f implemented with Langevin module in PaPIM. Reference curve calculated with Path Integral Molecular Dynamics (PIMD)

Other modifications

Some other routines have been modified during the implementation of Langevin module.

- `PaPIM.f90`: main code ; add calls to Langevin module
- `GlobType.f90`: add declarations for Langevin
- `ReadFiles.f90`: read input files

Compiling

A Fortran 90/95 compiler with MPI wrapper is required for successful compilation of the code. Although the correlation function subroutines are serial, the remaining code is parallelized so MPI wrappers have to be used. The code must be compiled using the FFTW library. Quantum correlation subroutines within `PIM_qtb` modules are compiled by executing the command `make` in the `./source` directory. The same `make` command generates a `PaPIM.exe` executable for testing of the correlation functions.

Testing

For `PIM_qtb` test purposes the `numdiff` package is used for automatic comparison purposes and should be made available before running the tests, otherwise the `diff` command will be used automatically instead but the user is warned that the test might fail due to numerical differences. The user is advised to download and install `numdiff` from [here](#). Tests and corresponding reference values are located in sub-directories `./tests/xxx`, where `xxx` stands for `oh` and `lj` systems. `lj` tests also requires a Python distribution. Before running the tests the code has to be properly compiled by running the `make` command in the `./source` sub-directory:

Tests can be executed automatically by running the command in the `./tests` sub-directory : `#. ./test+lgv.sh` for tests on OH bonds compared to previous classical implementation `#. ./test_lj.sh` for tests on a Ne:. All test are executed on one processor core. Due to small numerical discrepancies between generated outputs and reference values which can cause the tests to fail, the user is advised to manually examine the numerical differences between generated output and the corresponding reference values in case the tests fail.

Source Code

The `PIM_qtb` module source code is located at: <https://gitlab.e-cam2020.eu:10443/thomas.ple/PIM.git> (Temporary link).

Source Code Documentation

The documentation can also be compiled by executing the following commands in `./doc/QTB_doc` directory with “Sphinx” (documentation tool) python module installed:

```
sphinx-build -b html source build
make html
```

The source code documentation can be generated automatically in `./doc` sub-directory, html and latex format, by executing the following command in the `./doc` directory:

```
doxygen PIMqcf_doxygen_settings
```


References

PIM_qtb implements different methods based on Langevin dynamics. The trajectories generated can be exploited directly or used to sample initial conditions for Linearized Semi-Classical Initial Value Representation (LSC-IVR) calculations. The methods implemented are: classical Langevin dynamics, Quantum Thermal Bath (QTB) and two variants of adaptive QTB (adQTB-r and adQTB-f).

ClassMC

Software Technical Information

Language Fortran 95/90

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Momir Mališ

- *Purpose of Module*
- *Applications of the Module*
- *Compiling*
- *Testing*
- *Source Code*
- *Source Code Documentation*

Purpose of Module

Module **ClassMC** samples the system phase space using the classical Boltzmann distribution function and calculates the time correlation functions from the sampled initial conditions. The sampling is achieved by the Monte Carlo Metropolis algorithm. The corresponding system properties can be calculated from the sampled phase space with appropriate operators. The sampled phase space points can be propagated in time using classical molecular dynamics in order to investigate the time evolution of the system and calculate the corresponding correlation functions. Currently the electric dipole moment operator is implemented for the calculation of electric dipole moment autocorrelation functions from which system IR spectra can be directly obtained. The system potential energy is calculated using external subroutines provided by the user. Example external subroutines are provided for the OH and CH₄ systems, with potential energies are described by an harmonic potential, and the electric dipole moments by point charge approximation. An external subroutines for calculation of CH₅⁺ system potential energy and electric dipole moment, based on fitted values, is also given.

Applications of the Module

The main application of ClassMC code is classical sampling of the system's phase space and computing classical observables, which are necessary for comparison with the real experimental data or quantum simulations in order to detect and explain the, sometimes hardly detectable, quantum effects which are responsible for exact system properties. In this respect, the ClassMC module was extensively used in the study of the CH₅⁺ system classical distribution and its

classically obtained infrared spectrum in order to identify the quantum tunnelling effects responsible for the redshift of C-H stretching bands and the overall shape of the infrared spectrum.

Compiling

Fortran compiler with a MPI wrapper together with `lapack` libraries have to be available to successfully compile the code. The user is advised to examine the `Makefile` in the `./source`` sub-directory prior to code compilation in order to select an appropriate compiler and to check or adapt the compiler options to his local environment, or to generally modify the compiler options to his requirements. Upon adapting the `Makefile`, the code compilation is executed by command `make` in the `./source` sub-directory:

```
cd source
make
```

An executable `ClassMCRun.exe` is created upon successful compilation.

Testing

For ClassMC test purposes the `numdiff` package is used for automatic comparison purposes and should be made available before running the tests, otherwise the `diff` command will be used automatically instead but the user is warned that the test might fail due to numerical differences. The user is advised to download and install `numdiff` from [here](#). Tests and corresponding reference values are located in sub-directories `./tests/xxx/CLASSICAL`, where `xxx` stands for `oh`, `ch4`, and `ch5` systems. Before running the tests the module ClassMC has to be properly compiled by running the `make` command in the `./source` sub-directory. Tests can be executed automatically by running the command `./test.sh` in the `./tests` sub-directory for all three systems, or separately for each system by running the command `./test.sh` within the corresponding system CLASSICAL sub-directory:

```
cd tests
./test.sh [number of cores]
```

Tests are by default executed on two processor cores. This can be changed by setting the value of required cores as an integer number after the command `./test.sh` (example `./test.sh 20`, for the use of 20 processor cores in the test). The number of processor cores should not exceed 50. Due to small numerical discrepancies between generated outputs and reference values which can cause the tests to fail, the user is advised to manually examine the numerical differences between generated output and the corresponding reference values in case the tests fail.

Source Code

The ClassMC module source code is located at: <https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/ClassMC>.

Source Code Documentation

The source code documentation is given at <https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/ClassMC/doc>. The documentation files (html and latex format) are obtained by executing the `make` command in the `./doc` directory:

```
cd ./doc
make
```

ClassMC samples, via Metropolis Monte Carlo algorithm, the system's classical Boltzmann distribution function and calculates the classical time-dependent correlation functions from the sampled phase space. Results obtained from classical sampling can be used to assess the relevance of quantum effects for a given system.

PotMod

Software Technical Information

Language Fortran 95/90

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Momir Mališ

- *Purpose of Module*
- *Applications of the Module*
- *Compiling*
- *Testing*
- *Source Code*
- *Source Code Documentation*
- *References*

Purpose of Module

Module **PotMod** is a library of potential energy subroutines and interfaces to external potential energy calculation codes. It provides potential energies and corresponding gradients for included potentials or calls an external code to compute the required quantities. Currently, two subroutines are implemented within this module. A subroutine for the calculation of harmonic and Morse potential energies which requires a set of input parameters provided as an external file, and a subroutine containing the analytic ground state electronic energy for the CH_5^+ system. [Jin]

Applications of the Module

This module is extensively used by the *PaPIM* code and *PIM_wd* and *ClassMC* modules for providing the necessary potentials and gradients of studied systems.

Compiling

The code should be compiled in the `./source` sub-directory using a Fortran compiler. A Makefile is present for an automatic compilation. Execute command 'make' in the `./source` sub-directory to generate the `PotModRun.exe` executable:

```
cd source
make
```

Testing

For PotMod test purposes the `numdiff` package is used for automatic comparison purposes and should be made available before running the tests, otherwise the `diff` command will be used automatically instead but the user is warned that the test might fail due to numerical differences. The user is advised to download and install `numdiff` from [here](#). The module is accompanied by a corresponding Fortran 90 test subroutine and a reference output. The reference output is located in sub-directory `./tests/REFERENCE_VALUE`. The test can be executed automatically by running the script `./test.sh`:

```
cd tests
./test.sh
```

or manually by executing the compiled `PotModRun.exe` code within the sub-directory `./tests` (example `./source/PotModRun.exe > out`) and comparing the output with the reference values in file `REFERENCE_VALUE/tested_potentials`.

Source Code

The source code is given at <https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/PotMod>. File `harmonic_potential.f90` contains the subroutines for harmonic and Morse potential energy calculations, while file `ch5_pes.f90` contains the subroutines for calculation of CH_5^+ potential energy. File `PotMod.f90` controls and calls the included subroutines (`harmonic_potential.f90` and `ch5_pes.f90`). The remaining subroutines (`GlobType.f90`, `kinds.f90`, `ReadFiles.f90`, `PotModRun.f90`) are subroutines for test purposes, where `GlobType.f90` contains the definition of derived types used by PotMod module.

Source Code Documentation

The source code documentation is given at <https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/PotMod/doc/>. The documentation files (html and latex format) are obtained by executing the `make` command in the `./doc` sub-directory:

```
cd doc
make
```

References

PotMod is a library of potential energy functions and interfaces for external potential energy calculation codes. Currently available in the library are the harmonic and Morse potentials (different molecular systems can be simulated depending on parameters provided by the user); empirical potential of the ground state of CH_5^+ based on high level electronic structure calculations [ZJin]; and the call to the ab initio **CP2K** code using the **PaPIM-CP2K_Interface** module.

PaPIM-CP2K_Interface

Software Technical Information

Language Fortran 90/95

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Momir Mališ, Ari P. Seitsonen

- *Purpose of Module*
- *Applications of the Module*
- *Compiling*
- *Testing*
- *Source Code*
- *Parallelization scheme*
- *Source Code Documentation*

Purpose of Module

Module **PaPIM-CP2K_Interface** couples the *PaPIM code* with the *CP2K program package*, where the latter is used for calculation of system electronic structure properties. It directly links CP2K as a library for potential energy calculations to the PaPIM code and avoids the significantly slower exchange of information between the two codes by reading and writing to an external file. The CP2K program package provides a general framework for different modeling methods such as DFT using the mixed Gaussian and plane waves approaches, semi-empirical methods and classical force fields. This enables virtually any calculation of time-dependent correlation functions for any system, without depending on the availability of analytical potentials for the studied system, as was the previous case in PaPIM code. Using the MPI split communicator approach the CP2K subroutines can be executed on multiple cores for each sampling trajectory, enabling a parallel calculation of system potential energy and gradient values.

Applications of the Module

The inclusion of CP2K for computation of system's electronic structure properties enables calculation of time-dependent correlation functions to a vast range of systems, while CP2K can perform atomistic simulations of solid state, liquid, molecular, periodic, material, crystal, and biological systems. The PaPIM code has also been upgraded with periodic boundary conditions to enable simulations of solid and liquid state systems. For any system whose properties can be determined with the CP2K code, a corresponding time-dependent correlation function can be computed now with the PaPIM code.

Compiling

In order to compile this module the CP2K program package has to be properly set-up and the CP2K has to be compiled as a library as well. In the latter case, the CP2K root directory contains a sub-directory `lib` which contains the corresponding library files. In the absence of the latter, CP2K cannot be linked to PaPIM code. For information on installing

the CP2K code and compiling it as a library the user is advised to examine the CP2K installation documentation at [this link](#).

Fortran compiler with a MPI wrapper together with lapack libraries have to be available to successfully compile the code. The user is advised to examine the `Makefile` in the `./source` sub-directory prior to code compilation in order to select an appropriate compiler and to check or adapt the compiler's options to his local environment, or to generally modify the compiler options to his requirements. Special care should be made on the CP2K paths to the corresponding library files on certain systems. The `Makefile` contains two example cases encountered on cluster systems (which use Intel compilers) but any other variation is possible.

The compilation flag `--D__USE_CP2K` controls the inclusion of CP2K into the PaPIM code. In the default compilation settings the flag is commented out. To include CP2K into compilation the user is required to enable the flag in the `Makefile`. If the flag is omitted the PaPIM code will be compiled without CP2K and without the split communicator parallelization scheme. The latter is not used with any current analytic potential subroutine so it is omitted (for more details on the potential subroutine see [here](#)). We advise to compile the PaPIM code first successfully (by verifying the compilation by executing and checking the standard tests) before recompiling and linking it to CP2K.

Upon adapting the `Makefile`, the code compilation is executed by command `make` in the `./source` sub-directory. The executable `PaPIM.exe` is created at the same location upon successful compilation.

For module's testing purposes the user is advised to have `numdiff` package installed before running the tests. More details on the `numdiff` program package and its installation are available [here](#).

The PaPIM documentation is generated by executing the `make` command in the `./doc` sub-directory.

Testing

Testing comprises of a set of tests for the analytic potentials and a set of tests for the CP2K interface to PaPIM code. Tests and all corresponding reference files are located in sub-directory `./tests`. All tests have to be performed in this sub-directory. Details of the structure of the test files/sub-directories are explained in the [PaPIM code testing section](#). CP2K tests and reference files are located in the sub-directory `cp2k`.

Tests are performed automatically in the `./tests` sub-directory by executing the command:

```
./test.sh -c -n [number of cores]
```

The optional flag `-c` includes running of the CP2K tests and should only be used if PaPIM code was compiled with CP2K. If omitted only the tests using the analytic potential will be performed. Flag `-x` omits the analytic potential tests and executes only the CP2K tests.

Flag `-n [number of cores]` controls the number of processor cores used in the tests. Omitting this flag the tests will be performed on two processor cores (default value). Because the number of trajectories for sampling in certain tests are limited to 20, the number of processor cores should not exceed 20. In all cases the CP2K potential is calculated on one processor core for each trajectory. In order to change that, a different `group_size` variable should be specified manually in the corresponding `CONTROL` files located in sub-directory of each test case. Note that the total number of processor cores used in the tests should be divisible by the `group_size` value (see details in Parallelization and Benchmarking section).

For comparison of generated output values with reference data the test script uses `numdiff` command in order to compensate for small numerical differences. By default the script looks first for the `numdiff` command on the system, and in case it fails to locate it, the standard `diff` command will be used instead. However, the user is warned that due to small numerical differences between generated output and corresponding reference values the automated tests are most likely to fail. A local `numdiff` package copy can be included in the test by specifying its absolute path. For this and other options of the test script list them with the command `./test.sh -h`.

Source Code

The full PaPIM code with the interface subroutine to the CP2K is located at: <https://gitlab.e-cam2020.eu:10443/Quantum-Dynamics/PIM/tree/master/source>.

Git is recommended for downloading the full copy of the code.

The main interface subroutines for linking PaPIM to CP2K are located in the Fortran module file `cp2k_module.f90`. Corresponding commands used throughout the code can be located by searching for the `__USE_CP2K` keyword.

Parallelization scheme

Parallelization of linked PaPIM and CP2K codes is achieved with a MPI split communicator approach. A separate communicator is given for the PaPIM code and for the CP2K part. The latter is split into groups, each of a number of processor cores given by the `group_size` value. Therefore, the number of trajectories which can be sampled simultaneously is given by the quotient of the total number of used processor cores with the value of the `group_size`. For the same reason the total number of cores must be divisible by the `group_size` value. The figure below explains in a simplified graphical manner the parallelization used in the PaPIM code linked to CP2K.

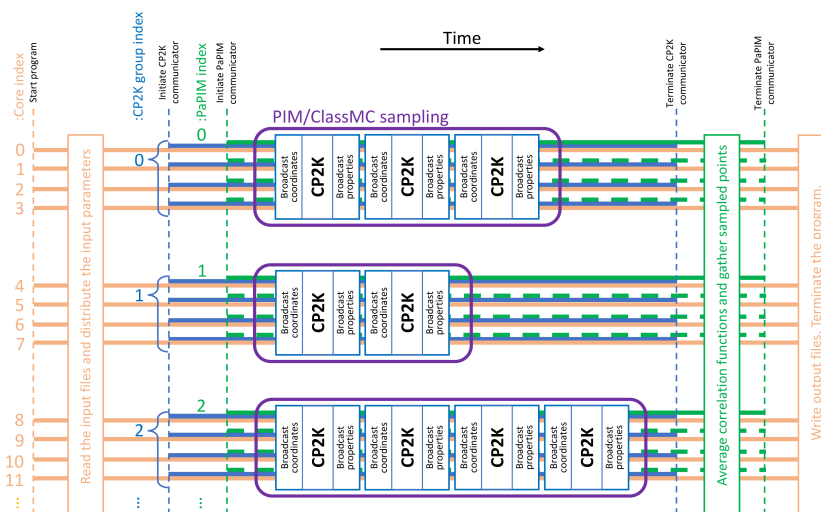


Fig. 3.5: Graphical representation of the MPI split communicator scheme used in parallelization of PaPIM-CP2K_interface module. An example with CP2K `group_size` of 4 is displayed.

Source Code Documentation

The source code documentation is located in the `./doc` sub-directory. The documentation files (html and latex format) are generated by executing the `make` command in the `./doc` sub-directory.

PaPIM-CP2K_Interface module links the PaPIM code with the **CP2K program package** as an internal library for calculation of system's electronic structure properties.

AuxMod

Software Technical Information

Language Fortran 95/90

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Ari P. Seitsonen, Momir Mališ

- *Purpose of Module*
- *Applications of the Module*
- *Compiling*
- *Testing*
- *Source Code*
- *Source Code Documentation*

Purpose of Module

Module **AuxMod** contains a set of subroutines which can be used for an easier construction of any program input file reader, and a library of common MPI commands adapted for easier implementation when programming a Fortran MPI parallel code. The module consists of an input parser designed to read any formatted file with the possibility to find a specific set of user pre-defined keywords and examine whether the read in variable types are consistent with the code requirements. The library of parallel subroutines contains a number of MPI commands for communicating information between all or a pair of processor cores, and are adapted for easier user implementation into his/her own code. The provided subroutines/libraries can also be considered as a Fortran template which the user can adapt or update depending on his/her specific requirements.

Applications of the Module

The AuxMod module was used to construct the input parser for the PaPIM code, while its modified MPI commands enable to parallelize the PaPIM code and the ClassMC module. Based on these example, the AuxMod provides a pre-constructed input reader and adapted MPI library for any future Fortran code development.

Compiling

The code should be compiled in the `./source` sub-directory using a Fortran compiler with a MPI wrapper. A Makefile is present for an automatic compilation. Execute command `make` in the `./source` sub-directory to generate the `AuxModRun.exe` executable:

```
cd source  
  
make
```


Testing

For AuxMod test purposes the `numdiff` package is used for automatic comparison purposes and should be made available before running the tests, otherwise the `diff` command will be used automatically instead but the user is warned that the test might fail due to numerical differences. The user is advised to download and install `numdiff` from [here](#). The module is accompanied with an example input file `TESTINPUT` located in the `tests` sub-directory together with the reference output in sub-directory `REFERENCE_OUTPUT`:

```
cd tests
../source/AuxModRun.exe < TESTINPUT
```

The user is also advised to test the code manually by changing the values in the `TESTINPUT` input file.

Source Code

The source code is given at <https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/AuxMod>. The file parser `.F90` contains all the subroutines for the within-one-line data type recognition, while the `auxmod.F90` contains the direct subroutines for input file reading, and can be considered as a template for further modification. The `pri.F90` contains the adapted MPI commands.

Source Code Documentation

The source code documentation is given at <https://gitlab.e-cam2020.eu:10443/Quantum-Dynamics/PIM/tree/AuxMod/doc>. The documentation files (html and latex format) are obtained by executing the `make` command in the `./doc` sub-directory:

```
cd doc
make
```

AuxMod is a library of subroutines which enables any user to easily construct its own Fortran input parser. It also contains a library of adapted MPI subroutines for easier programming of Fortran MPI parallel codes.

Openmpbeads

Software Technical Information

Language Fortran 90/95

Licence MIT license (MIT)

Documentation Tool Doxygen

Software Module Developed by Przemyslaw Juda, Momir Mališ

- *Purpose of Module*
- *Compiling*
- *Testing*

- [Source Code](#)
- [Source Code Documentation](#)

Purpose of Module

Sampling of quantum properties is performed via the so-called classical isomorphism of path integral. In this scheme, a quantum degree of freedom is mapped into a classical polymer with a certain number of beads. This number of beads increases with the relevance of quantum effects and can become very large. Because in the sampling procedure, generally, for each polymer bead a potential energy evaluation is required within a single sampling step, polymer sampling subroutines become a bottleneck. **Openmpbeads** is a patch to the [PaPIM](#) code which increases the code's performance by parallelizing the polymer chain sampling subroutines. The module introduces the OpenMP parallelization loops for the polymer sampling subroutines. In this way, a considerable increase of performance can be achieved.

Compiling

The [PaPIM](#) program source code (for PaPIM download see [here](#)) and [Git](#) should be available. The downloaded Openmpbeads patch should be placed in the PaPIM main directory, and applied to the PaPIM source code by executing the following command:

```
git apply openmpbeads.patch
```

After the patch has been successfully applied, the OpenMP parallelized PaPIM code can be re-compiled as described in the PaPIM [documentation](#).

Testing

The successful Openmpbeads patch application and compilation should be verified by executing the codes standard tests. The code's tests are located in the directory `./tests`. The same set of tests as for the verification of the PaPIM code is executed, but now with the addition of utilizing OpenMP parallelization. Thus a number of processor cores available for the test should be at least two. For details of the PaPIM code standard tests see [here](#). Before running the tests the code has to be properly compiled by running the `make` command in the `./source` sub-directory (see compilation of PaPIM code [here](#)). The `numdiff` package is used for automatic comparison purposes and should be made available before running the tests, otherwise the `diff` command will be used automatically instead but the user is warned that the test might fail due to small numerical differences. The tests are performed automatically by executing the following command in the `./tests` sub-directory:

```
cd tests

./test.sh -m [number of MPI cores] -o [number of OpenMP cores]
```

The `[number of MPI cores]` should not exceed 20, and the `[number of OpenMP cores]` should not exceed 5. The product of `[number of MPI cores]` and `[number of OpenMP cores]` should not exceed to the total number of available cores on the system and should also not exceed number 100. Due to small numerical discrepancies between generated outputs and reference values which can cause the tests to fail, the user is advised to manually examine the numerical differences between generated output and the corresponding reference values in case the tests fail.

Source Code

The Openmpbeads module patch is located at: <https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/openmpbeads>.

Source Code Documentation

The Openmpbeads patch also adds additional description to the PaPIM code's documentation. Details how to access and generate PaPIM documentation are given [here](#).

Openmpbeads is a patch to the PaPIM code which enables parallelization of the sampling of the polymer chains within the PIM algorithm, improving efficiency in sampling of the Wigner density.

PerGauss is an implementation of periodic boundary conditions for gaussian basis functions to be used within the quantics program package.

3.5.9 Quantics

Quantics is suite of programs for molecular quantum dynamics simulations. The package is able to set up and propagate a wavepacket using the MCTDH method [Beck]. Numerically exact propagation is also possible for small systems using a variety of standard integration schemes [Lefo], as is the solution of the time-independent Schrödinger equation using Lanczos diagonalisation. The program can also be used to generate a ground state wavefunction using energy relaxation (i.e. propagation in imaginary time) and with the “improved relaxation” it is even possible to generate (low lying) excited states. Within the Quantics package there are also programs to propagate density operators (by solving the Liouville-von Neumann equation for open or closed system) [Mey], a program for fitting complicated multi-dimensional potential energy function, programs for determining bound or resonance energies by filter-diagonalisation, parameters of a vibronic coupling Hamiltonian, and many more. Recent developments include the use of Gaussian wavepacket based methods (G-MCTDH) and interfaces to quantum chemistry programs such as Gaussian and Molpro allow direct dynamics calculations using the vMCG method [Ric]. The following modules are extension of Quantics functionalities developed at E-CAM Extended Software Development Workshops.

Second-Order Differencing Scheme (SOD)

Software Technical Information

Language Fortran 90

Licence None

Documentation Tool Documentation provided as in-line comments within the source code

Application Documentation Useful documentation can be found [here](#)

Relevant Training Material Training material is available through the test examples

Software Module Developed by Graham Worth, Kaite Spinlove, Marcus Taylor

- *Purpose of Module*
- *Background Information*
- *Testing*

- [Source Code](#)
- [References](#)

Purpose of Module

This module provides exact wavefunction propagation using the second-order differencing (SOD) integrator scheme to solve the time-dependent Schrödinger equation as described by Leforestier et al. [Lef] Within this scheme the time interval is determined through dividing \hbar by the eigenvalue of the Hamiltonian operator with the largest absolute value.

Background Information

Currently the SOD integration scheme resides within the [Quantics](#) software package available through [CCPForge](#).

Testing

A test example (`test90.inp`) is provided for the SOD integration scheme and can be found in the directory `~/quantics/elk_inputs`. This test works for [Quantics](#) Revision 787. The [Quantics](#) README file will help you to install the [Quantics](#) code. The test can be done through the following command

```
$ quantics test90.inp
```

A more detailed test documentation for [Quantics](#) code developers can be found in [this link](#)

Source Code

The source code for the second-order differencing propagator can be found within the [Quantics](#) software which can be downloaded via [gitlab](#). You firstly need to make an account (at [gitlab](#)). The [Quantics](#) project has a private repository so you also need to be a member of the project to clone it into your computer, then type:

```
git clone https://gitlab.com/quantics/quantics.git
```

Within the [Quantics](#) program, explicit code for the SOD routine is located in file `~/quantics/source/lib/ode/sodlib.f90`.

References

The **SodLib** module provides exact wavefunction propagation using the second-order differencing (SOD) integrator scheme to solve the time-dependent Schrödinger equation. This routine has been implemented and tested as an added functionality within the [Quantics](#) quantum dynamics package.

The Chebyshev Scheme (CH)

Software Technical Information

Language Fortran 90

Licence None

Documentation Tool Documentation provided as in-line comments within the source code

Application Documentation Useful documentation can be found [here](#)

Relevant Training Material Training material is available through the test examples

Software Module Developed by Graham Worth, Ceridwen Ash

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

Purpose of Module

This module implements the Chebyshev integration scheme for exact wavefunction propagation on the grid. This routine has been implemented and tested within the [Quantics](#) quantum dynamics package which is available on [CCPForge](#). The purpose of the module is to be used in quantum dynamical propagation problems as described by Leforestier et al. [[1Lef](#)]

Background Information

Currently, the Chebyshev integration scheme resides within the [Quantics](#) software package available through [CCPForge](#).

The module consists of a main routine, `chebstep`, which uses a separate routine `bessjn` to generate the real Bessel coefficients of integer order. Both are referenced and documented within the `cheblib` routine, and for further information on the Chebyshev integration scheme see Tal-Ezer et al. [[1Tal](#)]

Testing

A test example (`test89.inp`) is provided for the Chebyshev integration scheme and can be found in the directory `~/quantics/elk_inputs`. This test works for [Quantics](#) Revision 787. The [Quantics](#) README file will help you to install the [Quantics](#) code. The test can be done through the following command

```
$ quantics test89.inp
```

A more detailed test documentation for [Quantics](#) code developers can be found [in this link](#)

Source Code

The source code for the Chebyshev propagator can be found within the [Quantics](#) software which can be downloaded via [gitlab](#). You firstly need to make an account (at [gitlab](#)). The [Quantics](#) project has a private repository so you also need to be a member of the project to clone it into your computer, then type:

```
git clone https://gitlab.com/quantics/quantics.git
```

Within the **Quantics** program, explicit code for the Chebyshev routine is located in the file `~/quantics/source/lib/ode/chebplib.f90`.

References

The **ChebLib** module implements the Chebyshev integration scheme for exact wavefunction propagation on the grid. This routine has been implemented and tested as an added functionality within the **Quantics** quantum dynamics package.

Software Technical Information

Name Quantics

Language Fortran90

Licence GNU General Lesser Public License

Documentation Tool Documentation provided as in-line comments within the source code and in the Quantics online documentation

Application Documentation Useful documentation can be found [here](#)

Relevant Training Material Useful training can be found [here](#)

Software Developed by Johannes Ehrmaier, Graham Worth

QQ-Interface (Quantics-QChem-Interface)

- *Purpose of Module*
- *Applications*
- *Building and Testing*
- *Source Code*

Purpose of Module

The Quantics-Qchem-Interface module connects the full quantum nonadiabatic wavefunction propagation code Quantics to the time-dependent density functional (TDDFT) module of the electronic structure program QChem. QChem provides analytic gradients, Hessians and derivative couplings at TDDFT level. With this module it is possible to use the QChem TDDFT module for excited state direct dynamics calculations. Quantics will prepare the input file from a template, start QChem calculations whenever needed and will read the output of QChem. The QChem results are stored in the Quantics database and can be used in dynamics simulations. Due to the modular design of Quantics the TDDFT module of QChem can be used for all dynamics simulations, e.g. dd-vMCG (direct-dynamics with variational multi-configurational Gaussians) or surface hopping simulations.

Applications

The module will be used to examine the nonadiabatic excited state dynamics of small to medium sized molecules. The TDDFT module of QChem allows to treat systems that are too large for efficient CASSCF calculations. Until today

photoinduced dynamics simulations of such molecules were only possible using trajectory based algorithms. With Quantics a full quantum-mechanical description of the nuclear motion is possible.

Building and Testing

To use the module, get the latest version of Quantics from the repository and build it as usual. Moreover you have to have a running version of **QChem** installed on your system. An example calculation, simulating the photodissociation of water using 4 coupled states is added to the Quantics repository, the documentation of the example can be found [at this link](#). After you have copied the 'water.inp' and the 'run_qchem' files to your directory, you have specified the template for the electronic structure calculations and you performed the preparatory calculations, you can start the simulation with:

```
quantics -mnd water.inp
```

In the specific example, Quantics will search for a script called 'run_qchem' (specified in the input file) to start a QChem calculation. The file 'run_qchem' script is of course dependent on your system configuration and has to be adapted. For more information how to run the test simulation please refer to the Quantics documentation.

As vMCG dynamics is very sensitive to numerical issues it is possible, depending on your compiler and machine, that your results differ slightly from the provided reference values (in the order of a few percent), but the qualitative behavior of the results should be preserved.

Source Code

The source code for the QQ-Interface can be found within the Quantics software which can be downloaded via [gitlab](#). You firstly need to make an account (at gitlab). The Quantics project has a private repository so you also need to be a member of the project to clone it into your computer, then type:

```
git clone https://gitlab.com/quantics/quantics.git
```

Within the Quantics program, explicit code for the QQ-Interface routine is located in the file ~/quantics/source/opfuncs/funcqchemmod.f90. Most changes can be found in the subroutines 'ddqchem' and 'wrqchem'.

The **Quantics-QChem-Interface** is an interface between Quantics and **QChem**. The DFT algorithm implemented in QChem can be used to provide electronic structure information for direct dynamics simulations using the Quantics program package.

Zagreb surface hopping code

Software Technical Information

Language Fortran 2003

Licence GNU General Lesser Public License

Documentation Tool Documentation provided as in-line comments within the source code

Application Documentation Useful documentation can be found [here](#)

Relevant Training Material Training material is available through the test examples

Software Module Developed by Surface hopping code: Nadja Doslic, Marin Sapunar and Momir Malis. Module: Graham Worth, Cristina Sanz-Sanz.

- *Purpose of Module*
- *Background Information*
- *Application*
- *Testing*
- *Source Code*

Purpose of Module

This module implements an interface between the Tully's fewest switch surface hopping code, written and maintained by the group of Nadja Doslic in Zagreb, and Quantics code. The module has been added and tested within the Quantics quantum dynamics package which is available on Gitlab. The purpose of the module is to add the solution of Hamilton classical equation using the surface hopping approach into Quantics code, at the same time as Zagreb code benefits from all functionalities implemented in Quantics as input definitions, Hamiltonian operator description, direct dynamics calculations and parallel running.

Background Information

Currently, the Zagreb surface hopping quasiclassical trajectory code resides within the Quantics software package available through [gitlab](#).

The module consists of an interface between Quantics package and Zagreb surface hopping code. The module is fully integrated into Quantics code so that initial conditions, wavefunction definition, analysis programs, direct dynamics etc. . . can be used in the usual way described in Quantics documentation. The interface creates the required input files to run separate trajectories using the Zagreb surface hopping code. Although the module is implemented and run under Quantics, the Zagreb code requires some directives that must be given in the input file under the SH_ZAGREB_SECTION. This directives are described in the Zagreb code input manual which can be found in a separate pdf file in the Quantics documentation under the Zagreb surface hopping program.

Application

The Tully's surface hopping technique has been widely used in molecular dynamics simulations to incorporate the non-adiabatic effects. The module can be apply to all classical propagations in multistate systems, specially in those systems where the dynamics cannot be explained using only one electronic state.

Testing

A test example is provided for the excitation of an initial wavepacket into an excited state that is coupled with another excited state. The relevant input file is 'ferreti_tsh.inp' (which can be found in the 'inputs' subdirectory of the sources) and corresponding operator file is 'ferreti.op' (found in the 'operators' subdirectory). The test example is a simple analytical model provided by Ferreti et al. (JCP, 104,5517 (1996), <https://doi.org/10.1063/1.471791>). The documentation under the Zagreb surface hopping code will help you to install the Zagreb code. The test can be done through the following command:

```
$ quantics -mnd ferretti_tsh
```


A more detailed test documentation file ‘sh_zagreb.html’ can be found in the subdirectory ‘sh_zagreb’ of the documentation subdirectory ‘doc’ of the sources. The html file has been provided by the Zagreb code developers.

An output directory is provided for testing and comparison (it is available to download as a tarball, `output.tgz`). The output directory includes the output file (Quantics common output file) and the `zagreb_trj` directory. This directory only includes the first trajectory directory (`traj.1` directory) of a run of a total of 5 trajectories, for space saving reasons.

The run of this test would produce a directory called ‘ferretti_tsh’ in which there are the typical output files produced by Quantics program run, plus a directory ‘zagreb_trj’ in which all the individual trajectories appear (`traj.1`, `traj.2`, etc...) as independent directories. Inside any of the trajectory directory, there are the output files of the classical trajectories run for this test. In order to compare that the run of the test has gone well the file ‘dynamics.out’ inside the ‘traj.1’ directory should be compared. After the propagation of the total number of trajectories the output file inside the ‘ferretti_tsh’ directory is written and this output file should be as well compared. Please notice that, due to the random number generation, required by the algorithms of classical trajectories, some numerical deviation should be expected in both output files.

Source Code

The source code for the Zagreb surface hopping code can be found within the Quantics software which can be downloaded via [gitlab](#). You firstly need to make an account (at gitlab). The Quantics project has a private repository so you also need to be a member of the project to clone it into your computer, then type:

```
git clone https://gitlab.com/quantics/quantics.git
```

Within the Quantics program, explicit code for the Zagreb surface hopping code is located in the subdirectory ‘sh_zagreb’ into the subdirectory ‘source’ of the sources.

The **Zagreb_sh** module is an interface between between Quantics package and the Tully’s surface hoping code provided by the group of Nadja Doslic in Zagreb.

Quantics OpenMP Improvements Module

Software Technical Information

Language Fortran 90

Licence Academic License

Documentation Tool Documentation provided as in-line comments within the source code

Application Documentation Useful documentation can be found on the [Quantics documentation website](#).

Relevant Training Material Training material is available through the tests and examples

- *Purpose of Module*
- *Background Information*
- *Install*
- *Testing*
- *Source Code*

Purpose of Module

This module is related to code developed for 2 SVN revisions targeting OpenMP improvement: v855 and v878 of Quantics (which is available in the [E-CAM branch of Quantics](#)).

31 source files are changed in v855 compared to v854, such as `openmpmod.f90`, `quantics.F90`, `mmomplib.f90` and so on. In v878, OpenMP in database reading/interpolation (`dd_db.f90`) improved. To highlight the relevant sourcecode changes we include them here as patch files: `patch v855` and `patch v878`.

Background Information

Currently the code developed related to this module within the Quantics software package is available through the [CCPForge Qunatics page](#) with the relevant specific changes highlighted above.

Install

1. Under the `install` folder, once the fortran compile is available, do `./install_quantics`.
2. Once the quantics serial version is correctly installed, in the same folder, do `source QUANTICS_client`.
3. Run `compile -O quantics` in order to install the OpenMP version of Quantics.

Testing

Several test example are provided for this module and can be found at `inputs/`. For example the `p24+.inp`. This test works for Quantics's ECAM branch Revision v974 . The Quantics README file will help you to install the Quantics code. After creating a folder named `p24+` and put `p24+.inp` in this folder then change its name to `input`. The test can be done through the following command:

```
$ quantics.omp -omp np -w -I p24+.inp
```

Source Code

The source code for this module can be found within the Quantics software which can be downloaded via [CCPForge](#). You firstly need to make an account (at CCPForge). The quantics project has a private repository so you also need to be a member of the project to checkout. then type into terminal:

```
$ svn checkout --username your-user-name https://ccpforge.cse.rl.ac.uk/svn/quantics/  
↪ gmctdh/quantics/branches/ecam17
```

The **Quantics_openmp** module is an initial effort at OpenMP parallelisation improvements to Quantics.

Quantics Direct Dynamics MPI and OMP code

Software Technical Information

Language Fortran 2003

Licence GNU General Lesser Public License

Documentation Tool Documentation provided as in-line comments within the source code

Application Documentation Useful documentation can be found [here](#)

Relevant Training Material Training material is available through the test examples

Software Module Developed by Quantics code: G. A. Worth, K. Giri, G. W. Richings, M. H. Beck, A. J. Ackle, and H.-D. Meyer. Module: Thierry Tran and Graham Worth.

- *Purpose of Module*
- *Background Information*
- *Application*
- *Testing*
- *Source Code*

Purpose of Module

The module focuses on improving the parallel version of Direct Dynamics variational multi-configuration Gaussian wavepacket (DD-vmCG) method. At every step of the Direct dynamics propagation, the energies, gradients and Hessians are evaluated at the center of each Gaussian wavepacket by calling an external program. One of the challenges of Direct Dynamics is the cost of computation for the evaluation of the potential energy surfaces. The electronic structure calculations are performed for each Gaussian wavepacket individually and thus, parallelizing the call to the electronic structure method greatly decreased the time of computation between each nuclear step. There is already an existing OpenMP parallelization of the code and the purpose of this module is to add an extra MPI layer to it to allow affordable simulation of large systems by spreading the calculations across multiple computation nodes. The module has been added and tested within the Quantics quantum dynamics package which is available on Gitlab.

Background Information

The latest version of Quantics package and the code developed related to this module within the Quantics software package are merged and available through [Quantics.gitlab](#).

Application

This module will be extensively used in the near future to study the photochemistry of large systems, whose size limited the application of the previous version of the DD-vmCG code.

Testing

After Quantics code has been successfully installed. The Quantics README file will help you to install the Quantics code. All the tests available for Direct Dynamics are suitable to test this module and can be found at `inputs/`. A good example to test the MPI version of Quantics is butatriene. After you have copied the `but_dd.inp` file in `inputs/butatriene` and the `but_dddata` directory, the test can be done through the following command:

```
$ mpirun quantics.mpi -mpi test.inp
```

The following command should be in case the code is compiled with both OMP and MPI:

```
$ mpirun quantics.mix -mpi test.inp
```

Source Code

The source code for this module can be found within the Quantics software which can be downloaded via [Quantics.gitlab](https://gitlab.com/quantics/quantics.git). You firstly need to make an account (gitlab). The quantics project has a private repository so you also need to ask for access by emailing Graham Worth (g.a.worth@ucl.ac.uk). In order to clone it into your computer, then type:

```
$ git clone https://gitlab.com/quantics/quantics.git
```

The **Quantics_DD_MPIOMP** module is a further improvement on the parallel version of DD-vMCG in Quantics by adding an extra layer of MPI parallelization to the existing OpenMP parallelization.

Software Technical Information

Name Quantics, SHARC

Language Fortran90, Python 2.7.

Licence None

Documentation Tool In-code comments

Application Documentation <http://chemb125.chem.ucl.ac.uk/worthgrp/quantics/doc/quantics/input.html>

Relevant Training Material Not currently available.

Software Module Developed by Moritz Heindl, Sandra Gomez-Rodriguez

SHARC-gym

Purpose of Module

This module aims at building a bridge between surface hopping (SH)¹ and more accurate methods (summarized with the term QUANTUM in the following) like multiconfigurational time dependent hartree (MCTDH)² and variational multiconfigurational gaussian (vMCG)³ by exploiting both types of methods to overcome the shortcomings of the other in a hybrid approach called the SHARC-gym⁴.

In the computational simulation of molecular movements and reactions various degrees of simplification have been introduced. From exact quantum dynamics available only to model a few degrees of freedom up to huge coarse-grained simulations capable to model whole proteins different levels of sophistication are available. Exact quantum dynamics and methods that will converge to the exact result are capable to shed insight into the most intricate of mechanisms at the heart of processes like photosynthesis. Unfortunately, the use of these methods is hampered by the unfavorable scaling of the simulation time with the size of the investigated system, limiting those approaches to a few dozen degrees of freedom. During the last decades, surface hopping has risen to be one of the most popular approaches for the simulation of events that involve more than a single electronic state and more than 10 atoms. This popularity is

¹ J.C. Tully, R. K. Preston: Trajectory surface hopping approach to nonadiabatic molecular collisions: The reaction of H+ with D2, J. Chem. Phys., 55, 562 (1971).

² M.H. Beck, A. Jackle, G. A. Worth, H.-D. Meyer: The multiconfiguration time-dependent hartree (MCTDH) method: a highly efficient algorithm for propagating wavepackets, Phys. Rep., 324, 1 (2000).

³ G.W. Richings, I. Polyak, K.E. Spinlove, G.A. Worth, I. Burghardt, B. La-sorne: Quantum dynamics simulations using Gaussian wavepackets: the vMCG method, Int Rev Phys Chem, 34, 269 (2015).

⁴ S.Gomez, M. Heindl, A. Szabadi and L. Gonzalez: From Surface Hopping to Quantum Dynamics and Back. Finding Essential Electronic and Nuclear Degrees of Freedom and Optimal Surface Hopping Parameters, J. Phys. Chem. A, 123, 8321 (2019).

due to the ease of implementation of an SH algorithm and the possibility to plug in properties calculated using any of the most popular quantum chemistry packages. However, while the foundations of SH are easy to grasp, the ad hoc nature of SH means that there is never any guarantee that the simulated dynamics for a given system resembles results obtained via more elaborate methods that do not suffer from such crude approximations. Many of the shortcomings of SH have been highlighted in the scientific literature and remedies to overcome those have been proposed. This means that a whole range of various additional parameters and flavours of SH exist at present that are combined or used exclusively at the will of the user, hoping that these corrections will result in a more accurate modelling of the problem at hand.

The SHARC-gym allows the user to overcome this uncertainty by combining SH and QUANTUM methods in a hybrid fashion. The method follows an iterative procedure which is briefly stated here (see also Ref⁴):

1. Hamiltonian loop: The aim of this loop is to select the most important degrees of freedom using SH so that a stripped-down Hamiltonian can be used in QUANTUM dynamics. For this, a full-dimensional SH dynamics is conducted which serves as a reference throughout this loop. From this full-D SH reference, the degrees of freedom (molecular vibrations, movement or even electronic states) that drive the observed dynamics can be determined. Using these essential degrees of freedom, a new model with reduced dimensionality is constructed and a new SH simulation calculated. If this new simulation still contains the most important features of the dynamics, even more degrees of freedom can be cut from the Hamiltonian and the SH dynamics is repeated. Once too many modes have been stripped away and the results diverge from the full-D SH reference, this process is stopped and the Hamiltonian that was used before this last dynamics is used in the subsequent Parameter loop.
2. Parameter loop: In this loop, the reduced Hamiltonian is used in a QUANTUM simulation which serves as a QUANTUM reference throughout the loop. Now that a QUANTUM reference in this reduced Hamiltonian is available, the plethora of parameters available in SH can be validated for this system. If the initially used set of SH parameters was found to perform well, then the SHARC-gym is finished, resulting in a QUANTUM-validated set of parameters for the full-D SH dynamics and a reduced Hamiltonian that captures the essential dynamics of the much bigger system. If the best set of SH parameters diverges from the set that has been used to determine the reduced Hamiltonian, this new set of parameters has to be used again in the Hamiltonian loop and the process has to be repeated as a whole until the best agreement is found.

The hybrid approach of the SHARC-gym enables the use of more accurate QUANTUM methods on a subset of degrees of freedom of larger systems that - as a whole - cannot be treated using a QUANTUM method. This selection of important degrees of freedom is based solely on another dynamics result, eliminating the bias of selecting a set of reactive coordinates beforehand. The SH dynamics benefit from a validation of the chosen parameters against the QUANTUM reference. Furthermore, the SHARC-gym provides a huge amount of possible test systems to quantify the shortcomings of different parameters of SH or even SH as a whole as the SHARC-gym may result in a QUANTUM reference which disagree with all the different flavours of SH. The current implementation of the SHARC-gym uses the SH code SHARC^{5,6} and the set of QUANTUM methods implemented in QUANTICS⁷.

Background Information

The SHARC-gym is currently available from a GitHub repository. It needs a working SHARC installation which is available for free from <https://sharc-md.org/>. Future development will make the SHARC-gym available as a built-in in SHARC and will feature improved functionalities to easily use the QUANTICS set of quantum dynamics methods in combination with SHARC-gym.

⁵ S. Mai, P. Marquetand, L. Gonzalez: Nonadiabatic Dynamics: The SHARC Approach, WIREs Comput. Mol. Sci., 8, e1370 (2018)

⁶ S. Mai, M. Richter, M. Heindl, M. F. S. J. Menger, A. Atkins, M. Ruckebauer, F. Plasser, L. M. Ibele, S. Kropf, M. Oppel, P. Marquetand, L. Gonzalez: SHARC2.1: Surface Hopping Including Arbitrary Couplings — Program Package for Non-Adiabatic Dynamics, (2019)

⁷ G.A. Worth, K. Giri, G. W. Richings, M. H. Beck, A. J. Ackle, H.-D. Meyer: QUANTICS, a suite of programs for molecular QUANTUM dynamics simulations, Version 1.1 (2015)

Building and Testing

The SHARC-gym consists of a set of Python scripts written in Python 2.7. To build a working SHARC installation follow the corresponding installation guide ([SHARC installation](#)).

A test example for the SHARC-gym is available on the SHARC-gym GitHub page. Entering the `testcase` directory, follow the instructions written in `instructions.txt`.

Source Code

The source code can be found in the [SHARC-gym repository on GitHub](#).

The **SHARC-gym** module uses the surface hopping code SHARC and enables the use of a more accurate set of quantum methods implemented in QUANTICS.

3.5.10 CLstunfti

[CLstunfti](#) is an extendable Python toolbox to compute scattering of electrons with a given kinetic energy in liquids and amorphous solids. It uses a continuum trajectory model with differential ionization and scattering cross sections as input to simulate the motion of the electrons through the medium.

Software Technical Information

Name CLstunfti

Language Python, Fortran

Licence GNU General Public License v3

Documentation Tool ReST, Sphinx

Application Documentation <https://gitlab.com/axelschild/CLstunfti>

Relevant Training Material <https://gitlab.com/axelschild/CLstunfti/tree/master/examples>

Software Module Developed by Axel Schild

CLstunfti

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Purpose of Module

CLstunfti is an extendable Python toolbox to compute scattering of electrons with a given kinetic energy in liquids and amorphous solids. It uses a continuum trajectory model with differential ionization and scattering cross sections as input to simulate the motion of the electrons through the medium.

Originally, CLstunfth was developed to simulate two experiments: A measurement of the effective attenuation length (EAL) of photoelectrons in liquid water¹ and a measurement of the photoelectron angular distribution (PAD) of photoelectrons in liquid water². These simulations were performed to determine the elastic mean free path (EMFP) and the inelastic mean free path (IMFP) of liquid water³. Additionally, a program based on CLstunfth is currently being developed which simulates electron scattering in liquids in the presence of laser fields. This extension of CLstunfth is used for simulation of attosecond experiments in liquid water.

The EMFP and IMFP are two central theoretical parameters of every simulation of electron scattering in liquids, but they are not directly accessible experimentally. As CLstunfth can be used to determine the EMFP and IMFP from experimental data, and as it can be easily extended to simulate other problems of particle scattering in liquids, it was decided to make the source code publicly available. For this purpose, within the E-CAM module a documentation for the code was written and examples were designed to test the code and learn how to use CLstunfth.

Background Information

Within this E-CAM module, the necessary steps were taken to make CLstunfth a useful toolbox for other researchers by providing a documentation, examples, and also extensive inline documentation of the source code. CLstunfth is available at <https://gitlab.com/axelschild/CLstunfth> and is published together with the E-CAM module.

Building and Testing

To use CLstunfth, the following steps are necessary:

- **A few Python packages are needed. Specifically, you need:**

- h5py==2.10.0
- matplotlib==3.2.2
- scipy==1.5.0
- numexpr==2.7.1
- numpy==1.19.0

- Move the main folder CLstunfth in a folder named e.g. My_Python_Modules. Then, either run

```
python setup.py build_ext --inplace
```

in the main folder or change to the CLstunfth subfolder and run

```
f2py -c --opt='-O3 -ffast-math' ftools.f95 -m ftools
```

to compile the Fortran code as a module. To make Python know where CLstunfth is, run

```
export PYTHONPATH=$PYTHONPATH:$HOME/My_Python_Modules/CLstunfth
```

in your shell or add the line to the end of your .bashrc (or .zshrc or .cshrc) file.

- Build the documentation by running

¹ Suzuki, Nishizawa, Kurahashi, Suzuki, *Effective attenuation length of an electron in liquid water between 10 and 600 eV*, Phys. Rev. E 90, 010302 (2014).

² Thürmer, Seidel, Faubel, Eberhardt, Hemminger, Bradforth, Winter, *Photoelectron Angular Distributions from Liquid Water: Effects of Electron Scattering*, Phys. Rev. Lett. 111, 173005 (2013).

³ Schild, Peper, Perry, Rattenbacher, Wörner, *An alternative approach for the determination of mean free paths of electron scattering in liquid water based on experimental data*, submitted.

```
make html
```

It is found in `_build/html` (actually, it should already be there).

- Use CLstunfti!

The examples created for this E-CAM module are in the folder `examples`. Each example comes with a sample output which has the same name as the files created by the scripts, but appended with `_ref`. Some of the examples have a rather long runtime, as indicated below. This is because the examples should also show what is needed to compute the relevant targets correctly. If a quick test is preferred, the number of trajectories can be decreased.

The following examples are provided (note that part of the code is in the file `tools_eal_pad.py` in the `example` folder):

- Example 01 shows how to prepare an input for CLstunfti. It is run as

```
python 01_create_input.py
```

It will create the HDF5 file `prop_data.h5` which can be compared with the reference file `prop_data_ref.h5`.

- Example 02 shows how to compute the effective attenuation length (EAL), i.e., the effective/average depth from which photoelectrons are ionized. This is done by selecting many ionization depths z_0 and by fitting the number of electrons detected outside the liquid to

$$P(z_0) \propto e^{-z_0/\text{EAL}}$$

It is run as

```
python 02_compute_eal.py
```

and creates `02_eal.pdf` which can be compared with `02_eal_ref.pdf`. *The calculation takes ca. 1 minute on a 3.40GHz CPU.*

- Example 03 shows how to compute the photoelectron angular distribution (PAD) of electrons that leave the liquid after photoionization. This is done by rotating the PAD for photoionization (which simulates a rotation of the laser used for ionization) away from its default direction (the z -axis, as the default is that $z < 0$ is the liquid and $z = 0$ is the surface) and by detecting the number of electrons outside the liquid depending on the polar angle θ of the rotation. The PAD has the functional form

$$\text{PAD}(\theta) \propto 1 + \beta P_2(\cos(\theta))$$

where P_2 is the Legendre polynomial of second order. Hence, the PAD is fully characterized by the parameter β .

Two ways to do the calculation are provided. The first uses importance sampling of the ionization depth with an exponential distribution, is run with

```
python 03a_compute_pad.py
```

and creates `03a_pad.pdf` which can be compared with `03a_pad_ref.pdf`. *The calculation takes ca. 1 hour on a 3.40GHz CPU.*

The second way uses a linear sampling, where initial positions are added until deeper and deeper in the liquid until no trajectories are leaving it anymore. It is run with

```
python 03b_compute_pad.py
```

and creates `03b_pad.pdf` which can be compared with `03b_pad_ref.pdf`. *The calculation takes a few hours on a 3.40GHz CPU.*

- Example 04 shows how to find elastic and inelastic mean free paths if an EAL and PAD are given. From an initial guess for the EMFP and IMFP, it optimizes their values by comparing the calculated EAL and PAD with a target EAL and PAD. It is run with

```
python 04_find_emfp_imfp.py
```

and provides the terminal output given in `04_find_emfp_imfp_output.txt` for comparison. *The calculation takes ca. 1 hour on a 3.40GHz CPU.*

- Example 05 compares the angular distribution of photoelectrons after ionization, one scattering, two scatterings, etc. in the bulk (no surface) with the known solution. There are four parts. The calculations should be performed in the right order because the results are saved to files.

In the first part, the angular distribution of the electrons after up to nine scatterings in the bulk without inelastic scattering is computed. It is run with

```
python 05a_bulk_prep.py
```

and creates `05a_bulk.pdf` and `05a_bulk.h5` which can be compared with `05a_bulk_ref.pdf` and `05a_bulk_ref.h5`, respectively. *The calculation takes ca. 1.5 hours on a 3.40GHz CPU.*

In the second part, results of the first part are compared with a convolution of the initial PAD with the DSCS and with doing the exact equivalent of the convolution (the convolution only gives the exact result in 2D, in 3D it is more complicated). It is run with

```
python 05b_compare_bulk_convolution.py
```

and creates `05b_compare_bulk_convolution.pdf` which can be compared with `05b_compare_bulk_convolution_ref.pdf`. *The calculation takes a few seconds on a 3.40GHz CPU.*

In the third part, the angular distribution of the electrons after up to nine scatterings is computed outside the surface. It is run with

```
python 05c_surface.py
```

and creates `05c_surface.pdf` which can be compared with `05c_surface_ref.pdf`. *The calculation takes ca. 10 minutes on a 3.40GHz CPU.*

In the fourth part, the results of the first and third part are compared. It is run with

```
python 05d_comparison_bulk_surface.py
```

and creates `05d_comparison_bulk_surface.pdf` which can be compared with `05d_comparison_bulk_surface_ref.pdf`. *The calculation takes a few seconds on a 3.40GHz CPU.*

Source Code

For the module, the [documentation](#) and the [examples](#) were developed and the source code of CLstunfci was extensively commented.

The module **CLstunfci** makes CLstunfci available to the world by providing a documentation of the toolbox and inline documentations of the source code, as well as a set of examples that can also be used for testing.

Spin orbit coupling smoothing

Software Technical Information

Name Guessoc

Language Fortran 90

Licence GNU General Lesser Public License

Documentation Tool Documentation provided in a README file together with the source code.

Application Documentation Detailed documentation related to the running of the module can found here '<https://gitlab.e-cam2020.eu:10443/sanz/durham-ecam/blob/master/README>'

Relevant Training Material Training material is available through the test example

Software Module Developed by Cristina Sanz Sanz

- *Purpose of Module*
- *Building and Testing*
- *Source Code*

Purpose of Module

This module is a standalone program that allows to smooth the off diagonal values (this program is created for spin-orbit couplings) of an electronic structure calculation. The purpose of the module is to remove the sudden changes in the off-diagonal elements (spin-orbit couplings) due to the swap between near states and sign changes that electronic structure calculation programs produce along one of the coordinates (internuclear distance in the test example). The way to remove the discontinuities is based in the idea that after the diagonalisation of the matrix, the eigenvalues are the spin-orbit states that are obtained from the electronic structure calculation, so the (spin-orbit) coupling elements can be optimised (using conjugate gradient in this program) so that after the diagonalisation using the optimised value the eigenvalue is as near as possible as the one obtained from the electronic structure calculation. We ensure the continuity using the couplings of a point near to the point that we want to optimised. Further details can be found in PCCP, 21, 14429 (2019).

The module is particularly thought for computational chemists that need to fit the spin-orbit couplings to use them in quantum and/or classical dynamics simulations.

It is a very practical code that save time to scientist that need to remove discontinuities in the off-diagonal values of the (spin-orbit) couplings. It is particularly useful for matrices bigger 3x3 or 4x4 where the number of couplings makes it difficult to smooth manually.

The applicability of the code is general for all type of dynamical simulations that require the (spin-orbit) couplings as an analytical function or requires the derivatives of the (spin-orbit) couplings.

The code has been already used for the control of the photodissociation of IBr system, where the total number of the states for the simulations is 36, producing a total number of (spin-orbit) couplings equal to 1260. All the couplings needed to be fit to use them in a wavepacket propagation. (Sanz-Sanz C., Worth, G.A., PCCP, 21, 14429-14439 (2019)).

Building and Testing

The module includes a Makefile. To compile you need: a Fortran compiler (gfortran); lapack and blas libraries installed. Adjust library path according to your installation. The compiler included in the Makefile is gfortran and no specification is given for the location of lapack and blas subroutines. Be sure that you have access to the compiler and libraries. Otherwise, change the compiler to use and include the full path of the libraries.

Once the executable is created the user can run it just typing `./guessSO.exe`. The program reads the input files from the directory `inpmat/`. For the testing run there are 36x36 elements, each element of the matrix is in a different file. Open one of the files to see the structure of the input files. The program creates intermediate files, `ssXX-XX.dat`, so that the user can see the evolution of the running. The final output files are `adia.dat`, `adiai.dat`, `soXX-XX.dat`. Where `adiai.dat` are the initial eigenvalues of the non-optimised matrix (spin-orbit states in this example), `adia.dat` are the eigenvalues with the optimised matrix and the `soXX-XX.dat` are the individual elements of the optimised matrix in which the off-diagonal values should be now smooth and suitable for fitting.

Source Code

The source code for this module can be download via [gitlab](#). You firstly need to make an account (at [gitlab](#)). The code of the guessoc module has its own repository so you can clone it typing:

```
git clone ssh://git@gitlab.e-cam2020.eu:10022/sanz/durham-ecam.git or git clone https://gitlab.e-cam2020.eu:10443/sanz/durham-ecam.git
```

The **Spin orbit coupling smoothing** module is to smooth spin orbit couplings along internuclear distance.

Direct Dynamics Database improvements code

Software Technical Information

Language Fortran 2003

Licence GNU General Lesser Public License

Documentation Tool Documentation provided as in-line comments within the source code

Application Documentation Useful documentation can be found [here](#)

Relevant Training Material Training material is available through the test examples

Software Module Developed by Quantics code: G. A. Worth, K. Giri, G. W. Richings, M. H. Beck, A. J. Ackle, and H.-D. Meyer. Module: Georgia Christopoulou and Graham Worth.

- *Purpose of Module*
- *Background Information*
- *Application*
- *Testing*
- *Source Code*

Purpose of Module

The module focuses on improving the efficiency of Direct Dynamics variational multi-configuration Gaussian wavepacket (DD-vMCG) method. During every Direct Dynamics propagation step the calculated energies, gradients and hessian matrix are stored in a database. One important challenge of this method is the time needed to continually reread, sort and analyze this database which makes the calculation of a large system very expensive. Employing a dynamic and smaller version of the database with selected points only relevant to each basis function, each time the program needs to use stored data points, reduces massively the cost of the calculation. Thus, treatment of larger systems is now possible. The module has been added and tested within the Quantics quantum dynamics package which is available on Gitlab and at the same time the code benefits from parallel running.

Background Information

The latest version of quantics package and the code developed related to this module within the Quantics software package are merged and available through [Quantics.gitlab](https://gitlab.com/quantics/quantics).

Application

This module will be extensively used in the near future to study the photochemistry of large systems, whose size limited the application of the previous version of the DD-vMCG code.

Testing

After Quantics code has been successfully installed. The Quantics README file will help you to install the Quantics code. All the tests available for Direct Dynamics are suitable to test this module and can be found at `inputs/`. A good and quick example is Butatriene. After you have copied the `but_dd.inp` file and the `but_dddata` directory, the test can be done through the following command:

```
$ quantics but_dd.inp
```

Source Code

The source code for this module can be found within the Quantics software which can be downloaded via [Quantics.gitlab](https://gitlab.com/quantics/quantics). You firstly need to make an account (gitlab). The quantics project has a private repository so you also need to ask for access by emailing Graham Worth (g.a.worth@ucl.ac.uk). In order to clone it into your computer, then type:

```
$ git clone https://gitlab.com/quantics/quantics.git
```

The **Direct Dynamics Database** The Direct Dynamics Database module is an improved, more efficient version of the database used to provide the potential energy surfaces in the Direct Dynamics variational multi-configuration Gaussian wavepacket (DD-vMCG) method [Wor1] which is included in the powerful and flexible [Quantics](https://gitlab.com/quantics/quantics) package program [Wor2].

3.5.11 EIVibRot

[EIVibRot](#) is a package for general quantum dynamics simulation using curvilinear coordinates. The code has no built-in limitation in terms of the number of degrees of freedom. It applied a numerical but exact kinetic energy operator with Tnum (Automatic differentiation), which enables much flexibility in the choice of the curvilinear coordinates

[Tnum]. Moreover, the Smolyak algorithm [Smo] is employed to avoid the conventional direct-product basis sets and grids, which allows the simulation of larger systems. Typically, the package could be used for

- 1) Vibrational levels, intensities for floppy molecular systems;
- 2) Wave-packet propagation with or without time dependent Hamiltonian;
- 3) Quantum gate and optimal control;
- 4) Optimization with the given set of curvilinear coordinates.

Software Technical Information

Name ElVibRot Time-independent MPI

Language Fortran 90

Licence GNU Lesser General Public License (<http://www.gnu.org/licenses/>)

Documentation Tool Doxygen

Application Documentation See [ElVibRot doc](#) and [Tnum doc](#)

Relevant Training Material Not currently available

Software Module Developed by David Lauvergnat, Ahai Chen

ElVibRot-TID-MPI

- *Purpose of Module*
- *Background Information*
- *Applications of the Module*
- *Building and Testing*
- *Source Code*
- *References*

Purpose of Module

The ElVibRot-TID-MPI (ElVibRot time-independent MPI) module is a parallelized time-independent quantum simulation program. The Davidson algorithm is the main method employed for getting the Eigen levels of the Hamiltonian. This module is a part of the [ElVibRot](#) package designed for general quantum dynamics simulation using curvilinear coordinates. The code has no built-in limitation in terms of the number of degrees of freedom. It applied a numerical but exact kinetic energy operator with Tnum [Tn1], which enables much flexibility in the choice of the curvilinear coordinates. To avoid the conventional direct-product basis sets and grids, the Smolyak algorithm [Sm1] is employed to make possible the simulation of larger systems.

Background Information

The core of the quantum simulation lies in solving the Schrodinger equation with the Hamiltonian of the considered system. The principle fence of the simulation comes from the exponential growth of computational demand with the

increasing of the degrees of freedom of the system, the curse of dimensionality. It prompts numbers of algorithms in the past decades to deal with this difficulty. The Smolyak algorithm, proposed by Smolyak in 1963 [Sm1], provides a powerful method to deal with high-dimensional problems. By introducing the Smolyak algorithm in this module, the wavefunction is expanded as a weighted sum of small Smolyak wavefunction contributions, thus significantly reduce the computational demand of the simulation. Taking advantage of the structure of the newly transformed wavefunction, the MPI can be well implemented. As a result, the simulation could be performed with high accuracy, and in the meantime, impressive parallel efficiency. The code is designed to works on different levels of clusters. The module provides three MPI schemes to adapt the simulation of different kinds of systems and working machines. The default setting will automatically choose the scheme according to the balance of resource consumed and the parallelization efficiency.

Applications of the Module

This module is intended to provide a parallel program for the quantum simulation of general molecular system. Typically, it could be used to calculate the vibrational levels of molecular systems. The general capability of the simulation could be up to tens of degrees of freedom. The code has been applied for the simulation of Malondialdehyde ($C_3H_4O_2$), which is of 21 degrees of freedom. The parallelization of the code enables the simulation of larger systems.

Building and Testing

The code is compatible with `gfortran`, `mpifort`, `ifort`, `pgf90`, etc. Building the program requires OpenMPI v2.0 or above. OpenMPI should be built as 64-bit for the simulation of very large system.

- build with MPI

set makefile:

```
F90=mpifort
MPICORE=gfortran ! gfortran or ifort according to the compiler for MPI
```

other main options:

```
F90=gfortran      ! compile with gfortran
F90=ifort         ! compile with ifort
F90=pgf90        ! compile with pgf90
parallel_make=1  ! enable parallel make with -j argument
OMP=1            ! enable openMP
OPT=1            ! enable code optimization
INT=4            ! 4 or 8 for 32-bits or 64-bits integer
LAPACK=1         ! enable LAPACK
ARPACK=1         ! enable ARPACK
QML=1           ! enable QMLib
```

To build:

```
make
```

To test:

```
make test
```

To clean test files

```
make cleantest
```

Three MPI schemes will be tested for 6 and 21 degrees of freedom systems. In directory

```
./Working_tests/MPI_tests
```

check folders 6D_Davidson_* and 21D_Davidson_* for examples. For more details, see [ElVibRot](#).

Source Code

See the [MPI branch](#) of ElVibRot

References

The **ElVibRot-TID-MPI** (ElVibRot Time-independent MPI) module is a parallelized time-independent quantum simulation program. The Davidson algorithm is the main method employed for getting the Eigen levels of the Hamiltonian.

Software Technical Information

Name ElVibRot Time-dependent MPI

Language Fortran 90

Licence GNU Lesser General Public License (<http://www.gnu.org/licenses/>)

Documentation Tool Doxygen

Application Documentation See [ElVibRot doc](#) and [Tnum doc](#)

Relevant Training Material Not currently available

Software Module Developed by David Lauvergnat, Ahai Chen

ElVibRot-TD-MPI

- *Purpose of Module*
- *Background Information*
- *Applications of the Module*
- *Building and Testing*
- *Source Code*
- *References*

Purpose of Module

The ElVibRot time-dependent MPI (ElVibRot-TD-MPI) module is a parallelized time-dependent quantum simulation program. It is a part of the [ElVibRot](#) package designed for general quantum dynamics simulation using curvilinear coordinates. There is no built-in limitation on the degrees of freedom for the target system. The code employed a numerical but exact kinetic energy operator with Tnum [[Tn](#)]. The Smolyak algorithm [[Sm](#)] is applied to avoid the direct-product basis sets and grids in the simulation.

Background Information

Quantum dynamics simulation has provided powerful insight into the underlying mechanism of chemical reactions, laser molecular interaction, etc. The simulation, with the conventional expansion on the direct-product basis, is limited by the exponential growth of computational cost with the increase of degrees of freedom. The multi-configuration time-dependent Hartree, a well-known package developed with the aim of generality, expands the wavefunction as a sum of Hartree products with single-particle functions, leading to a very efficient wavepackage propagation. The quantum diffusion Monte Carlo and the Feynman path integral approaches get around the problem by avoiding expanding the wavefunction on a basis set. The variational multi-configuration Gaussian applies on-the-fly quantum chemical calculation of the potential energy to approach the quantum effects in the photochemistry. However, the Smolyak method provides another way to deal with high-dimensional problems, without losing accuracy and universality. The application of Smolyak algorithm enables the simulation of large systems (> 12 degrees of freedom) as the wavefunction is expanded as a weighted sum of small Smolyak wavefunction contributions. MPI is implemented depends on this framework. The module is designed to works on different levels of clusters. Three MPI schemes are provided in accord with a series of well-known propagation methods, including the Chebyshev, Runge-Kunta, short iterative Lanczos and Taylor expansion, etc. The three MPI schemes correspond to the simulation with the mode of most efficiency, memory saving, and massive cluster parallelization, respectively. The default setting will automatically choose the scheme according to the balance of resource consumed and the parallelization efficiency.

Applications of the Module

This module is intended to provide a parallel program for general wavepackage propagation. The general capability of the simulation could be up to tens of degrees of freedom. The propagation time could be up to hundreds of femtoseconds with general computation time, according to the selected propagation method. The code has been applied for the simulation of Pyrazine ($C_4H_4N_2$), which is of 24 degrees of freedom. This module could be a practical tool for general quantum molecular simulation, supporting the further study of molecular dynamics in chemical reactions, ultrafast processes, etc.

Building and Testing

The code is compatible with `gfortran`, `mpifort`, `ifort`, `pgf90`, etc. Building the program requires OpenMPI v2.0 or above. OpenMPI should be built as 64-bit for the simulation of very large systems.

- build with MPI

set makefile:

```
F90=mpifort
MPICORE=gfortran ! gfortran or ifort according to the compiler for MPI
```

other main options:

```
F90=gfortran      ! compile with gfortran
F90=ifort         ! compile with ifort
F90=pgf90         ! compile with pgf90
parallel_make=1  ! enable parallel make with -j argument
OMP=1            ! enable openMP
OPT=1            ! enable code optimization
INT=4            ! 4 or 8 for 32-bits or 64-bits integer
LAPACK=1         ! enable LAPACK
ARPACK=1         ! enable ARPACK
QML=1           ! enable QMLib
```

To build:


```
make
```

To test:

```
make test
```

To clean test files

```
make cleantest
```

Three MPI schemes will be tested for 12 and 24 degrees of freedom systems. In directory

```
./Working_tests/MPI_tests
```

check folders 12D_propagation_* and 24D_propagation_* for examples. For more details, see [ElVibRot](#).

Source Code

See the [MPI branch](#) of ElVibRot

References

The **ElVibRot-TD-MPI** (ElVibRot Time-dependent MPI) module is a parallelized time-dependent quantum simulation program. The available propagation methods include Chebyshev, Runge-Kunta, short iterative Lanczos and Taylor expansion, etc.

3.5.12 References

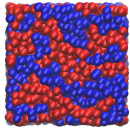
General Information

Contents

- *Meso- and Multi-scale Modules*
 - *Introduction*
 - *Pilot Projects*
 - *Software related to Extended Software Development Workshops*
- *How to contribute?*
- search

Meso- and Multi-scale Modules

4.1 Introduction



This is a collection of the modules that have been created by E-CAM community within the area of Meso- and Multi-scale Modelling. This documentation is created using ReStructured Text and the git repository for the documentation source files can be found at <https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library> which are public and open to contributions.

In the context of E-CAM, the definition of a software module is any piece of software that could be of use to the E-CAM community and that encapsulates some additional functionality, enhanced performance or improved usability for people performing computational simulations in the domain areas of interest to us.

This definition is deliberately broader than the traditional concept of a module as defined in the semantics of most high-level programming languages and is intended to capture inter alia workflow scripts, analysis tools and test suites as well as traditional subroutines and functions. Because such E-CAM modules will form a heterogeneous collection we prefer to refer to this as an E-CAM software repository rather than a library (since the word library carries a particular meaning in the programming world). The modules do however share with the traditional computer science definition the concept of hiding the internal workings of a module behind simple and well-defined interfaces. It is probable that in many cases the modules will result from the abstraction and refactoring of useful ideas from existing codes rather than being written entirely de novo.

Perhaps more important than exactly what a module is, is how it is written and used. A final E-CAM module adheres to current best-practice programming style conventions, is well documented and comes with either regression or unit tests (and any necessary associated data). E-CAM modules should be written in such a way that they can potentially take advantage of anticipated hardware developments in the near future (and this is one of the training objectives of E-CAM).

4.2 Pilot Projects

One of primary activity of E-CAM is to engage with pilot projects with industrial partners. These projects are conceived together with the partner and typically are to facilitate or improve the scope of computational simulation within the partner. The related code development for the pilot projects are open source (where the licence of the underlying software allows this) and are described in the modules associated with the pilot projects.

4.3 Software related to Extended Software Development Workshops

4.3.1 DL_MESO_DPD

The following modules connected to the DL_MESO_DPD code (master version) have been produced so far:

Analysis of charge dipole moments in DL_MESO_DPD

Software Technical Information

Language Fortran 2003

Licence BSD

Documentation Tool RST and LaTeX-generated .pdf file

Application Documentation [Click to download the manual with more details](#)

Relevant Training Material See the Testing section

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*

Purpose of Module

This module, `gen_dipole.f90`, is a generalization of the `dipole.f90` post-processing utility of DL_MESO_DPD, the Dissipative Particle Dynamics (DPD) code from the [DL_MESO](#) package.

It processes trajectory (HISTORY) files to obtain the charge dipole moments of all the (neutral) molecules in the system. It produces files *dipole_** containing the time evolution of relevant quantities (see below). In the case of a single molecular species, it also prints to the standard output the Kirkwood number g_k and the relative electric permittivity ϵ_r for this species, together with an estimate for their errors (standard deviation).

The module can be applied to systems including molecules with a generic charge structure, as long as each molecule is neutral (otherwise the charge dipole moment would be frame-dependent).

The charge dipole moment of a neutral molecule is $\vec{p}_{mol} = \sum_{i \in mol} q_i \vec{r}_i$ where \vec{r}_i are the bead positions and q_i their charges. The total charge dipole moment of the simulated volume V is $\vec{P} = \sum_{mol \in V} \vec{p}_{mol}$. If more than one molecular species are present, one can split \vec{P} into the different species' contributions.

In general:

For any molecular species a file `dipole_{molecule name}` is produced, whose columns are snapshot index, $P_x, P_y, P_z, \sum_{i=1}^{N_{mol}} \frac{\vec{p}_i^2}{N_{mol}}, \frac{\vec{P}^2}{V}$. It is intended that for any quantity the contribution given from the species {molecule name} is reported (i.e., the sums are restricted to molecules of a single type).

Possible uses of the output files are: monitoring the polarization in response to an external electric field, measuring the fluctuations in molecular/total charge dipole moments.

Extra output for a single molecular species:

The Kirkwood number for a pure system is $g_k = \frac{\langle \vec{P}^2 \rangle}{N_{mol} \langle \vec{p}^2 \rangle}$, where $\langle \dots \rangle$ indicates an average over trajectories. If the dipoles' orientations are not correlated, then $g_k \simeq 1$. Also, the relative dielectric permittivity of the medium is calculated from linear response theory: $\epsilon_r = 1 + \frac{4\pi}{3} l_B \frac{\langle \vec{P}^2 \rangle}{V}$, where l_B is Bjerrum length and tin-foil boundary conditions are used.

Background Information

The base code for this module is DL_MESO_DPD, the Dissipative Particle Dynamics code from the mesoscopic simulation package DL_MESO, developed by M. Seaton at Daresbury Laboratory. This open source code is available from STFC under both academic (free) and commercial (paid) licenses. The module is to be used with DL_MESO in its last released version, version 2.7 (dating December 2018).

A variant of this module for use with a previous version of DL_MESO, version 2.6 (dating November 2015), can be found in the `old-v2.6` directory¹.

Testing

The present module `gen_dipole.f90` is compiled with the available Fortran90 compiler, e.g.:

```
gfortran -o gen_dipole.exe gen_dipole.f90
```

and the executable must be in the same directory of the HISTORY file to be analyzed. The user will be asked to provide the Bjerrum length used in single molecule simulations: all other information (including electric charges on all bead types) required for analyses is provided in the HISTORY file.

To input the Bjerrum length, one can either enter it from the keyboard or write it into a text file (say, `input.txt`) and run the program in this way:

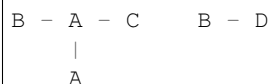
```
gen_dipoleaf.exe < input.txt
```

We propose two tests to familiarize users with the utility and a third one on a physically relevant system.

The first two tests involve two (toy) molecular species: a branched one (four beads, T-shaped) and a simple dimer. All the beads carry charges. In the first case 10 molecules of each type are present and are followed for a few time steps. In the second case we suggest analyzing a single snapshot with just two molecules and all the beads sitting at user-defined positions (via a CONFIG file).

Four type of beads are used with charges $q_A = 0.2, q_B = -1, q_C = 0.6, q_D = 1$; the Bjerrum length is fixed as $l_B = 1$.

The bonding connections in the two molecules are pictorially given below:



¹ A small change to specifying charge smearing schemes and lengths in CONTROL files has been made since version 2.6: the `old-v2.6` folder includes CONTROL files for the tests shown here that will work with this version of DL_MESO.

First test

Run the DL_MESO_DPD simulation on a single node (serial run) using the following CONTROL file:

```
Two kinds of molecules: branched and dimer

volume 3.0 3.0 3.0
temperature 1.0
cutoff 1.0

timestep 0.01
steps 1000
equilibration steps 0
traj 0 100 0
stats every 100
stack size 100
print every 100
job time 1000.0
close time 10.0

ensemble nvt mdvv
conf origin zero

ewald sum 1.0 5 5 5
bjerrum 1.0
smear gauss
smear length 0.5 equal

finish
```

and the FIELD file:

```
Two kinds of molecules: branched and dimer

SPECIES 4
A 1.0 0.2 0 0
B 1.0 -1.0 0 0
C 1.0 0.6 0 0
D 1.0 1.0 0 0

MOLECULES 2
BRANCH
nummols 10
beads 4
B 0.0 0.0 0.0
A 0.0 0.2 0.0
C 0.0 0.4 0.0
A 0.2 0.2 0.0
bonds 3
harm 1 2 5.0 0.25
harm 2 3 5.0 0.25
harm 2 4 5.0 0.25
finish
BD
nummols 10
beads 2
B 0.0 0.0 0.3
D 0.0 0.0 0.1
```

(continues on next page)

(continued from previous page)

```

bonds 1
harm 1 2 5.0 0.25
finish

INTERACTIONS 4
A      A      dpd 25.0 1.0 4.5
B      B      dpd 25.0 1.0 4.5
C      C      dpd 25.0 1.0 4.5
D      D      dpd 25.0 1.0 4.5

CLOSE

```

Analyzing the HISTORY file with *gen_dipole.exe*, this output is printed on the standard output

```

nchist:          0          10          0          10
Number of snapshots:      11
<P_x>, <P_y>, <P_z>:
3.635198E-01   -9.224687E-02   5.177166E-01   8.410412E-01   3.127008E-01   5.
↪555975E-01
error:
6.342381E-01   3.990649E-01   7.284979E-01   3.364702E-01   4.627111E-01   5.
↪658327E-01
<P^2>/V:
4.857672E-01   2.793887E-01
error:
1.374485E-01   8.232245E-02
<p^2>:
1.381681E+00   8.502445E-01
error:
1.453820E-01   9.631156E-02

```

The first line shows the histogram of cluster sizes: in this case, it correctly gives 10 molecules of two beads, and 10 molecules of 4 beads. Since internally the module checks that each molecule is a connected cluster², this line should always give a histogram with the molecule sizes (up to the detected maximum number of beads per molecule).

The resulting *dipole_BD* file is

```

↪02      1      5.411139E-01   -3.599928E-01   -4.640084E-01   4.000000E-02   2.361863E-
↪01      2     -2.554169E+00   -5.878279E-01   -1.010825E+00   1.310513E+00   2.922626E-
↪042780E+00      3      1.110258E-01   1.629865E+00   5.048394E+00   9.721937E-01   1.
↪01      4     -1.527498E+00   5.440944E-01   1.932210E+00   8.096611E-01   2.356565E-
↪01      5      1.962445E+00   7.272408E-01   1.737201E+00   7.240524E-01   2.739976E-
↪02      6     -2.179062E-01   -4.113665E-01   -1.396401E+00   8.849979E-01   8.024596E-
↪01      7      1.552674E-01   2.753005E+00   -1.427860E-01   1.113470E+00   2.823531E-
↪01      8     -8.580591E-01   1.490655E+00   7.832797E-01   1.145730E+00   1.322905E-
↪01

```

(continues on next page)

² Disambiguation on the concept of molecule. In DL_MESO a *defined molecule* is a set of beads, which can be bonded or not. For the purpose of this module it is *required* that each molecule is a connected cluster (via stretching bonds). In fact, this - together with the reasonable assumption that each stretching bond cannot be stretched to more than half the system linear size - allows us to univocally define the charge dipole moment of each molecule.

(continued from previous page)

| | | | | | | |
|-----|----|---------------|--------------|---------------|--------------|------------|
| ↪01 | 9 | 2.232485E+00 | 2.690999E+00 | -1.087863E+00 | 6.544034E-01 | 4.966263E- |
| ↪01 | 10 | -1.225526E-01 | 3.129381E-01 | 1.879348E+00 | 7.360628E-01 | 1.349962E- |
| ↪02 | 11 | -7.368671E-01 | 4.618429E-01 | -1.166975E+00 | 9.616044E-01 | 7.844829E- |

and the *dipole_BRANCH* one is

| | | | | | | |
|-------------|----|---------------|---------------|---------------|--------------|------------|
| ↪02 | 1 | -3.258640E-01 | -4.896253E-01 | -3.064526E-01 | 1.040000E-01 | 1.629013E- |
| ↪563133E+00 | 2 | 4.901658E+00 | 4.033448E+00 | -1.381903E+00 | 1.525634E+00 | 1. |
| ↪01 | 3 | -1.233831E+00 | 9.305151E-01 | 3.127786E+00 | 2.095572E+00 | 4.507868E- |
| ↪01 | 4 | -3.059065E+00 | -4.542448E-01 | -1.278650E+00 | 1.559279E+00 | 4.147838E- |
| ↪01 | 5 | 1.576574E+00 | -4.318085E+00 | 2.111019E+00 | 1.067546E+00 | 9.476980E- |
| ↪01 | 6 | 4.662262E-01 | -1.342421E+00 | 2.268855E+00 | 1.601749E+00 | 2.654505E- |
| ↪02 | 7 | -6.572569E-01 | -9.276115E-02 | 5.649749E-01 | 1.480806E+00 | 2.814029E- |
| ↪01 | 8 | -1.704146E+00 | -1.273019E+00 | 7.975968E-01 | 1.078223E+00 | 1.911427E- |
| ↪01 | 9 | 9.061683E-01 | 1.749560E+00 | -2.253253E-01 | 1.526127E+00 | 1.456620E- |
| ↪01 | 10 | 2.677888E-01 | 3.099173E+00 | -6.472147E-01 | 1.494043E+00 | 3.739063E- |
| ↪01 | 11 | 2.860466E+00 | 3.852343E+00 | -1.590978E+00 | 1.665513E+00 | 9.464453E- |

If instead the simulation is run on multiple nodes, only the results for the first snapshot will be unchanged (i.e., the first line of each *dipole_** file). The other results will vary because different sequences of random numbers will be used by DL_MESO_DPD for the time evolution of the system.

Second test

Run DL_MESO_DPD using the same CONTROL and FIELD files as above, with the following changes:

- change “*steps 1000*” to “*steps 1*” (in CONTROL)
- change “*nummols 10*” to “*nummols 1*” (NB: appears twice in FIELD)

Also, use this CONFIG file that will initially align the molecule branches with the Cartesian axes:

```
Two kind of molecules, branched and dimer
0      1
3.0    0.0    0.0
0.0    3.0    0.0
0.0    0.0    3.0
B      1
0.0 0.0 0.0
A      2
0.0 0.2 0.0
C      3
0.0 0.4 0.0
A      4
```

(continues on next page)

(continued from previous page)

```

0.2 0.2 0.0
B      5
0.0 0.0 0.3
D      6
0.0 0.0 0.1

```

where the identity of each bead is fixed by the FIELD file and is shown below:

```

B (1) - A (2) - C (3)    B (5) - D (6)
      |
      A (4)

```

One can easily check that the dipole of each molecule is as expected (within machine precision):

$$\vec{p}_{BRANCH} = (0.04, 0.32, 0), \quad \vec{p}_{BD} = (0, 0, -0.2).$$

The resulting *dipole*_BD file is

```

      1      0.000000E+00      0.000000E+00      -2.000000E-01      4.000000E-02      1.481481E-
↪ 03

```

and the *dipole*_BRANCH one is

```

      1      4.000000E-02      3.200000E-01      2.220446E-16      1.040000E-01      3.851852E-
↪ 03

```

The results of this test will not depend on the number of nodes used to run the simulation³.

Third test: water in oil

Here we suggest considering a fluid made up of harmonically bonded dimers $(+q, -q)$. Appropriately fixing the partial charges q and the Bjerrum length l_B , this system mimics water in an oil background as far as its dielectric properties are concerned. For more details about this model, please see the page [Test case: a dimer solvent](#).

Run DL_MESO_DPD using the following CONTROL file:

```

DL_MESO charged harmonic dimers with dpd repulsion

volume 64.0
temperature 1.0
cutoff 1.0

timestep 0.01
steps 70000
equilibration steps 20000
traj 20000 100
stats every 100
stack size 100
print every 100
job time 7200.0
close time 100.0

ensemble nvt mdvv

ewald sum 1.0 5 5 5

```

(continues on next page)

³ The tiny value for P_z in *dipole*_BRANCH may vary, but for this test it should be no greater than the smallest available non-negligible floating-point number.

(continued from previous page)

```

bjerrum 42.0
smear gauss
smear length 0.5 equal

finish

```

and the FIELD file:

```

DL_MESO charged harmonic dimers with dpd repulsion

SPECIES 2
solp 1.0 0.46 0
solm 1.0 -0.46 0

MOLECULES 1
DIMER
nummols 96
beads 2
solp 0.0 0.0 0.0
solm 0.1 0.0 0.0
bonds 1
harm 1 2 5.0 0.0

finish

INTERACTIONS 3
solp solp dpd 25.0 1.0 4.5
solm solm dpd 25.0 1.0 4.5
solp solm dpd 25.0 1.0 4.5

CLOSE

```

Analyzing the HISTORY file with *gen_dipole.exe*, this output is printed to the standard output:

```

nchist:          0          96
Number of snapshots:      501
<P_x>, <P_y>, <P_z>:
-4.348820E-02    5.931873E-02    6.210429E-02
error:
1.035501E-01    9.685632E-02    9.780560E-02
<P^2>/V:
2.324029E-01
error:
8.812749E-03
<p^2>:
1.416901E-01
error:
4.351294E-04
kirkwood factor:
1.093480E+00
error:
4.482298E-02
Bjerrum length?
42.0
epsilon_r:
4.188645E+01
error:

```

(continues on next page)

(continued from previous page)

1.550420E+00

In particular, we see that:

- $\vec{P} = (0.0 \pm 0.1, 0.1 \pm 0.1, 0.1 \pm 0.1)$
- $\epsilon_r = 42 \pm 2$
- $g_k = 1.09 \pm 0.04$

Please note that the error estimates are calculated assuming all the samples are independent. From the results obtained in the testing case of the module *gen_dipoleaff90*, one sees that the auto-correlation time of \vec{P} in this system is about 1-2 DPD time units, so the sampling choice used here (trajectories are written every 100 time steps, i.e., at each DPD time unit) seems reasonable, even if a little bit optimistic. To confirm the reliability of the error estimate, one can carry out another run with a different random number sequence (using the CONTROL file directive *seed*) and see if the two results are compatible within error bars.

Source Code

```

1  PROGRAM gen_dipole
2  !*****
3  ! module to analyze charge dipole moments in DL_MESO
4  !
5  ! authors: m. a. seaton and s. chiacchiera, February 2017 (amended January 2021)
6  !*****
7
8  IMPLICIT none
9  INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND (15, 307)
10 INTEGER, PARAMETER :: li = SELECTED_INT_KIND (12)
11 INTEGER, PARAMETER :: ntraj=10
12 INTEGER, PARAMETER :: endversion = 1
13 REAL(KIND=dp), PARAMETER :: pi=3.141592653589793_dp
14
15 CHARACTER(80) :: text
16 CHARACTER(8), ALLOCATABLE :: namspe (:), nammol (:)
17
18 INTEGER, ALLOCATABLE :: ltp (:), ltm (:), mole (:), bndtbl (:,:)
19 INTEGER, ALLOCATABLE :: nbdmol (:), readint (:)
20 INTEGER, ALLOCATABLE :: visit (:), from (:)
21 INTEGER :: nrtout
22 INTEGER :: chain, imol, ioerror, i, numtraj, j, k, nmoldef, ibond, nbdmolmx
23
24 !molecule
25 INTEGER :: nummol, lfrzn, rnmol, keytrj, srfx, srfy, srfz
26 INTEGER :: nav
27 INTEGER :: endver, Dlen, nstep, framesize, lend, leni
28 INTEGER(KIND=li) :: filesize, currentpos, lend_li, leni_li
29
30 REAL(KIND=dp), ALLOCATABLE :: xxx (:), yyy (:), zzz (:), readdata (:)
31 REAL(KIND=dp), ALLOCATABLE :: nmol (:), chg (:), molchg (:)
32 REAL(KIND=dp), ALLOCATABLE :: dipx_box (:), dipy_box (:), dipz_box (:)
33 REAL(KIND=dp), ALLOCATABLE :: dip2_box (:), dip2_ave (:)
34 REAL(KIND=dp), ALLOCATABLE :: dip2_err (:)
35 REAL(KIND=dp), ALLOCATABLE :: sum_dipx_box (:), sum_dipy_box (:), sum_dipz_box (:)
36
37 !box2 (:)
38 REAL(KIND=dp), ALLOCATABLE :: sum_dipx_box2 (:), sum_dipy_box2 (:), sum_dipz_
39
40 !box2 (:)

```

(continues on next page)

(continued from previous page)

```

35     REAL(KIND=dp), ALLOCATABLE :: sum_dip2_box (:), sum_dip_box4 (:)
36     REAL(KIND=dp), ALLOCATABLE :: dipx_box_ave (:), dipy_box_ave (:), dipz_box_ave_
↪(:)
37     REAL(KIND=dp), ALLOCATABLE :: dipx_box2_ave (:), dipy_box2_ave (:), dipz_box2_
↪ave (:)
38     REAL(KIND=dp), ALLOCATABLE :: dipx_box_err (:), dipy_box_err (:), dipz_box_err_
↪(:)
39     REAL(KIND=dp), ALLOCATABLE :: dip_box2_ave (:), dip_box2_err (:), dip_box4_ave_
↪(:)
40     REAL(KIND=dp), ALLOCATABLE :: gk (:), gk_err (:)
41     REAL(KIND=dp), ALLOCATABLE :: sum_dip2_box2 (:)
42     REAL(KIND=dp), ALLOCATABLE :: epsilon_r (:), epsilon_r_err(:)
43     REAL(KIND=dp) :: bjerelec
44     REAL(KIND=dp) :: volm, dimx, dimy, dimz, shrdx, shrdy, shrdz
45     REAL(KIND=dp) :: amass, rcii
46     REAL(KIND=dp) :: time
47
48     LOGICAL :: eof, swapend, bigend
49
50     !   determine number of bytes for selected double precision and integer kinds
51     !   (the default SELECTED_REAL_KIND (15, 307) should return 8 bytes)
52
53     lend = STORAGE_SIZE (1.0_dp) / 8
54     leni = BIT_SIZE (1) / 8
55     lend_li = INT (lend, KIND=li)
56     leni_li = INT (leni, KIND=li)
57
58     !   check endianness of machine
59
60     bigend = (IACHAR(TRANSFER(1,"a"))==0)
61
62     !   determine if HISTORY file exists, which endianness to use,
63     !   if type of real is correct
64
65     INQUIRE (file = 'HISTORY', EXIST = eof)
66     IF (.NOT. eof) THEN
67         PRINT *, "ERROR: cannot find HISTORY file"
68         STOP
69     END IF
70
71     OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted', status_
↪= 'unknown')
72
73     swapend = .false.
74     READ (ntraj) endver, Dlen
75
76     IF (endver/=endversion) THEN
77         swapend = .true.
78         CLOSE (ntraj)
79         IF (bigend) THEN
80             OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',
↪status = 'unknown', convert = 'little_endian')
81             ELSE
82             OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',
↪status = 'unknown', convert = 'big_endian')
83             END IF
84             READ (ntraj) endver, Dlen

```

(continues on next page)

(continued from previous page)

```

85         IF (endver/=endversion) THEN
86             PRINT *, "ERROR: corrupted HISTORY file or created with incorrect version_
↳ of DL_MESO"
87             STOP
88         END IF
89     END IF
90
91     IF (Dlen/=lend) THEN
92         PRINT *, "ERROR: incorrect type of real number used in HISTORY file"
93         PRINT *, "      recompile gen_dipole.f90 with reals of ", Dlen, " bytes"
94         STOP
95     END IF
96
97     !      read file size, number of frames and timestep numbers
98
99     READ (ntraj) filesize, numtraj, nstep
100
101     ! Read where the number of beads, molecules and bonds are determined
102     ! Arrays are filled with names of particles and molecules
103
104     READ (ntraj) text
105
106     READ (ntraj) nspe, nmoldef, nusyst, nsyst, numbond, keytrj, srfx, srfy, srfz
107
108     IF (numbond==0) THEN
109         PRINT *, 'ERROR: no molecules in trajectory data!'
110         STOP
111     END IF
112
113     IF (srfx==1 .OR. srfy==1 .OR. srfz==1) THEN
114         WRITE (*,*) "ERROR: Systems under shear not yet implemented!"
115         STOP
116     END IF
117
118     framesize = (keytrj+1) * 3
119     ALLOCATE (readint (1:nsyst), readdata (1:framesize))
120
121     !      get number of beads to be tracked when reading trajectory file (molecular beads)
122     nmbeads = nsyst - nusyst
123
124     ALLOCATE (namspe (nspe), nammol (nmoldef))
125     ALLOCATE (xxx (1:nmbeads), yyy (1:nmbeads), zzz (1:nmbeads))
126     ALLOCATE (ltp (1:nmbeads), ltm (1:nmbeads), mole (1:nmbeads))
127     ALLOCATE (nmol (1:nmoldef), nbdmol (1:nmoldef))
128     ALLOCATE (chg (nspe))
129     ALLOCATE (bndtbl (numbond, 2))
130     ALLOCATE (visit (nmbeads), from (nmbeads))
131
132     DO i = 1, nspe
133         READ (ntraj) namspe (i), amass, rcii, chg (i), lfrzn
134     END DO
135
136     DO i = 1, nmoldef
137         READ (ntraj) nammol (i)
138     END DO
139
140     !      reading of bead species and molecule types

```

(continues on next page)

(continued from previous page)

```

141
142     nummol = 0 !counter for number of molecules
143     ibond = 0 !counter for bonds
144
145     DO i = 1, nsyst
146         READ (ntraj) global, species, molecule, chain
147         IF (global>nusyst .AND. global<=nsyst) THEN
148             ltp (global-nusyst) = species
149             ltm (global-nusyst) = molecule
150             mole (global-nusyst) = chain
151             nummol = MAX (nummol, chain)
152         END IF
153     END DO
154
155     ! reading of bond tables
156
157     IF (numbond>0) THEN
158         DO i = 1, numbond
159             READ (ntraj) bndtbl (i, 1), bndtbl (i, 2)
160         END DO
161     END IF
162
163     bndtbl = bndtbl - nusyst
164
165     ! reached end of header: find current position in file
166
167     INQUIRE (unit=ntraj, POS=currentpos)
168
169     ! determine numbers of molecules and beads per molecule type
170     nmol = 0.0_dp
171     nbdmol = 0
172     chain = 0
173     imol = 0 ! necessary to avoid out of bounds
174
175     DO i = 1, nmbeads
176         IF (mole (i) /= chain) THEN
177             chain = mole (i)
178             imol = ltm (i)
179             nmol (imol) = nmol (imol) + 1.0_dp
180         END IF
181         IF (imol > 0) nbdmol (imol) = nbdmol (imol) + 1
182     END DO
183
184     DO i = 1, nmoldef
185         rnmol = NINT (nmol (i))
186         IF (rnmol>0) THEN
187             nbdmol (i) = nbdmol (i) / rnmol
188         END IF
189     END DO
190
191     nbdmolmx = MAXVAL (nbdmol (1:nmoldef))
192
193     ! obtain connectivity information (needed only once)
194     CALL connect (nmbeads, numbond, bndtbl, nbdmolmx, visit, from)
195
196     ! Checking for charge neutrality of all molecules
197     ALLOCATE (molchg (nummol))

```

(continues on next page)

(continued from previous page)

```

198     molchg (:) = 0._dp
199
200
201     DO i = 1, nmbeads
202         imol = mole (i)
203         molchg (imol) = molchg (imol) + chg (ltp (i))
204     END DO
205
206     DO i = 1, nummol
207         IF (ABS (molchg (i)) > 1.d-16) THEN
208             WRITE (*,*) "molecule number",i," is not neutral! (The dipole moment is_
↪frame-dependent)"
209             WRITE (*,*) "its charge is=", molchg (i)
210             WRITE (*,*) "its type is=", nammol (i)
211             STOP
212         ENDIF
213     END DO
214
215     CALL check_molecules !checks that beads are labelled are as expected
216
217     !reading trajectories and computing charge dipole moments
218     ALLOCATE (dipx_box (nmoldef), dipy_box (nmoldef), dipz_box (nmoldef))
219     ALLOCATE (dip2_box (nmoldef))
220     ALLOCATE (sum_dipx_box (nmoldef), sum_dipy_box (nmoldef), sum_dipz_box_
↪(nmoldef))
221     ALLOCATE (sum_dipx_box2 (nmoldef), sum_dipy_box2 (nmoldef), sum_dipz_box2_
↪(nmoldef))
222     ALLOCATE (sum_dip2_box (nmoldef), sum_dip_box4 (nmoldef))
223     ALLOCATE (dipx_box_ave (nmoldef), dipy_box_ave (nmoldef), dipz_box_ave_
↪(nmoldef))
224     ALLOCATE (dip2_ave (nmoldef), dip2_err (nmoldef))
225     ALLOCATE (dipx_box2_ave (nmoldef), dipy_box2_ave (nmoldef), dipz_box2_ave_
↪(nmoldef))
226     ALLOCATE (dipx_box_err (nmoldef), dipy_box_err (nmoldef), dipz_box_err_
↪(nmoldef))
227     ALLOCATE (dip_box2_ave (nmoldef), dip_box2_err (nmoldef), dip_box4_ave_
↪(nmoldef))
228     ALLOCATE (sum_dip2_box2 (nmoldef))
229     ALLOCATE (gk (nmoldef), gk_err (nmoldef))
230     ALLOCATE (epsilon_r (nmoldef), epsilon_r_err(nmoldef))
231
232     ! Open and write output file(s)
233
234     nrtout = ntraj + 1
235     DO j = 1, nmoldef
236         OPEN (nrtout+j-1, file = 'dipole_'//nammol(j), status ='replace')
237     END DO
238
239     eof = .false.
240     nav = 0
241
242     sum_dipx_box = 0.0_dp
243     sum_dipy_box = 0.0_dp
244     sum_dipz_box = 0.0_dp
245
246     sum_dip2_box = 0.0_dp
247     sum_dip_box4 = 0.0_dp

```

(continues on next page)

(continued from previous page)

```

248
249     sum_dip2_box2 = 0.0_dp
250
251     sum_dipx_box2 = 0.0_dp
252     sum_dipy_box2 = 0.0_dp
253     sum_dipz_box2 = 0.0_dp
254
255     DO k = 1, numtraj
256
257         READ (ntraj, IOSTAT=ioerror) time, nbeads, dimx, dimy, dimz, shrdx, shrdy,
↪shrdz
258
259         IF (ioerror/=0) THEN
260             eof = .true.
261             IF (k==1) THEN
262                 WRITE (*,*) 'ERROR: cannot find trajectory data in HISTORY files'
263                 STOP
264             END IF
265             EXIT
266         END IF
267
268         nav = nav + 1
269         volm = dimx * dimy * dimz
270
271         READ (ntraj) readint (1:nsyst)
272         DO i = 1, nsyst
273             global = readint (i)
274             READ (ntraj) readdata (1:framesize)
275             IF (global>nusyst .AND. global<=nsyst) THEN
276                 xxx (global-nusyst) = readdata (1)
277                 yyy (global-nusyst) = readdata (2)
278                 zzz (global-nusyst) = readdata (3)
279             END IF
280         END DO
281
282         CALL compute_charge_dipoles (dipx_box, dipy_box, dipz_box, dip2_box)
283
284         DO j = 1, nmoldef
285             WRITE (nrtout+j-1, '(1p,I8,5(2x,e14.6))') k, dipx_box(j), dipy_box(j),
↪dipz_box(j), dip2_box(j) / nmol(j) , &
286                 (dipx_box(j)**2 + dipy_box(j)**2 + dipz_box(j)**2) / volm
287         END DO
288
289         sum_dipx_box = sum_dipx_box + dipx_box
290         sum_dipy_box = sum_dipy_box + dipy_box
291         sum_dipz_box = sum_dipz_box + dipz_box
292
293         sum_dipx_box2 = sum_dipx_box2 + dipx_box * dipx_box
294         sum_dipy_box2 = sum_dipy_box2 + dipy_box * dipy_box
295         sum_dipz_box2 = sum_dipz_box2 + dipz_box * dipz_box
296
297         sum_dip_box4 = sum_dip_box4 + (dipx_box**2 + dipy_box**2 + dipz_box**2)**2
298
299         sum_dip2_box = sum_dip2_box + dip2_box
300
301         sum_dip2_box2 = sum_dip2_box2 + dip2_box * dip2_box
302

```

(continues on next page)

(continued from previous page)

```

303  END DO ! end of loop over trajectories
304
305  dipx_box_ave = sum_dipx_box/REAL(nav, KIND=dp)
306  dipy_box_ave = sum_dipy_box/REAL(nav, KIND=dp)
307  dipz_box_ave = sum_dipz_box/REAL(nav, KIND=dp)
308
309  dipx_box2_ave = sum_dipx_box2/REAL(nav, KIND=dp)
310  dipy_box2_ave = sum_dipy_box2/REAL(nav, KIND=dp)
311  dipz_box2_ave = sum_dipz_box2/REAL(nav, KIND=dp)
312
313  dip2_ave = sum_dip2_box(:)/REAL(nav, KIND=dp)/REAL(nmol(:), KIND=dp)
314  dip_box2_ave = dipx_box2_ave + dipy_box2_ave + dipz_box2_ave
315  dip_box4_ave = sum_dip_box4/REAL(nav, KIND=dp)
316
317  dipx_box_err = SQRT((dipx_box2_ave - dipx_box_ave**2)/REAL(nav, KIND=dp))
318  dipy_box_err = SQRT((dipy_box2_ave - dipy_box_ave**2)/REAL(nav, KIND=dp))
319  dipz_box_err = SQRT((dipz_box2_ave - dipz_box_ave**2)/REAL(nav, KIND=dp))
320
321  dip_box2_err = sqrt((dip_box4_ave - dip_box2_ave**2)/REAL(nav, KIND=dp))
322
323  ! Error on dip2_ave is computed considering each snapshot as a sample
324
325  dip2_err = sum_dip2_box2 / dble(nav * nmol ** 2) - dip2_ave ** 2
326  dip2_err = sqrt(dip2_err / REAL(nav, KIND=dp))
327
328  WRITE (*,*) "Number of snapshots: ",nav
329  WRITE (*,*) "<P_x>, <P_y>, <P_z>:"
330  WRITE (*,98) dipx_box_ave, dipy_box_ave, dipz_box_ave
331  WRITE (*,*) "error:"
332  WRITE (*,98) dipx_box_err, dipy_box_err, dipz_box_err
333  WRITE (*,*) "<P^2>/V:"
334  WRITE (*,98) dip_box2_ave/volm
335  WRITE (*,*) "error:"
336  WRITE (*,98) dip_box2_err/volm
337  WRITE (*,*) "<p^2>:"
338  WRITE (*,98) dip2_ave
339  WRITE (*,*) "error:"
340  WRITE (*,98) dip2_err
341
342  IF (nmoldef == 1) THEN
343    gk = dip_box2_ave / dip2_ave / REAL(nmol, KIND=dp)
344    gk_err = (dip_box2_err / dip_box2_ave + dip2_err / dip2_ave) * gk
345    WRITE (*,*) "kirkwood factor:"
346    WRITE (*,98) gk
347    WRITE (*,*) "error:"
348    WRITE (*,98) gk_err
349    WRITE (*,*) "Bjerrum length?"
350    READ (*,*) bjerelec
351    epsilon_r = 1.0_dp + 4.0_dp * pi / 3.0_dp * bjerelec * dip_box2_ave / volm
352    epsilon_r_err = 4.0_dp * pi / 3.0_dp * bjerelec * dip_box2_err / volm
353    WRITE (*,*) "epsilon_r:"
354    WRITE (*,98) epsilon_r
355    WRITE (*,*) "error:"
356    WRITE (*,98) epsilon_r_err
357  ENDIF
358
359  ! Close the trajectory file

```

(continues on next page)

(continued from previous page)

```

360      CLOSE (ntraj)
361
362      !close output files
363      DO j = 1, nmoldef
364          CLOSE (nrtout+j-1)
365      END DO
366
367      DEALLOCATE (readint, readdata)
368      DEALLOCATE (namspe, nammol)
369      DEALLOCATE (xxx, yyy, zzz)
370      DEALLOCATE (ltp, ltm, mole)
371      DEALLOCATE (nmol, nbdmol)
372      DEALLOCATE (chg, molchg)
373      DEALLOCATE (dipx_box, dipy_box, dipz_box)
374      DEALLOCATE (dip2_box)
375      DEALLOCATE (sum_dipx_box, sum_dipy_box, sum_dipz_box)
376      DEALLOCATE (sum_dipx_box2, sum_dipy_box2, sum_dipz_box2)
377      DEALLOCATE (sum_dip2_box, sum_dip_box4, sum_dip2_box2)
378      DEALLOCATE (dipx_box_ave, dipy_box_ave, dipz_box_ave)
379      DEALLOCATE (dip2_ave, dip2_err)
380      DEALLOCATE (dipx_box2_ave, dipy_box2_ave, dipz_box2_ave)
381      DEALLOCATE (dipx_box_err, dipy_box_err, dipz_box_err)
382      DEALLOCATE (dip_box2_ave, dip_box2_err, dip_box4_ave)
383      DEALLOCATE (gk, gk_err)
384      DEALLOCATE (epsilon_r, epsilon_r_err)
385      DEALLOCATE (bndtbl)
386      DEALLOCATE (visit, from)
387
388 98      FORMAT (1p, 9(e13.6, 3x))
389
390      CONTAINS
391
392      SUBROUTINE check_molecules
393      !*****
394      ! subroutine to check molecular content and labelling
395      !
396      ! authors: s. chiacchiera, February 2017
397      !
398      !*****
399
400      IMPLICIT NONE
401      INTEGER i, j, k, tm, tp, imol, im, ibd
402      INTEGER mxmolsize
403      INTEGER, ALLOCATABLE :: molbeads (:,:)
404
405      mxmolsize = 0
406      DO i = 1, nmoldef
407          mxmolsize = MAX (nbdmol(i), mxmolsize)
408      END DO
409      ALLOCATE (molbeads (nmoldef, mxmolsize))
410      molbeads (:,:) = 0
411
412      imol = 0
413      ibd = 0
414      DO i = 1, nmoldef
415          DO j = 1, NINT (nmol(i))
416              imol = imol + 1

```

(continues on next page)

(continued from previous page)

```

415         DO k = 1, nbdmol(i)
416             ibd = ibd +1
417             tm = ltm (ibd)
418             tp = ltp (ibd)
419             im = mole (ibd)
420             IF (j==1) THEN
421                 molbeads (i, k) = tp
422             ELSE
423                 IF (molbeads (i, k) /= tp) THEN
424                     WRITE (*,*) "ERROR: Problem with molecular content!"
425                     STOP
426                 ENDIF
427             ENDIF
428             IF (tm/=i.OR.im/=imol) THEN
429                 WRITE (*,*) "ERROR: Problem with molecules labels!"
430                 STOP
431             ENDIF
432         END DO
433     END DO
434 END DO
435 IF (imol/=nummol) THEN
436     WRITE (*,*) "ERROR: imol and nummol differ!"
437     STOP
438 ENDIF
439 DEALLOCATE (molbeads)
440 RETURN
441 END SUBROUTINE check_molecules
442
443 SUBROUTINE compute_charge_dipoles (dipx_box, dipy_box, dipz_box, dip2_box)
444 !*****
445 ! subroutine to compute charge dipole moments
446 !
447 ! authors: m. a. seaton and s. chiacchiera, February 2017
448 !
449 ! input: xxx, yyy, zzz (at a given time step) and chg
450 ! input: visit and from (obtained using connect)
451 ! output: the x,y,z components of the total dipole, sum p_i^2/N_mol for each molecule
452 !         type (at a given time step)
453 !*****
454     IMPLICIT NONE
455     INTEGER i, j, k, tm, tp, imol, ibd, count, ipr
456     REAL(KIND=dp), DIMENSION(nmoldef) :: dipx_box, dipy_box, dipz_box
457     REAL(KIND=dp), DIMENSION(nmoldef) :: dip2_box
458     REAL(KIND=dp) :: x, y, z, dx, dy, dz, xpre, ypre, zpre
459     REAL(KIND=dp) :: dipx, dipy, dipz, dip2
460     REAL(KIND=dp), DIMENSION(nmbeads) :: xabs, yabs, zabs
461
462     dipx_box (:) = 0._dp
463     dipy_box (:) = 0._dp
464     dipz_box (:) = 0._dp
465
466     dip2_box (:) = 0._dp
467
468     imol = 0
469     count = 0
470     ! xabs = 0._dp ! just to keep it clean
471     ! yabs = 0._dp

```

(continues on next page)

(continued from previous page)

```

472      ! zabs = 0._dp
473
474      DO i = 1, nmoldef
475          tm = i
476          DO j = 1, NINT (nmol(i))
477              imol = imol + 1
478
479              dipx = 0._dp ! dipole of a SINGLE molecule
480              dipy = 0._dp
481              dipz = 0._dp
482
483              DO k = 1, nbdmol(i)
484                  count = count + 1
485                  ibd = visit (count)
486                  ipr = from (count)
487
488                  IF (ipr /= 0) THEN
489                      xpre = xabs (ipr)
490                      ypre = yabs (ipr)
491                      zpre = zabs (ipr)
492                  ELSE
493                      IF (k == 1) THEN
494                          xpre = 0._dp
495                          ypre = 0._dp
496                          zpre = 0._dp
497                      ELSE
498                          WRITE (*,*) "Unconnected molecule!"
499                          STOP
500                      ENDIF
501                  ENDIF
502
503                  tp = ltp (ibd)
504
505                  dx = xxx (ibd) - xpre
506                  dy = yyy (ibd) - ypre
507                  dz = zzz (ibd) - zpre
508
509                  dx = dx - dimx * ANINT (dx/dimx)
510                  dy = dy - dimy * ANINT (dy/dimy)
511                  dz = dz - dimz * ANINT (dz/dimz)
512
513                  x = xpre + dx
514                  y = ypre + dy
515                  z = zpre + dz
516
517
518                  dipx = dipx + x * chg (tp)
519                  dipy = dipy + y * chg (tp)
520                  dipz = dipz + z * chg (tp)
521
522                  xabs (ibd) = x
523                  yabs (ibd) = y
524                  zabs (ibd) = z
525
526              END DO
527
528      dipx_box (tm) = dipx_box (tm) + dipx

```

(continues on next page)

(continued from previous page)

```

529         dipy_box (tm) = dipy_box (tm) + dipy
530         dipz_box (tm) = dipz_box (tm) + dipz
531
532         dip2 = dipx * dipx + dipy * dipy + dipz * dipz
533
534         dip2_box (tm) = dip2_box (tm) + dip2
535
536         END DO
537     END DO
538
539     IF (imol/=nummol) THEN
540         WRITE (*,*) "ERROR: imol and nummol differ!"
541         STOP
542     ENDIF
543
544     RETURN
545 END SUBROUTINE compute_charge_dipoles
546
547 End PROGRAM gen_dipole
548 SUBROUTINE connect (nbeads, nbonds, bndtbl, mxmolsize, visit, from)
549 !*****
550 ! Analyzes all the bonds (bndtbl) to obtain a schedule (visit, from)
551 ! to visit the beads so that each cluster is visited along a connected
552 ! path. "visit" gives the order to include beads, "from" gives the bead
553 ! to attach them to.
554 ! (Note: vocabulary from infection propagation used to move along
555 ! clusters)
556 !
557 ! author: s. chiacchiera, February 2017
558 ! amended: m. a. seaton, January 2021
559 !*****
560     IMPLICIT none
561     INTEGER, INTENT (INOUT) :: bndtbl (nbonds,2)
562     INTEGER, INTENT (IN) :: nbeads, nbonds
563     INTEGER, INTENT (IN) :: mxmolsize
564     INTEGER :: ic, i, k, nn, nclu, nper, lab, ref, count
565     INTEGER, ALLOCATABLE :: firstnn (:), lastnn (:), deg (:)
566     INTEGER, ALLOCATABLE :: labnn (:)
567     INTEGER, ALLOCATABLE :: state (:)
568     INTEGER, ALLOCATABLE :: perlab (:), perref (:)
569     INTEGER, ALLOCATABLE :: nchist (:)
570     INTEGER, INTENT (OUT) :: visit (nbeads), from (nbeads)
571
572     ALLOCATE (firstnn (nbeads), lastnn (nbeads), deg (nbeads), labnn (2*nbonds))
573     ALLOCATE (state (nbeads))
574     ALLOCATE (perlab (nbeads), perref (nbeads))
575     ALLOCATE (nchist (mxmolsize))
576     !-----
577     CALL organize (nbeads, nbonds, labnn, firstnn, lastnn, deg)
578     !-----
579     state (:) = 0
580     nchist (:) = 0
581     visit (:) = 0
582     from (:) = 0
583     count = 0
584     !-----
585     ic = 0

```

(continues on next page)

(continued from previous page)

```

586      !-----
587      DO WHILE (ic < nbeads) ! ic = label of bead used to "grow" a cluster
588          ic = ic + 1
589          IF( state (ic) /= 0) THEN
590              WRITE (*,*) "ERROR: labels are not as expected!"
591              STOP
592          END IF
593          nclu = 1
594          count = count + 1
595          visit (ic) = ic
596          IF (deg (ic) == 0) THEN
597              state (ic) = -1
598              IF (nclu <= mxmolsize) nchist (nclu) = nchist (nclu) +1
599              CYCLE
600          END IF
601          state (ic) = 1 ! ic is "infected"
602
603          ! nearest neighbours of ic are marked as "goint to be infected" -> a.k.a.
604      ↪perimeter
605          nper = 0
606          perlab (:) = 0
607          perref (:) = 0
608          DO k = firstnn (ic), lastnn (ic)
609              nn = labnn (k)
610              IF( state (nn) /= 0) THEN
611                  WRITE (*,*) "ERROR: labels are not as expected!"
612                  STOP
613              END IF
614              nper = nper + 1
615              perlab (nper) = nn !new bead in perimeter
616              perref (nper) = ic !its reference bead (origin of the link)
617              state (nn) = 2
618          END DO
619          state (ic) = 3 ! ic is "dead"
620
621          DO WHILE (nper > 0)
622              i = 1 ! pick a bead of "perimeter" to be analyzed
623              lab = perlab (i)
624              ref = perref (i)
625              perlab (i) = perlab (nper)
626              perref (i) = perref (nper)
627              nper = nper - 1
628              IF (state (lab) == 3) THEN
629                  CYCLE
630              END IF
631              state (lab) = 1 ! "lab" is added to the cluster
632              nclu = nclu + 1
633              count = count + 1
634              visit (count) = lab
635              from (count) = ref
636
637              DO k = firstnn (lab), lastnn (lab) ! check nn of newly added
638                  nn = labnn (k)
639                  IF( (state (nn) == 2) .OR. (state (nn) == 3)) CYCLE
640                  nper = nper + 1
641                  perlab (nper) = nn !new bead in perimeter
642                  perref (nper) = lab !its reference bead (origin of the link)

```

(continues on next page)

(continued from previous page)

```

642         state (nn) = 2
643     END DO
644     state (lab) = 3
645
646     END DO
647     nchist (nclu) = nchist (nclu) + 1
648     ic = ic + nclu - 1 ! prepare ic for the next cluster
649 END DO
650 WRITE (*,*) "nchist: ", nchist
651 !-----
652 DEALLOCATE (firstnn, lastnn, deg, labnn)
653 DEALLOCATE (state)
654 DEALLOCATE (perlab, perref)
655 DEALLOCATE (nchist)
656 RETURN
657 !-----
658 CONTAINS
659 !-----
660 SUBROUTINE organize (N, NL, labnn, firstnn, lastnn, deg)
661     !*****
662     ! Analyzes the bonds (bndtbl) to obtain the degree (=number of bonds)
663     ! of each bead, and the nearest neighbours list.
664     ! N in the number of beads (vertices) and NL of bonds (links).
665     !
666     ! author: s. chiacchiera, February 2017
667     !*****
668     IMPLICIT none
669     INTEGER, INTENT(IN) :: N, NL
670     INTEGER :: i,l,count_lab, i1,i2
671     INTEGER, DIMENSION (N), INTENT(OUT) :: deg
672     INTEGER, DIMENSION (N), INTENT(OUT) :: firstnn, lastnn
673     INTEGER, DIMENSION (2*NL), intent(OUT) :: labnn
674
675     deg(:)=0
676     firstnn(:)=0
677     lastnn(:)=0
678     labnn(:)=0
679
680     count_lab=0
681
682     DO i=1,N
683         DO l=1,NL
684             IF (bndtbl(l,1).EQ.i) THEN
685                 deg(i)=deg(i)+1
686                 count_lab=count_lab+1
687                 labnn(count_lab)=bndtbl(l,2)
688             ENDIF
689             IF (bndtbl(l,2).EQ.i) THEN
690                 deg(i)=deg(i)+1
691                 count_lab=count_lab+1
692                 labnn(count_lab)=bndtbl(l,1)
693             ENDIF
694         END DO
695     END DO
696
697     i1=1
698     i2=0

```

(continues on next page)

(continued from previous page)

```

699      DO i=1,N
700          IF (deg (i)==0) CYCLE
701          firstnn(i)=i1
702          i2=i1+deg (i) -1
703          lastnn(i)=i2
704          i1=i2+1
705      END DO
706      RETURN
707
708      END SUBROUTINE organize
709      !-----
710      END SUBROUTINE connect

```

Formatting the HISTORY files of DL_MESO_DPD

Software Technical Information

Language Fortran 2003

Licence BSD

Documentation Tool RST and LaTeX-generated .pdf

Application Documentation [Click to download the manual with more details](#)

Relevant Training Material See the Testing section

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*

Purpose of Module

This module `format_history.f90` is a post-processing utility for DL_MESO_DPD, the Dissipative Particle Dynamics (DPD) code from the [DL_MESO](#) package.

It converts the trajectory (HISTORY) file from *unformatted* to a human readable form, (optionally) including explanatory comments about all the quantities. This module is mainly for learning/checking purposes. The first aim is to help the user to check that the system was prepared as intended (e.g., showing all the bead properties and initial positions, all the bonds etc.). The idea is to use it on small systems when familiarizing with the structure of input files needed for the simulation. Secondly, it can be used as a starting point for user-defined analyses of trajectories.

Background Information

The base code for this module is DL_MESO_DPD, the Dissipative Particle Dynamics code from the mesoscopic simulation package [DL_MESO](#), developed by M. Seaton at Daresbury Laboratory. This open source code is available

from STFC under both academic (free) and commercial (paid) licenses. The module is to be used with DL_MESO in its currently released version, version 2.7 (dating December 2018).

A version of the module that works with HISTORY files generated by the previous version of DL_MESO, version 2.6 (dating November 2015), can be found in the `old-v2.6` directory.

Testing

The present module is compiled with the available Fortran 2003 compiler, e.g.:

```
gfortran -o format.exe format_history.f90
```

and the executable must be in the same directory of the HISTORY file to be analyzed. To test the module, run the simulation with the *toy* input files given in the following. (Note that these files contain commented lines as suggestions for further tests.) For the CONTROL file

```
Simple test

volume 3.0 3.0 3.0
temperature 1.0
cutoff 1.0

timestep 0.01
steps 6
equilibration steps 2
traj 2 2 0
stats every 2
stack size 2
print every 2
job time 100.0
close time 10.0

#surface shear y
#surface frozen x
#surface hard x

ensemble nvt mdvv

finish
```

and for the FIELD file

```
Simple test

SPECIES 3
A 1.0 0.0 1 0
B 1.0 0.0 0 0
C 1.0 0.0 0 0

MOLECULES 2
AB
nummols 1
beads 2
A 0.0 0.0 0.0
B 0.1 0.0 0.0
bonds 1
harm 1 2 5.0 0.0
```

(continues on next page)

(continued from previous page)

```

finish
AC
nummols 1
beads 2
A      0.0 0.0 0.0
C      0.1 0.0 0.0
bonds 1
harm 1 2 3.0 0.0
finish

INTERACTIONS 3
A      A      dpd 25.0 1.0 4.5
B      B      dpd 25.0 1.0 4.5
C      C      dpd 25.0 1.0 4.5

#EXTERNAL
#shear 3.0 0.0 0.0

CLOSE

```

After analyzing the trajectories, for a serial run (i.e., on a single processor core) and for `lcomm`, `lmcheck` and sorted all set to `.TRUE.`, this output should be printed on the screen

```

# Check of beads: i, ltp(i), ltm(i), mole(i)
      1          1          0          0
      2          1          1          1
      3          2          1          1
      4          1          2          2
      5          3          2          2

# Check of molecules: nammol(i), nbdmol(i), nbomol(i), nmol(i)
AB              2          1          1
AC              2          1          1
# Total number of molecules =                2
# Check of bonds: bndbtl(i,1), bndbtl(i,2)
      2          3
      4          5

```

and the HISTORY-F file should be

```

# Simulation name:
Simple test
# nspe, nmoldef, nusyst, nsyst, numbond
      3          2          1          5          2
# keytrj, srfx, srfy, srfz
      0          0          0          0
# SPECIES:
# namspe, amass, rcii, chge, lfrzn
A      1.000      1.000      0.000      0
B      1.000      1.000      0.000      0
C      1.000      1.000      0.000      0
# MOLECULES:
# nammol
AB
AC
# BEADS:
# global, species, molecule, chain

```

(continues on next page)

(continued from previous page)

```

1      1      0      0
2      1      1      1
3      2      1      1
4      1      2      2
5      3      2      2

# BONDS:
# extremes of the bond
2      3
4      5

# --- TRAJECTORIES --- (key = 0 )
# mglobal, x, y, z
# time, nbeads, dimx, dimy, dimz, shrdx, shrdy, shrdz
0.000      5      3.000      3.000      3.000      0.000      0.
→000      0.000
# snapshot number: 1
1 -5.594619E-03 -1.752832E-03 6.889737E-03
2 -1.691781E-01 6.786290E-01 1.166801E-02
3 -2.345695E-01 7.399172E-01 -1.447126E-01
4 5.198776E-01 -8.222446E-01 1.038467E-02
5 4.631501E-01 -8.058678E-01 1.464801E-01
# time, nbeads, dimx, dimy, dimz, shrdx, shrdy, shrdz
0.020      5      3.000      3.000      3.000      0.000      0.
→000      0.000
# snapshot number: 2
1 -1.061544E-02 -6.284880E-03 1.390368E-02
2 -1.395560E-01 6.796778E-01 4.361174E-02
3 -2.443224E-01 7.728033E-01 -1.800995E-01
4 5.280036E-01 -8.191712E-01 -3.253643E-02
5 4.401757E-01 -8.383441E-01 1.858304E-01
# time, nbeads, dimx, dimy, dimz, shrdx, shrdy, shrdz
0.040      5      3.000      3.000      3.000      0.000      0.
→000      0.000
# snapshot number: 3
1 -1.483900E-02 -1.635085E-02 2.109335E-02
2 -1.083779E-01 6.830112E-01 8.086194E-02
3 -2.571191E-01 8.101380E-01 -2.211317E-01
4 5.390852E-01 -8.157553E-01 -8.230434E-02
5 4.149363E-01 -8.723621E-01 2.321906E-01

```

If the simulation is run in parallel, the particles may not necessarily be written to the HISTORY file in order of particle index, but the module can sort the particles in each trajectory snapshot before printing to the HISTORY-F file.

Source Code

```

1 PROGRAM format_history
2 !*****
3 !
4 ! module to format dl_meso HISTORY files
5 !
6 ! authors - m. a. seaton & s. chiacchiera, february 2017 (amended january 2021)
7 !
8 !*****
9 IMPLICIT none
10 INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND (15, 307)
11 INTEGER, PARAMETER :: li = SELECTED_INT_KIND (12)

```

(continues on next page)

(continued from previous page)

```

12  INTEGER, PARAMETER :: ntraj=10, nuser=5
13  INTEGER, PARAMETER :: endversion = 1
14
15  CHARACTER(80) :: text
16  CHARACTER(8), ALLOCATABLE :: namspe (:), nammol (:)
17
18  INTEGER, ALLOCATABLE :: ltp (:), ltm (:), mole (:), bndtbl (:,:)
19  INTEGER, ALLOCATABLE :: nbdmol (:), nbomol (:), readint (:), globindex (:)
20  INTEGER :: chain, imol, ioerror, i, k, nmoldef, numframe
21  INTEGER :: nspe, nbeads, nusyst, nsyst, global, species, molecule, numbond
22  INTEGER :: nummol, lfrzn, rnmol, keytrj, srfx, srfy, srfz
23  INTEGER :: bead1, bead2
24  INTEGER :: nform
25  INTEGER :: endver, Dlen, nstep, framesize, lend, leni
26  INTEGER(KIND=li) :: filesize, mypos, headerpos, currentpos, lend_li, leni_li,
↳ framesizeli, numbeadsli
27
28  REAL(KIND=dp), ALLOCATABLE :: nmol (:), readdata (:)
29  REAL(KIND=dp) :: dimx, dimy, dimz, shrdx, shrdy, shrdz
30  REAL(KIND=dp) :: amass, rcii, chge
31  REAL(KIND=dp) :: time
32
33  LOGICAL :: eof, lcomm, lmcheck, swapend, bigend, sorted
34
35  ! Switches for commenting, checking molecules and sorting particles in output
36
37  lcomm = .TRUE.
38  lmcheck = .TRUE.
39  sorted = .TRUE.
40
41  ! determine number of bytes for selected double precision kind
42  ! (the default SELECTED_REAL_KIND (15, 307) should return 8 bytes)
43
44  lend = STORAGE_SIZE (1.0_dp) / 8
45  leni = BIT_SIZE (1) / 8
46  lend_li = INT (lend, KIND=li)
47  leni_li = INT (leni, KIND=li)
48
49  ! check endianness of machine
50
51  bigend = (IACHAR(TRANSFER(1,"a"))==0)
52
53  ! Determine if HISTORY file exists, which endianness to use,
54  ! if type of real is correct
55
56  INQUIRE (file = 'HISTORY', EXIST = eof)
57  IF (.NOT. eof) THEN
58    PRINT *, "ERROR: cannot find HISTORY file"
59    STOP
60  END IF
61
62  OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted', status_
↳ = 'unknown')
63
64  swapend = .false.
65  READ (ntraj) endver, Dlen
66

```

(continues on next page)

(continued from previous page)

```

67     IF (endver/=endversion) THEN
68         swapend = .true.
69         CLOSE (ntraj)
70         IF (bigend) THEN
71             OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',
↪status = 'unknown', convert = 'little_endian')
72         ELSE
73             OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',
↪status = 'unknown', convert = 'big_endian')
74         END IF
75         READ (ntraj) endver, Dlen
76         IF (endver/=endversion) THEN
77             PRINT *, "ERROR: corrupted HISTORY file or created with incorrect version
↪of DL_MESO"
78             STOP
79         END IF
80     END IF
81
82     IF (Dlen/=lend) THEN
83         PRINT *, "ERROR: incorrect type of real number used in HISTORY file"
84         PRINT *, "      recompile format_history.f90 with reals of ", Dlen, " bytes"
85         STOP
86     END IF
87
88     ! Open the output file
89     nform = ntraj + 1
90     OPEN (nform, file = 'HISTORY'//"F", status = 'replace')
91
92     !      read file size, number of frames and timestep numbers
93
94     READ (ntraj) filesize, numframe, nstep
95
96     ! read the number of beads, molecules and bonds
97     ! Arrays are filled with names of particles and molecules: if checking
↪molecules,
98     ! arrays for species, molecule types etc. also filled
99
100    READ (ntraj) text
101    READ (ntraj) nspe, nmoldef, nusyst, nsyst, numbond, keytrj, srfx, srfy, srfz
102
103    IF (lcomm) WRITE (nform,*) "# Simulation name:"
104    WRITE (nform,*) text
105
106    IF (lcomm) WRITE (nform,*) "# nspe, nmoldef, nusyst, nsyst, numbond"
107    WRITE (nform,*) nspe, nmoldef, nusyst, nsyst, numbond
108    IF (lcomm) WRITE (nform,*) "# keytrj, srfx, srfy, srfz"
109    WRITE (nform,*) keytrj, srfx, srfy, srfz
110
111    framesize = (keytrj+1) * 3
112    ALLOCATE (namspe (nspe), nammol (nmoldef), globindex (nsyst), readint (nsyst),
↪readdata (framesize))
113    IF (lmcheck) THEN
114        ALLOCATE (ltp (1:nsyst), ltm (1:nsyst), mole (1:nsyst))
115        ALLOCATE (nmol (1:nmoldef), nbdmol (1:nmoldef), nbomol (1:nmoldef))
116        ALLOCATE (bndtbl (numbond, 2))
117    END IF
118

```

(continues on next page)

(continued from previous page)

```

119 IF (lcomm) WRITE (nform,*) "# SPECIES:"
120 IF (lcomm) WRITE (nform,*) "# namspe, amass, rcii, chge, lfrzn"
121 DO i = 1, nspe
122   READ (ntraj) namspe (i), amass, rcii, chge, lfrzn
123   WRITE (nform,96) namspe (i), amass, rcii, chge, lfrzn
124 END DO
125
126 IF (nmoldef>0) THEN
127   IF (lcomm) WRITE (nform,*) "# MOLECULES:"
128   IF (lcomm) WRITE (nform,*) "# nammol"
129   DO i = 1, nmoldef
130     READ (ntraj) nammol (i)
131     WRITE (nform,*) nammol (i)
132   END DO
133 END IF
134
135 ! (if required) read and fill arrays with properties of beads and molecules
136
137 nummol = 0 ! counter for number of molecules
138
139 IF (lcomm) WRITE (nform,*) "# BEADS:"
140 IF (lcomm) WRITE (nform,*) "# global, species, molecule, chain"
141 IF (lmcheck) THEN
142   ! Build ltp, ltm, mole
143   DO i = 1, nsyst
144     READ (ntraj) global, species, molecule, chain
145     ltp (global) = species
146     ltm (global) = molecule
147     mole (global) = chain
148     nummol = MAX (nummol, chain)
149     WRITE (nform,*) global, species, molecule, chain
150   END DO
151 ELSE
152   DO i = 1, nsyst
153     READ (ntraj) global, species, molecule, chain
154     WRITE (nform,*) global, species, molecule, chain
155   END DO
156 END IF
157
158 IF (numbond>0) THEN
159   IF (lcomm) WRITE (nform,*) "# BONDS:"
160   IF (lcomm) WRITE (nform,*) "# extremes of the bond"
161   IF (lmcheck) THEN
162     ! Build bndtbl
163     DO i = 1, numbond
164       READ (ntraj) bead1, bead2
165       bndtbl (i, 1) = bead1
166       bndtbl (i, 2) = bead2
167       WRITE (nform,*) bead1, bead2
168     END DO
169   ELSE
170     DO i = 1, numbond
171       READ (ntraj) bead1, bead2
172       WRITE (nform,*) bead1, bead2
173     END DO
174   END IF
175 END IF

```

(continues on next page)

(continued from previous page)

```

176
177 !      reached end of header: find current position in file
178
179 INQUIRE (unit=ntraj, POS=headerpos)
180 framesizeli = INT (framesize, KIND=li)
181 numbeadsli = INT (nsyst, KIND=li)
182
183 IF (lmcheck) THEN
184 ! determine numbers of molecules, beads and bonds per molecule type
185 nmol = 0.0_dp
186 nbdmol = 0
187 nbomol = 0
188 chain = 0
189 imol = 0 ! necessary to avoid out of bounds
190
191 DO i = 1, nsyst
192   IF (mole (i) /= chain) THEN
193     chain = mole (i)
194     imol = ltm (i)
195     nmol (imol) = nmol (imol) + 1.0_dp
196   END IF
197   IF (imol > 0) nbdmol (imol) = nbdmol (imol) + 1
198 END DO
199
200 DO i = 1, numbond
201   imol = ltm (bndtbl (i,1))
202   nbomol (imol) = nbomol (imol) + 1
203 END DO
204
205 DO i = 1, nmoldef
206   rnmol = NINT (nmol (i))
207   IF (rnmol>0) THEN
208     nbdmol (i) = nbdmol (i) / rnmol
209     nbomol (i) = nbomol (i) / rnmol
210   END IF
211 END DO
212
213 ! Write to std output the arrays built
214 WRITE (*,*) "# Check of beads: i, ltp(i), ltm(i), mole(i)"
215 DO i = 1, nsyst
216   WRITE(*,*) i, ltp (i), ltm (i), mole (i)
217 END DO
218
219 !Check of molecule beads and numbers
220 IF (nmoldef>0) THEN
221   WRITE (*,*) "# Check of molecules: nammol(i), nbdmol(i), nbomol(i),
↪nmol(i) "
222   DO i = 1, nmoldef
223     WRITE (*,*) nammol (i), nbdmol (i), nbomol (i), NINT(nmol(i))
224   END DO
225   WRITE (*,*) "# Total number of molecules = ",nummol
226 END IF
227
228 ! Write to std output bndtbl
229 IF (numbond > 0) THEN
230   WRITE (*,*) "# Check of bonds: bndbtl(i,1), bndbtl(i,2)"
231   DO i = 1, numbond

```

(continues on next page)

(continued from previous page)

```

232         WRITE (*,*) bndtbl (i,1), bndtbl (i,2)
233     END DO
234 END IF
235 END IF
236
237 ! Now read in trajectories
238
239 eof = .false.
240
241 IF (lcomm) WRITE (nform,*) "# --- TRAJECTORIES --- (key =", keytrj,")"
242 SELECT CASE (keytrj)
243 CASE (0)
244     IF (lcomm) WRITE (nform,*) "# mglobal, x, y, z"
245 CASE(1)
246     IF (lcomm) WRITE (nform,*) "# mglobal, x, y, z, vx, vy, vz"
247 CASE(2)
248     IF (lcomm) WRITE (nform,*) "# mglobal, x, y, z, vx, vy, vz, fx, fy, fz"
249 END SELECT
250
251 DO k = 1, numframe
252
253     currentpos = headerpos + INT (k-1, KIND=li) * ((7_li + numbeadsli *
↪framesizeli) * lend_li + (1_li + numbeadsli) * leni_li)
254     READ (ntraj, POS=currentpos, IOSTAT=ioerror) time, nbeads, dimx, dimy, dimz,
↪shrdx, shrdy, shrdz
255
256     IF (ioerror/=0) THEN
257         eof = .true.
258         IF (k==1) THEN
259             PRINT *, 'ERROR: cannot find trajectory data in HISTORY file'
260             STOP
261         END IF
262         EXIT
263     END IF
264
265     IF (lcomm) WRITE (nform,*) "# time, nbeads, dimx, dimy, dimz, shrdx, shrdy,
↪shrdz"
266     WRITE (nform,98) time, nbeads, dimx, dimy, dimz, shrdx, shrdy, shrdz
267
268     IF (lcomm) WRITE (nform,*) "# snapshot number:", k
269
270     IF (sorted) THEN
271
272         READ (ntraj) readint (1:nbeads)
273         CALL quicksort_integer_indexed (readint, 1, nbeads, globindex)
274         DO i = 1, nbeads
275             global = globindex (i)
276             mypos = currentpos + leni_li * (1_li + numbeadsli) + (7_li + INT (global-
↪1, KIND=li) * framesizeli) * lend_li
277             READ (ntraj, POS=mypos) readdata (1:framesize)
278             WRITE (nform,99) global, readdata (1:framesize)
279         END DO
280
281     ELSE
282
283         READ (ntraj) globindex (1:nbeads)
284         DO i = 1, nbeads

```

(continues on next page)

(continued from previous page)

```

285         READ (ntraj, POS=mypos) readdata (1:framesize)
286         WRITE (nform,99) global, readdata (1:framesize)
287     END DO
288
289     END IF
290
291 END DO
292
293     ! Close the trajectory file
294     CLOSE (ntraj)
295
296     ! close the output file
297     CLOSE (nform)
298
299     DEALLOCATE (readint, readdata, globindex)
300     DEALLOCATE (namspe, nammol)
301     IF (lmcheck) DEALLOCATE (ltp, ltm, mole, nmol, nbdmol, bndtbl, nbomol)
302
303 99    FORMAT (I10,2x,1p,9(e13.6,3x))
304 98    FORMAT (f10.3,3x,1x,I10,6(f10.3,3x))
305 96    FORMAT (A9,3x,3(f10.3,3x),I2)
306
307 CONTAINS
308
309     SUBROUTINE quicksort_integer_indexed (list, stride, n, indices)
310
311     ! *****
312     !
313     !     sort integers in array into numerical order, recording original
314     !     positions of values (routine to prepare indices array)
315     !
316     !     copyright ukri stfc daresbury laboratory
317     !     authors - m. a. seaton august 2013
318     !
319     ! *****
320
321     INTEGER, INTENT (INOUT) :: list (:)
322     INTEGER, INTENT (IN) :: stride, n
323     INTEGER, INTENT (OUT) :: indices (:)
324     INTEGER :: i
325
326     DO i = 1, n
327         indices (i) = i
328     END DO
329
330     CALL qsort_integer (list, indices, stride, 1, n)
331
332     END SUBROUTINE quicksort_integer_indexed
333
334     RECURSIVE SUBROUTINE qsort_integer (list, index, stride, low, high)
335
336     ! *****
337     !
338     !     sort integers in array into numerical order, recording original
339     !     positions of values
340     !
341     !     copyright ukri stfc daresbury laboratory

```

(continues on next page)

(continued from previous page)

```

342  !      authors - m. a. seaton august 2013
343  !
344  ! *****
345
346  INTEGER, INTENT (INOUT) :: list (:), index (:)
347  INTEGER, INTENT (IN) :: low, high
348  INTEGER, INTENT (IN) :: stride
349  INTEGER :: i, j, k, reference, temp
350
351  IF (high < low + 6) THEN
352
353  !      resort to bubble sort for very small lists (5 items or fewer)
354
355      DO i = low, high - 1
356          DO j = i+1, high
357              IF (list (stride * (i - 1) + 1) > list (stride * (j - 1) + 1)) THEN
358                  DO k = 1, stride
359                      temp = list (stride * (i - 1) + k)
360                      list (stride * (i - 1) + k) = list (stride * (j - 1) + k)
361                      list (stride * (j - 1) + k) = temp
362                  END DO
363                  temp = index (i)
364                  index (i) = index (j)
365                  index (j) = temp
366              END IF
367          END DO
368      END DO
369
370  ELSE
371
372  !      apply partition-based sort
373
374      reference = list (stride * ((low+high)/2 - 1) + 1)
375      i = low - 1
376      j = high + 1
377      DO
378          DO
379              i = i + 1
380              IF (list (stride * (i-1) + 1) >= reference) EXIT
381          END DO
382          DO
383              j = j - 1
384              IF (list (stride * (j-1) + 1) <= reference) EXIT
385          END DO
386          IF (i < j) THEN
387              DO k = 1, stride
388                  temp = list (stride * (i-1) + k)
389                  list (stride * (i-1) + k) = list (stride * (j-1) + k)
390                  list (stride * (j-1) + k) = temp
391              END DO
392              temp = index (i)
393              index (i) = index (j)
394              index (j) = temp
395          ELSE IF (i == j) THEN
396              i = i + 1
397              EXIT
398          ELSE

```

(continues on next page)

(continued from previous page)

```

399         EXIT
400     END IF
401 END DO

402
403     IF (low<j) CALL qsort_integer (list, index, stride, low, j)
404     IF (i<high) CALL qsort_integer (list, index, stride, i, high)
405
406 END IF
407
408 END SUBROUTINE qsort_integer
409
410 END PROGRAM format_history

```

Autocorrelation functions of charge dipole moments in DL_MESO_DPD

Software Technical Information

Language Fortran 2003

Licence BSD

Documentation Tool RST and LaTeX-generated .pdf file

Application Documentation [Click to download the manual with more details](#)

Relevant Training Material See the Testing section

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*

Purpose of Module

This module, `gen_dipoleaf.f90`, is a generalization of the `dipoleaf.f90` post-processing utility of DL_MESO_DPD, the Dissipative Particle Dynamics (DPD) code from the [DL_MESO](#) package.

It processes the trajectory (HISTORY) files to obtain the charge dipole moments of all the (neutral) molecules in the system, and subsequently computes the dipole autocorrelation functions (DAFs) for each molecule type. It produces a file *DIPAFDAT* containing both the un-normalized and normalized DAFs, and, optionally, a file *DIPAFFFT* containing the Fourier transform (FT) of the latter.

The module can be applied to systems including molecules with a generic charge structure, as long as each molecule is neutral (otherwise the charge dipole moment would be frame-dependent)¹.

¹ Disambiguation on the concept of molecule. In DL_MESO a *defined molecule* is a set of beads that can be bonded together or not. For the purpose of this module it is *required* that each molecule is a connected cluster (via stretching bonds). In fact, this - together with the reasonable assumption that each stretching bond cannot be stretched to more than half the system linear size - allows us to univocally define the charge dipole moment of each molecule.

CAVEAT: this module only analyzes molecular trajectories. If a charged molecule is present, an error message will be given, while unbonded charges will not be detected and erroneous results may be obtained. Therefore please keep in mind **to not apply** this module to systems with unbonded charges.

The charge dipole moment of a neutral molecule is $\vec{p}_{mol} = \sum_{i \in mol} q_i \vec{r}_i$ where \vec{r}_i are the bead positions and q_i their charges. The total charge dipole moment of the simulated volume V is $\vec{P} = \sum_{mol \in V} \vec{p}_{mol}$. If more than one molecular species are present, one can split \vec{P} into the different species contributions: $\vec{P} = \sum_{j=1}^{nmoldef} \vec{P}_j$.

Given a scalar quantity A , its non-normalized autocorrelation function (AF) is $C_{AA}(t) = \langle A(0)A(t) \rangle$, where $\langle \dots \rangle$ indicates an average over trajectories. The normalized one is $c_{AA}(t) = \frac{\langle A(0)A(t) \rangle}{\langle A(0)A(0) \rangle} = \frac{C_{AA}(t)}{C_{AA}(0)}$.

Here for the j -th molecular species we replace A with \vec{P}_j , and the product with a scalar product. In this case the average over trajectories translates into a sum over different time origins.

The output file *DIPAFDAT* contains the DAFs for each molecular species and, at the end of the file, the DAF for the system *total* charge dipole moment \vec{P} . The output file *DIPAFFFT* contains the FT of these functions, in the same order.

More in detail, the header of the file *DIPAFDAT* contains the simulation title and a line with the number of snapshots in HISTORY and of those used for the AFs (*naf*). Then a block follows for each molecule type, started by the *{molecule name}*, then three columns of data, $t, C_{\vec{P}\vec{P}}, c_{\vec{P}\vec{P}}$. It is intended that in any block only the contribution to \vec{P} from a given species is used. The last block is called *{all species}* and refers to the full system charge dipole moment.

The header of the file *DIPAFFFT* is as for *DIPAFDAT* (notice that the number of points for the FT is also set equal to *naf*). Then a block follows for each molecule type, started by the molecule name, then three columns of data, $\omega, \Re[\hat{c}_{\vec{P}\vec{P}}(\omega)], \Im[\hat{c}_{\vec{P}\vec{P}}(\omega)]$, where \hat{c} is the discrete FT of $c(t)$.

Possible uses of the output file are: to analyze it to obtain the decay time of autocorrelations, which can be used to define an efficient sampling of the simulation; to compare it with the analogous microscopic one obtained for individual molecules (see *Autocorrelation functions of individual charge dipole moments in DL_MESO_DPD*).

Background Information

The base code for this module is DL_MESO_DPD, the Dissipative Particle Dynamics code from the mesoscopic simulation package DL_MESO, developed by M. Seaton at Daresbury Laboratory. This open source code is available from STFC under both academic (free) and commercial (paid) licenses. The module is to be used with DL_MESO in its most recently released version, version 2.7 (dating December 2018).

A variant of this module for use with a previous version of DL_MESO, version 2.6 (dating November 2015), can be found in the `old-v2.6` directory³.

The present module also requires the library FFTW (version 3.x) to be installed.

Testing

The present module `gen_dipoleaf.f90` is compiled with the available Fortran 2003 compiler, e.g.:

```
gfortran -o gen_dipoleaf.exe gen_dipoleaf.f90 -I/usr/local/include -L/usr/local/lib -lfftw3 -lm
```

where `-I` indicates the location of the FFTW include file `fftw3.f03` and `-L` points to the directory containing the FFTW library files. The above command gives the most likely locations for these files, although these may need to be adjusted if FFTW has been installed somewhere else on your machine.

² M. P. Allen and D. J. Tildesley, "Computer simulation of liquids", Oxford University Press, Oxford (1987).

³ A small change to specifying charge smearing schemes and lengths in CONTROL files has been made since version 2.6: the `old-v2.6` folder includes the CONTROL file for the test shown here that will work with this version of DL_MESO.

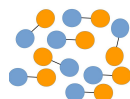
The executable must be in the same directory as the HISTORY file to be analyzed. The user is asked to provide the maximum number of snapshots to be used for the AFs (*naf*) and a switch for the Fourier transform: 1 for *yes*, 0 (or any other integer) for *no*.

To input these parameters one can either enter them from the keyboard or write them into a text file (say, *input.txt*), one per line in the right order, and run the program in this way:

```
gen_dipoleaf.exe < input.txt
```

Test: water in oil

Test case: a dimer solvent



Here we describe a physical system in which the software modules dealing with charge dipole moments in DPD simulations can be tested. It is a *polarizable fluid* made of harmonically bonded dimers $(+q, -q)$, pictorially represented on the left (not in scale). Fixing appropriately the partial charge q and the Bjerrum length l_B , this system mimics *water in an oil background*, as long as the dielectric properties are concerned.

We recall that the electric permittivity is ϵ_0 for vacuum, and $\epsilon < \epsilon_0$ for a medium. The medium effect can be split into a background and a relative term $\epsilon/\epsilon_0 = \epsilon_b \epsilon_r$. The background is constant and uniform, whereas the explicit term is due to dynamic microscopic objects (dimers in this case) which carry a charge dipole moment. The strength of electrostatic interactions in a background is set by the bare Bjerrum length $l_B = e^2 / (4\pi\epsilon_0\epsilon_b k_B T)$. On the other hand, from linear response theory, the bulk value of the relative permittivity is $\epsilon_r = 1 + \frac{\langle \vec{P}^2 \rangle}{3\epsilon_0\epsilon_b k_B T V}$, where tin-foil boundary conditions are assumed.

Two types of beads are present in the simulation, *solp* and *solm*, the solvent positive and negative partial charges, respectively. We fix the bare Bjerrum length $l_B = 42$ (appropriate for oil¹), the repulsion parameter $A = 25$, the harmonic spring constant $k = 5$, the bead density $\rho = 3$, the partial charges $|q| = 0.46$ and the Gaussian smearing length $\sigma = 0.5$. All quantities are given in DPD units, where $k_B T = 1$, $r_c = 1$ and $m = 1$. This fluid has a relative permittivity $\epsilon_r \simeq 40$, as can be checked using the `gen_dipole.f90` utility. This value is compatible with the ratio of water and oil permittivities $\epsilon^{water}/\epsilon^{oil} \simeq 40$.

The FIELD file defining the composition and interactions for a system of volume $V = 64$ is

```
DL_MESO charged harmonic dimers with dpd repulsion

SPECIES 2
solp  1.0  0.46  0
solm  1.0 -0.46  0

MOLECULES 1
DIMER
nummols 96
beads 2
solp  0.0 0.0 0.0
solm  0.1 0.0 0.0
bonds 1
harm 1 2 5.0 0.0
```

(continues on next page)

¹ Notice that the physical length scale is set choosing r_c : if we choose $r_c = 0.646nm$ (appropriate to match water density at room temperature if $N_m = 3$, i.e., one bead represents three water molecules), the Bjerrum length of oil in DPD units is $l_B = 27nm \simeq 42r_c$, hence the value given above for the oil background.

(continued from previous page)

finish

INTERACTIONS 3

solp solp dpd 25.0 1.0 4.5

solm solm dpd 25.0 1.0 4.5

solp solm dpd 25.0 1.0 4.5

CLOSE

As a test, we suggest considering a fluid made of harmonically bonded dimers $(+q, -q)$. Appropriately fixing the partial charge q and the Bjerrum length l_B , this system mimics water in an oil background as far as the dielectric properties are concerned. For more details about this model, please see the page [Test case: a dimer solvent](#).

Run DL_MESO_DPD using the following CONTROL file:

DL_MESO charged harmonic dimers **with** dpd repulsion

volume 64.0

temperature 1.0

cutoff 1.0

timestep 0.01

steps 70000

equilibration steps 20000

traj 20000 10

stats every 100

stack size 100

print every 100

job time 7200.0

close time 10.0

ensemble nvt mdvv

ewald sum 1.0 5 5 5

bjerrum 42.0

smear gauss

smear length 0.5 equal

finish

and the FIELD file:

DL_MESO charged harmonic dimers **with** dpd repulsion

SPECIES 2

solp 1.0 0.46 0

solm 1.0 -0.46 0

MOLECULES 1

DIMER

nummols 96

beads 2

solp 0.0 0.0 0.0

solm 0.1 0.0 0.0

bonds 1

harm 1 2 5.0 0.0

(continues on next page)

(continued from previous page)

```

finish

INTERACTIONS 3
solp solp dpd 25.0 1.0 4.5
solm solm dpd 25.0 1.0 4.5
solp solm dpd 25.0 1.0 4.5

CLOSE

```

Analyzing the HISTORY file with *gen_dipoleaf.exe* choosing *naf=100*, i.e., using this input.txt:

```

100
1

```

this output is printed to the standard output:

```

nchist:          0          96
Number of time steps in autocorrelation profile?
100
switch for FFT computation? (1=yes, 0 or any other integer=no)
1

```

The first line shows the histogram of cluster sizes: in this case, it correctly gives 96 molecules of two beads. Since internally the module checks that each molecule is a connected cluster¹, this line should always give a histogram with the molecule sizes (shown up to the maximum number of beads per molecule).

The *DIPAFDAT* file is (only the first fifteen lines are shown):

```

DL_MESO charged harmonic dimers with dpd repulsion
      5001      100

DIMER
0.000000E+00  1.462829E+01  1.000000E+00
1.000000E-01  1.403865E+01  9.596923E-01
2.000000E-01  1.258271E+01  8.601627E-01
3.000000E-01  1.074853E+01  7.347772E-01
4.000000E-01  8.870221E+00  6.063746E-01
5.000000E-01  7.113151E+00  4.862601E-01
6.000000E-01  5.559272E+00  3.800358E-01
7.000000E-01  4.250179E+00  2.905452E-01
8.000000E-01  3.187677E+00  2.179119E-01
9.000000E-01  2.349055E+00  1.605831E-01

```

and the *DIPAFFFT* file is (only the first fifteen lines are shown):

```

DL_MESO charged harmonic dimers with dpd repulsion
      5001      100

DIMER
0.000000E+00  5.757016E+00  0.000000E+00
6.283185E-01  5.682940E+00 -1.682430E+00
1.256637E+00  4.084017E+00 -2.585684E+00
1.884956E+00  3.921155E+00 -3.116275E+00
2.513274E+00  3.381181E+00 -3.800208E+00
3.141593E+00  2.320091E+00 -3.074025E+00

```

(continues on next page)

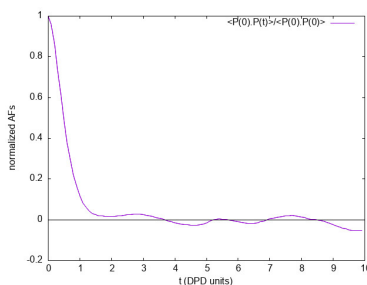
(continued from previous page)

```

3.769911E+00  1.751324E+00 -3.126105E+00
4.398230E+00  1.443640E+00 -2.653086E+00
5.026548E+00  1.105522E+00 -2.423820E+00
5.654867E+00  9.472693E-01 -2.056408E+00

```

Below we show a plot of the normalized AF $\frac{\langle \vec{P}(0)\vec{P}(t) \rangle}{\langle \vec{P}(0)\vec{P}(0) \rangle}$ (obtained using the first and third columns of *DIPAFDAT*)



Source Code

```

1  PROGRAM gen_dipoleaf
2  !*****
3  ! module to compute autocorrelation functions of charge dipole moments in DL_MESO_DPD
4  !
5  ! authors: m. a. seaton and s. chiacchiera, March 2017 (amended August 2017, January
6  ! 2021)
7  !
8  !*****
9
10  USE, INTRINSIC :: iso_c_binding
11  IMPLICIT none
12  INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND (15, 307)
13  INTEGER, PARAMETER :: li = SELECTED_INT_KIND (12)
14  INTEGER, PARAMETER :: ntraj=10
15  INTEGER, PARAMETER :: endversion = 1
16  REAL(KIND=dp), PARAMETER :: pi=3.141592653589793_dp
17
18  CHARACTER(80) :: text
19  CHARACTER(8), ALLOCATABLE :: namspe (:), nammol (:)
20
21  INTEGER, ALLOCATABLE :: ltp (:), ltm (:), mole (:), bndtbl (:,:)
22  INTEGER, ALLOCATABLE :: nbdmol (:), readint (:)
23  INTEGER, ALLOCATABLE :: visit (:), from (:)
24  INTEGER :: nrtout
25  INTEGER :: chain, imol, ioerror, i, numtraj, j, k, l, nmoldef, ibond, nbdmolmx
26  INTEGER :: nspe, nbeads, nusyst, nmbeads, nsyst, numbond, global, species,
27  !molecule
28  INTEGER :: nummol, lfrzn, rnmol, keytrj, srfx, srfy, srfz
29  INTEGER :: naf, nsamp, nl
30  INTEGER :: endver, Dlen, nstep, framesize, lend, leni
31  INTEGER(KIND=li) :: filesize, mypos, currentpos, lend_li, leni_li, framesizeli,
32  !numbeadsli
33
34  REAL(KIND=dp), ALLOCATABLE :: xxx (:), yyy (:), zzz (:), readdata (:)
35  REAL(KIND=dp), ALLOCATABLE :: nmol (:), chg (:), molchg (:)

```

(continues on next page)

(continued from previous page)

```

32  REAL(KIND=dp), ALLOCATABLE :: dipx_box (:), dipy_box (:), dipz_box (:)
33  REAL(KIND=dp), ALLOCATABLE :: dipdata (:,:,), dipdata_box (:,:), corrddata (:)
34  REAL(KIND=dp) :: dimx, dimy, dimz, shrdx, shrdy, shrdz
35  REAL(KIND=dp) :: amass, rcii
36  REAL(KIND=dp) :: time
37  REAL(KIND=dp) :: domega, dt, time0
38  REAL(KIND=dp) :: dx0, dy0, dz0
39
40  INTEGER :: nftpts
41  COMPLEX(C_DOUBLE_COMPLEX), ALLOCATABLE :: fftdata (:)
42
43  LOGICAL :: eof, lfft, swapend, bigend
44
45  !   determine number of bytes for selected double precision and integer kinds
46  !   (the default SELECTED_REAL_KIND (15, 307) should return 8 bytes)
47
48  lend = STORAGE_SIZE (1.0_dp) / 8
49  leni = BIT_SIZE (1) / 8
50  lend_li = INT (lend, KIND=li)
51  leni_li = INT (leni, KIND=li)
52
53  !   check endianness of machine
54
55  bigend = (IACHAR(TRANSFER(1,"a"))==0)
56
57  !   determine if HISTORY file exists, which endianness to use,
58  !   if type of real is correct
59
60  INQUIRE (file = 'HISTORY', EXIST = eof)
61  IF (.NOT. eof) THEN
62      PRINT *, "ERROR: cannot find HISTORY file"
63      STOP
64  END IF
65
66  OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted', status =
↳ 'unknown')
67
68  swapend = .false.
69  READ (ntraj) endver, Dlen
70
71  IF (endver/=endversion) THEN
72      swapend = .true.
73      CLOSE (ntraj)
74      IF (bigend) THEN
75          OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',
↳ status = 'unknown', convert = 'little_endian')
76      ELSE
77          OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',
↳ status = 'unknown', convert = 'big_endian')
78      END IF
79      READ (ntraj) endver, Dlen
80      IF (endver/=endversion) THEN
81          PRINT *, "ERROR: corrupted HISTORY file or created with incorrect version
↳ of DL_MESO"
82          STOP
83      END IF
84  END IF

```

(continues on next page)

(continued from previous page)

```

85
86     IF (Dlen/=lend) THEN
87         PRINT *, "ERROR: incorrect type of real number used in HISTORY file"
88         PRINT *, "      recompile gen_dipoleaf.f90 with reals of ", Dlen, " bytes"
89         STOP
90     END IF
91
92     !      read file size, number of frames and timestep numbers
93
94     READ (ntraj) filesize, numtraj, nstep
95
96     ! Read where the number of beads, molecules and bonds are determined
97     ! Arrays are filled with names of particles and molecules
98
99     READ (ntraj) text
100
101     READ (ntraj) nspe, nmoldef, nusyst, nsyst, numbond, keytrj, srfx, srfy, srfz
102
103     IF (numbond==0) THEN
104         PRINT *, 'ERROR: no molecules in trajectory data!'
105         STOP
106     END IF
107
108     IF (srfx>1 .OR. srfy>1 .OR. srfz>1) THEN
109         WRITE (*,*) "ERROR: Hard walls, electrostatics not implemented in DL_MESO_
↪DPD yet!"
110         STOP
111     END IF
112
113     IF (srfx==1 .OR. srfy==1 .OR. srfz==1) THEN
114         WRITE (*,*) "ERROR: Systems under shear not yet implemented!"
115         STOP
116     END IF
117
118     framesize = (keytrj+1) * 3
119     ALLOCATE (readint (1:nsyst), readdata (1:framesize))
120
121     !      get number of beads to be tracked when reading trajectory file (molecular beads)
122     nmbeads = nsyst - nusyst
123
124     ALLOCATE (namspe (nspe), nammol (nmoldef))
125     ALLOCATE (xxx (1:nmbeads), yyy (1:nmbeads), zzz (1:nmbeads))
126     ALLOCATE (ltp (1:nmbeads), ltm (1:nmbeads), mole (1:nmbeads))
127     ALLOCATE (nmol (1:nmoldef), nbdmol (1:nmoldef))
128     ALLOCATE (chg (nspe))
129     ALLOCATE (bndtbl (numbond, 2))
130     ALLOCATE (visit (nmbeads), from (nmbeads))
131
132     DO i = 1, nspe
133         READ (ntraj) namspe (i), amass, rcii, chg (i), lfrzn
134     END DO
135
136     DO i = 1, nmoldef
137         READ (ntraj) nammol (i)
138     END DO
139
140     !      reading of bead species and molecule types

```

(continues on next page)

(continued from previous page)

```

141 nummol = 0 !counter for number of molecules
142 ibond = 0 !counter for bonds
143
144
145 DO i = 1, nsyst
146   READ (ntraj) global, species, molecule, chain
147   IF (global>nusyst .AND. global<=nsyst) THEN
148     ltp (global-nusyst) = species
149     ltm (global-nusyst) = molecule
150     mole (global-nusyst) = chain
151     nummol = MAX (nummol, chain)
152   END IF
153 END DO
154
155 ! reading of bond tables
156
157 IF (numbond>0) THEN
158   DO i = 1, numbond
159     READ (ntraj) bndtbl (i, 1), bndtbl (i, 2)
160   END DO
161 END IF
162
163 bndtbl = bndtbl - nusyst
164
165 ! reached end of header: find current position in file
166
167 INQUIRE (unit=ntraj, POS=currentpos)
168
169 ! find timestep size from times in first two frames
170
171 framesizeli = INT (framesize, KIND=li)
172 numbeadsli = INT (nsyst, KIND=li)
173
174 READ (ntraj, IOSTAT=ioerror, POS=currentpos) time0
175 mypos = currentpos + (numbeadsli + 1_li) * leni_li + (framesizeli * numbeadsli_
↵+ 7_li) * lend_li
176 READ (ntraj, IOSTAT=ioerror, POS=mypos) time
177
178 dt = time - time0
179
180 ! determine numbers of molecules and beads per molecule type
181 nmol = 0.0_dp
182 nbdmol = 0
183 chain = 0
184 imol = 0 !necessary to avoid out of bounds
185
186 DO i = 1, nmbeads
187   IF (mole (i) /= chain) THEN
188     chain = mole (i)
189     imol = ltm (i)
190     nmol (imol) = nmol (imol) + 1.0_dp
191   END IF
192   IF (imol > 0) nbdmol (imol) = nbdmol (imol) + 1
193 END DO
194
195 DO i = 1, nmoldef
196   rnmol = NINT (nmol (i))

```

(continues on next page)

(continued from previous page)

```

197     IF (rnmol>0) THEN
198         nbdmol (i) = nbdmol (i) / rnmol
199     END IF
200 END DO
201
202 nbdmolmx = MAXVAL (nbdmol (1:nmoldef))
203
204 ! obtain connectivity information (needed only once)
205 CALL connect (nmbeads, numbond, bndtbl, nbdmolmx, visit, from)
206
207 !Checking for charge neutrality of all molecules
208 ALLOCATE (molchg (nummol))
209
210 molchg (:) = 0._dp
211
212 DO i = 1, nmbeads
213     imol = mole (i)
214     molchg (imol) = molchg (imol) + chg (ltp (i))
215 END DO
216
217 DO i = 1, nummol
218     IF (ABS (molchg (i)) > 1.0e-16_dp) THEN
219         WRITE (*,*) "molecule number",i," is not neutral! (The dipole moment is_
↪frame-dependent)"
220         WRITE (*,*) "its charge is=", molchg (i)
221         WRITE (*,*) "its type is=", nammol (i)
222         STOP
223     ENDIF
224 END DO
225
226 CALL check_molecules !checks that beads are labelled as expected
227
228 ! Get the maximum number of time steps for autocorrelation
229 WRITE (*,*) "Number of time steps in autocorrelation profile? "
230 READ (*,*) naf
231 IF (naf<1 .OR. naf>numtraj) naf = numtraj
232
233 ! Get the switch for FFT computation
234 WRITE (*,*) "switch for FFT computation? (1=yes, 0 or any other integer=no)"
235 READ (*,*) n1
236 lfft = (n1 == 1)
237
238 !reading trajectories and computing charge dipole moments
239 ALLOCATE (dipdata (4, nmoldef, numtraj))
240 ALLOCATE (dipx_box (nmoldef), dipy_box (nmoldef), dipz_box (nmoldef))
241
242 eof = .false.
243
244 DO k = 1, numtraj
245
246     mypos = currentpos + INT (k-1, KIND=li) * ((numbeadsli + 1_li) * leni_li +_
↪(framesizeli * numbeadsli + 7_li) * lend_li)
247     READ (ntraj, POS = mypos, IOSTAT=ioerror) time, nbeads, dimx, dimy, dimz,_
↪shrdx, shrdy, shrdz
248
249     IF (ioerror/=0) THEN
250         eof = .true.

```

(continues on next page)

(continued from previous page)

```

251     IF (k==1) THEN
252         PRINT *, 'ERROR: cannot find trajectory data in HISTORY files'
253         STOP
254     END IF
255     EXIT
256 END IF

257
258 READ (ntraj) readint (1:nsyst)
259 DO i = 1, nsyst
260     global = readint (i)
261     READ (ntraj) readdata (1:framesize)
262     IF (global>nusyst .AND. global<=nsyst) THEN
263         xxx (global-nusyst) = readdata (1)
264         yyy (global-nusyst) = readdata (2)
265         zzz (global-nusyst) = readdata (3)
266     END IF
267 END DO

268
269 CALL compute_charge_dipoles (dipx_box, dipy_box, dipz_box)
270
271     ! the dipole components are stored for all the snapshots
272 DO j = 1, nmoldef
273     dipdata (1, j, k) = dipx_box (j)
274     dipdata (2, j, k) = dipy_box (j)
275     dipdata (3, j, k) = dipz_box (j)
276     dipdata (4, j, k) = time
277 END DO

278
279 END DO ! end of loop over trajectories

280
281 IF (k <= numtraj) THEN
282     WRITE (*,*) "ERROR: problem with the number of snapshots!"
283     STOP
284 END IF

285
286     ! Close the trajectory file
287 CLOSE (ntraj)

288
289 nsamp = numtraj - naf + 1
290
291 ALLOCATE (corrdata (naf))
292
293     ! define FFT size if needed
294 IF (lfft) THEN
295     nftpts = naf ! modify here to change the size of the DFT
296     omega = 2.0_dp * pi / (dt * REAL(nftpts, KIND=dp))
297     ALLOCATE (fftdata (nftpts))
298 END IF

299
300     ! Open output file, compute the autocorrelation and write it there
301 nrtout = ntraj + 1
302
303 IF (numtraj>0) THEN
304
305     OPEN (nrtout, file='DIPAFDAT', status='replace')
306     WRITE (nrtout, '(a80)') text
307     WRITE (nrtout, '(2i10)') numtraj,naf

```

(continues on next page)

(continued from previous page)

```

308     WRITE (nrtout, '(/)')
309
310     ! Open the FT otuput file if needed
311     IF (lfft) THEN
312         OPEN (nrtout+1, file='DIPAFFFT', status='replace')
313         WRITE (nrtout+1, '(a80)') text
314         WRITE (nrtout+1, '(2i10)') numtraj,nftpts
315         WRITE (nrtout+1, '(/)')
316     END IF
317
318     DO j = 1, nmoldef
319         corrddata = 0.0_dp
320         WRITE (nrtout, '(a8)') nammol (j)
321         IF (lfft) WRITE (nrtout+1, '(a8)') nammol (j)
322         DO i = 1, nsamp
323             dx0 = dipdata (1, j, i)
324             dy0 = dipdata (2, j, i)
325             dz0 = dipdata (3, j, i)
326             DO l = 1, naf
327                 corrddata (l) = corrddata (l) + dipdata (1, j, i+l-1) * dx0 + dipdata_
328 ↪ (2, j, i+l-1) * dy0 &
329                 + dipdata (3, j, i+l-1) * dz0
330             END DO
331         END DO
332         corrddata = corrddata / REAL (nsamp, KIND=dp)
333         DO i = 1, naf
334             WRITE (nrtout, '(1p,3e14.6)') REAL (i-1, KIND=dp)*dt, corrddata (i),_
335 ↪ corrddata (i)/corrddata(1)
336         END DO
337         WRITE (nrtout, '(/)')
338         IF (lfft) THEN
339             fftdata (:) = corrddata (:)/ corrddata (1) ! adapt here if nftpts_
340 ↪ differs from naf
341             CALL fft (fftdata)
342             DO i = 1, nftpts
343                 WRITE (nrtout+1, '(1p,3e14.6)') REAL (i-1, KIND=dp)*domega, fftdata_
344 ↪ (i)
345             END DO
346             WRITE (nrtout+1, '(/)')
347         END IF
348     END DO
349
350     ! Calculation for the total system dipole
351     ALLOCATE (dipdata_box (4, numtraj))
352     dipdata_box = SUM (dipdata, 2) !sum of dipoles over all molecular species
353     corrddata = 0.0_dp
354     WRITE (nrtout, '("all species")')
355     IF (lfft) WRITE (nrtout+1, '("all species")')
356     DO i = 1, nsamp
357         dx0 = dipdata_box (1, i)
358         dy0 = dipdata_box (2, i)
359         dz0 = dipdata_box (3, i)
360         DO l = 1, naf
361             corrddata (l) = corrddata (l) + dipdata_box (1, i+l-1) * dx0 + dipdata_
362 ↪ box (2, i+l-1) * dy0 &
363             + dipdata_box (3, i+l-1) * dz0
364         END DO
365     END DO

```

(continues on next page)

(continued from previous page)

```

360      corrddata = corrddata / REAL (nsamp, KIND=dp)
361      DO i = 1, naf
362          WRITE (nrtout, '(1p,3e14.6)') REAL (i-1, KIND=dp)*dt, corrddata (i),
↪corrddata (i)/corrddata(1)
363      END DO
364      WRITE (nrtout, '(/)')
365      IF (lfft) THEN
366          fftdata (:) = corrddata (:) / corrddata (1) ! adapt here if nftpts
↪differs from naf
367          call fft (fftdata)
368          DO i = 1, nftpts
369              WRITE (nrtout+1, '(1p,3e14.6)') REAL (i-1, KIND=dp)*domega,
↪fftdata (i)
370          END DO
371          WRITE (nrtout+1, '(/)')
372      END IF
373      DEALLOCATE (dipdata_box)
374  END IF
375
376      ! Close the output files
377      CLOSE (nrtout)
378      IF (lfft) CLOSE (nrtout+1)
379
380      DEALLOCATE (readint, readdata)
381      DEALLOCATE (namspe, nammol)
382      DEALLOCATE (xxx, yyy, zzz)
383      DEALLOCATE (ltp, ltm, mole)
384      DEALLOCATE (nmol, nbdmol)
385      DEALLOCATE (chg, molchg)
386      DEALLOCATE (dipx_box, dipy_box, dipz_box)
387      DEALLOCATE (bndtbl)
388      DEALLOCATE (visit, from)
389      DEALLOCATE (dipdata, corrddata)
390      IF (lfft) DEALLOCATE (fftdata)
391
392  CONTAINS
393
394      SUBROUTINE check_molecules
395      !*****
396      ! subroutine to check molecular content and labelling
397      !
398      ! authors: s. chiacchiera, February 2017
399      !
↪*****
↪
400      IMPLICIT NONE
401      INTEGER i, j, k, tm, tp, imol, im, ibd
402      INTEGER mxmolsize
403      INTEGER, ALLOCATABLE :: molbeads (:,:)
404
405      mxmolsize = 0
406      DO i = 1, nmoldef
407          mxmolsize = MAX (nbdmol(i), mxmolsize)
408      END DO
409      ALLOCATE (molbeads (nmoldef, mxmolsize))
410      molbeads (:,:) = 0
411

```

(continues on next page)

(continued from previous page)

```

412     imol = 0
413     ibd = 0
414     DO i = 1, nmoldef
415         DO j = 1, NINT (nmol(i))
416             imol = imol + 1
417             DO k = 1, nbdmol(i)
418                 ibd = ibd + 1
419                 tm = ltm (ibd)
420                 tp = ltp (ibd)
421                 im = mole (ibd)
422                 IF (j==1) THEN
423                     molbeads (i, k) = tp
424                 ELSE
425                     IF (molbeads (i, k) /= tp) THEN
426                         WRITE (*,*) "ERROR: Problem with molecular content!"
427                         STOP
428                     ENDIF
429                 ENDIF
430             IF (tm/=i.OR.im/=imol) THEN
431                 WRITE (*,*) "ERROR: Problem with molecules labels!"
432                 STOP
433             ENDIF
434         END DO
435     END DO
436 END DO
437 IF (imol/=nummol) THEN
438     WRITE (*,*) "ERROR: imol and nummol differ!"
439     STOP
440 ENDIF
441 DEALLOCATE (molbeads)
442 RETURN
443 END SUBROUTINE check_molecules
444
445 SUBROUTINE compute_charge_dipoles (dipx_box, dipy_box, dipz_box)
446 !*****
447 ! subroutine to compute charge dipole moments
448 !
449 ! authors: m. a. seaton and s. chiacchiera, February 2017
450 !
451 ! input: xxx, yyy, zzz (at a given time step) and chg
452 ! input: visit and from (obtained using connect)
453 ! output: the x,y,z components of the total dipole, for each molecule type (at a given
454 !         time step)
455 !
456 ! (NB: this is a slightly modified version, with fewer outputs)
457 !
458 !*****
459
460 IMPLICIT NONE
461 INTEGER i, j, k, tm, tp, imol, ibd, count, ipr
462 REAL(KIND=dp), DIMENSION(nmoldef) :: dipx_box, dipy_box, dipz_box
463 REAL(KIND=dp) :: x, y, z, dx, dy, dz, xpre, ypre, zpre
464 REAL(KIND=dp) :: dipx, dipy, dipz
465 REAL(KIND=dp), DIMENSION(nmbeads) :: xabs, yabs, zabs
466
467 dipx_box (:) = 0._dp
468 dipy_box (:) = 0._dp

```

(continues on next page)

(continued from previous page)

```

467     dipz_box (:) = 0._dp
468
469     imol = 0
470     count = 0
471     ! xabs = 0._dp ! just to keep it clean
472     ! yabs = 0._dp
473     ! zabs = 0._dp
474
475     DO i = 1, nmoldef
476         tm = i
477         DO j = 1, NINT (nmol(i))
478             imol = imol + 1
479
480             dipx = 0._dp ! dipole of a SINGLE molecule
481             dipy = 0._dp
482             dipz = 0._dp
483
484             DO k = 1, nbdmol(i)
485                 count = count + 1
486                 ibd = visit (count)
487                 ipr = from (count)
488
489                 IF (ipr /= 0) THEN
490                     xpre = xabs (ipr)
491                     ypre = yabs (ipr)
492                     zpre = zabs (ipr)
493                 ELSE
494                     IF (k == 1) THEN
495                         xpre = 0._dp
496                         ypre = 0._dp
497                         zpre = 0._dp
498                     ELSE
499                         WRITE (*,*) "Unconnected molecule!"
500                         STOP
501                     ENDIF
502                 ENDIF
503
504                 tp = ltp (ibd)
505
506                 dx = xxx (ibd) - xpre
507                 dy = yyy (ibd) - ypre
508                 dz = zzz (ibd) - zpre
509
510                 dx = dx - dimx * ANINT (dx/dimx)
511                 dy = dy - dimy * ANINT (dy/dimy)
512                 dz = dz - dimz * ANINT (dz/dimz)
513
514                 x = xpre + dx
515                 y = ypre + dy
516                 z = zpre + dz
517
518
519                 dipx = dipx + x * chg (tp)
520                 dipy = dipy + y * chg (tp)
521                 dipz = dipz + z * chg (tp)
522
523                 xabs (ibd) = x

```

(continues on next page)

(continued from previous page)

```

524         yabs (ibd) = y
525         zabs (ibd) = z
526
527         END DO
528
529         dipx_box (tm) = dipx_box (tm) + dipx
530         dipy_box (tm) = dipy_box (tm) + dipy
531         dipz_box (tm) = dipz_box (tm) + dipz
532
533         END DO
534     END DO
535
536     IF (imol/=nummol) THEN
537         WRITE (*,*) "ERROR: imol and nummol differ!"
538         STOP
539     ENDIF
540
541     RETURN
542 END SUBROUTINE compute_charge_dipoles
543
544 SUBROUTINE fft (x)
545 !
546 ! *****
547 ! Subroutine to call FFTW (v3) one-dimensional complex DFT.
548 ! Notice that the input array is overwritten with the its Discrete Fourier Transform.
549 !
550 ! author: s. chiacchiera, August 2017
551 ! amended: m. a. seaton, January 2021
552 !
553 ! *****
554
555     USE, INTRINSIC :: iso_c_binding
556     IMPLICIT none
557     INCLUDE 'fftw3.f03'
558     COMPLEX(C_DOUBLE_COMPLEX), INTENT(INOUT) :: x (:)
559     INTEGER :: n
560     TYPE(C_PTR) :: plan
561
562     n = SIZE (x)
563
564     plan = fftw_plan_dft_1d (n, x, x, FFTW_FORWARD, FFTW_ESTIMATE)
565     CALL fftw_execute_dft (plan, x, x)
566     CALL fftw_destroy_plan (plan)
567
568     RETURN
569 END SUBROUTINE fft
570
571 End PROGRAM gen_dipoleaf
572
573 SUBROUTINE connect (nbeads, nbonds, bndtbl, mxmolsize, visit, from)
574 ! *****
575 ! Analyzes all the bonds (bndtbl) to obtain a schedule (visit, from)
576 ! to visit the beads so that each cluster is visited along a connected
577 ! path. "visit" gives the order to include beads, "from" gives the bead
578 ! to attach them to.

```

(continues on next page)

(continued from previous page)

```

577 ! (Note: vocabulary from infection propagation used to move along
578 ! clusters)
579 !
580 ! author: s. chiacchiera, February 2017
581 ! amended: m. a. seaton, January 2021
582 ! *****
583 IMPLICIT none
584 INTEGER, INTENT (IN) :: bndtbl (nbonds,2)
585 INTEGER, INTENT (IN) :: nbeads, nbonds
586 INTEGER, INTENT (IN) :: mxmolsize
587 INTEGER :: ic, i, k, nn, nclu, nper, lab, ref, count
588 INTEGER, ALLOCATABLE :: firstnn (:), lastnn (:), deg (:)
589 INTEGER, ALLOCATABLE :: labnn (:)
590 INTEGER, ALLOCATABLE :: state (:)
591 INTEGER, ALLOCATABLE :: perlab (:), perref (:)
592 INTEGER, ALLOCATABLE :: nchist (:)
593 INTEGER, INTENT (OUT) :: visit (nbeads), from (nbeads)
594
595 ALLOCATE (firstnn (nbeads), lastnn (nbeads), deg (nbeads), labnn (2*nbonds))
596 ALLOCATE (state (nbeads))
597 ALLOCATE (perlab (nbeads), perref (nbeads))
598 ALLOCATE (nchist (mxmolsize))
599 !-----
600 CALL organize (nbeads, nbonds, labnn, firstnn, lastnn, deg)
601 !-----
602 state (:) = 0
603 nchist (:) = 0
604 visit (:) = 0
605 from (:) = 0
606 count = 0
607 !-----
608 ic = 0
609 !-----
610 DO WHILE (ic < nbeads) ! ic = label of bead used to "grow" a cluster
611   ic = ic + 1
612   IF ( state (ic) /= 0) THEN
613     WRITE (*,*) "ERROR: labels are not as expected!"
614     STOP
615   END IF
616   nclu = 1
617   count = count + 1
618   visit (ic) = ic
619   IF (deg (ic) == 0) THEN
620     state (ic) = -1
621     IF (nclu <= mxmolsize) nchist (nclu) = nchist (nclu) +1
622     CYCLE
623   END IF
624   state (ic) = 1 ! ic is "infected"
625
626   ! nearest neighbours of ic are marked as "goint to be infected" -> a.k.a.
627   ↳perimeter
628   nper = 0
629   perlab (:) = 0
630   perref (:) = 0
631   DO k = firstnn (ic), lastnn (ic)
632     nn = labnn (k)
633     IF ( state (nn) /= 0) THEN

```

(continues on next page)

(continued from previous page)

```

633         WRITE (*,*) "ERROR: labels are not as expected!"
634         STOP
635     END IF
636     nper = nper + 1
637     perlab (nper) = nn !new bead in perimeter
638     perref (nper) = ic !its reference bead (origin of the link)
639     state (nn) = 2
640 END DO
641 state (ic) = 3 ! ic is "dead"
642
643 DO WHILE (nper > 0)
644     i = 1 ! pick a bead of "perimeter" to be analyzed
645     lab = perlab (i)
646     ref = perref (i)
647     perlab (i) = perlab (nper)
648     perref (i) = perref (nper)
649     nper = nper - 1
650     IF (state (lab) == 3) THEN
651         CYCLE
652     END IF
653     state (lab) = 1 ! "lab" is added to the cluster
654     nclu = nclu + 1
655     count = count + 1
656     visit (count) = lab
657     from (count) = ref
658
659     DO k = firstnn (lab), lastnn (lab) ! check nn of newly added
660         nn = labnn (k)
661         IF (state (nn) == 2) .OR. (state (nn) == 3) CYCLE
662         nper = nper + 1
663         perlab (nper) = nn !new bead in perimeter
664         perref (nper) = lab !its reference bead (origin of the link)
665         state (nn) = 2
666     END DO
667     state (lab) = 3
668
669 END DO
670 nchist (nclu) = nchist (nclu) +1
671 ic = ic + nclu - 1 ! prepare ic for the next cluster
672 END DO
673 WRITE (*,*) "nchist: ", nchist
674 !-----
675 DEALLOCATE (firstnn, lastnn, deg, labnn)
676 DEALLOCATE (state)
677 DEALLOCATE (perlab, perref)
678 DEALLOCATE (nchist)
679 RETURN
680 !-----
681 CONTAINS
682 !-----
683 SUBROUTINE organize (N, NL, labnn, firstnn, lastnn, deg)
684 !*****
685 ! Analyzes the bonds (bndtbl) to obtain the degree (=number of bonds)
686 ! of each bead, and the nearest neighbours list.
687 ! N in the number of beads (vertices) and NL of bonds (links).
688 !
689 ! author: s. chiacchiera, February 2017

```

(continues on next page)

(continued from previous page)

```

690      !*****
691      IMPLICIT none
692      INTEGER, INTENT(IN) :: N, NL
693      INTEGER :: i,l,count_lab, i1,i2
694      INTEGER, DIMENSION (N), INTENT(OUT) :: deg
695      INTEGER, DIMENSION (N), INTENT(OUT) :: firstnn, lastnn
696      INTEGER, DIMENSION (2*NL), intent(OUT) :: labnn
697
698      deg(:)=0
699      firstnn(:)=0
700      lastnn(:)=0
701      labnn(:)=0
702
703      count_lab=0
704
705      DO i=1,N
706          DO l=1,NL
707              IF (bndtbl(l,1).EQ.i) THEN
708                  deg(i)=deg(i)+1
709                  count_lab=count_lab+1
710                  labnn(count_lab)=bndtbl(l,2)
711              ENDIF
712              IF (bndtbl(l,2).EQ.i) THEN
713                  deg(i)=deg(i)+1
714                  count_lab=count_lab+1
715                  labnn(count_lab)=bndtbl(l,1)
716              ENDIF
717          END DO
718      END DO
719
720      i1=1
721      i2=0
722      DO i=1,N
723          IF (deg(i)==0) CYCLE
724          firstnn(i)=i1
725          i2=i1+deg(i)-1
726          lastnn(i)=i2
727          i1=i2+1
728      END DO
729
730      RETURN
731
732      END SUBROUTINE organize
733      !-----
734      END SUBROUTINE connect
735

```

Autocorrelation functions of individual charge dipole moments in DL_MESO_DPD

Software Technical Information

Language Fortran 2003

Licence BSD

Documentation Tool RST and LaTeX-generated .pdf file

Application Documentation Click to download the manual with more details

Relevant Training Material See the Testing section

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*

Purpose of Module

This module, `gen_moldipaf.f90`, is a post-processing utility for DL_MESO_DPD, the Dissipative Particle Dynamics (DPD) code from the [DL_MESO](#) package.

It processes the trajectory (HISTORY) files to obtain the charge dipole moments of all the (neutral) molecules in the system, and subsequently computes the dipole autocorrelation functions (DAFs) of individual molecules for each molecule type. It produces a file *MDIPAFDAT* containing both the un-normalized and normalized DAFs, and, optionally, a file *MDIPAFFFT* containing the Fourier transform (FT) of the latter. It is analogous to `gen_dipoleaf.f90`, but deals with individual (for a single molecule) rather than macroscopic (for the simulated volume) charge dipole moments.

The module can be applied to systems including molecules with a generic charge structure, as long as each molecule is neutral (otherwise the charge dipole moment would be frame-dependent)¹.

CAVEAT: this module only analyzes molecular trajectories. If a charged molecule is present, an error message will be given, while unbonded charges will not be detected and erroneous results may be obtained. Therefore please keep in mind **to not apply** this module to systems with unbonded charges.

The charge dipole moment of a neutral molecule is $\vec{p}_{mol} = \sum_{i \in mol} q_i \vec{r}_i$ where \vec{r}_i are the bead positions and q_i their charges. The total charge dipole moment of the simulated volume V is $\vec{P} = \sum_{mol \in V} \vec{p}_{mol}$. If more than one molecular species are present, one can split \vec{P} into the different species contributions: $\vec{P} = \sum_{j=1}^{N_{moldef}} \vec{P}^{(j)} = \sum_{j=1}^{N_{moldef}} \sum_{k=1}^{N_{mol}^{(j)}} \vec{p}_k^{(j)}$, where N_{moldef} is the number of molecule types (definitions) and $N_{mol}^{(j)}$ the number of molecules of type j .

Given a scalar quantity A , its non-normalized autocorrelation function (AF) is $C_{AA}(t) = \langle A(0)A(t) \rangle$, where $\langle \dots \rangle$ indicates an average over trajectories. The normalized one is $c_{AA}(t) = \frac{\langle A(0)A(t) \rangle}{\langle A(0)A(0) \rangle} = \frac{C_{AA}(t)}{C_{AA}(0)}$ ².

Here for the j -th molecular species we replace A with $\vec{p}^{(j)}$, and the product with a scalar product. In this case the average over trajectories translates into two sums, one over different time origins and one over molecules of species j .

The output file *MDIPAFDAT* contains the DAFs for each molecular species and, at the end of the file, the DAF obtained by averaging over *all* the particles. The output file *MDIPAFFFT* contains the FT of these functions in the same order.

More in detail, the header of the file *MDIPAFDAT* contains the simulation title and a line with the number of snapshots in HISTORY and of those used for the AFs (*naf*). Then a block follows for each molecule type, started by the

¹ Disambiguation on the concept of molecule. In DL_MESO a *defined molecule* is a set of beads, which can be bonded or not. For the purpose of this module it is *required* that each molecule is a connected cluster (via stretching bonds). In fact, this - together with the reasonable assumption that each stretching bond cannot be stretched to more than half the system linear size - allows us to univocally define the charge dipole moment of each molecule.

² M. P. Allen and D. J. Tildesley, "Computer simulation of liquids", Oxford University Press, Oxford (1987).

{molecule name}, then three columns of data, $t, C_{\bar{p}\bar{p}}, c_{\bar{p}\bar{p}}$. It is intended that in any block only the molecules for a given species are summed over. The last block is called *{all species}* and refers to an average over all the molecules.

The header of the file *MDIPAFFFT* is as for *MDIPAFDAT* (notice that the number of points for the FT is also set equal to *naf*). Then a block follows for each molecule type, started by the molecule name, then three columns of data, $\omega, \Re[\hat{c}_{\bar{p}\bar{p}}(\omega)], \Im[\hat{c}_{\bar{p}\bar{p}}(\omega)]$, where \hat{c} is the discrete FT of $c(t)$.

Possible uses of the output file are: to analyze it to obtain the decay time of autocorrelations, which can be used to define an efficient sampling time for the simulation; to compare it with the analogous macroscopic value obtained for all the molecules (of a given type) in the system (see [Autocorrelation functions of charge dipole moments in DL_MESO_DPD](#)).

Background Information

The base code for this module is DL_MESO_DPD, the Dissipative Particle Dynamics code from the mesoscopic simulation package [DL_MESO](#), developed by M. Seaton at Daresbury Laboratory. This open source code is available from STFC under both academic (free) and commercial (paid) licenses. The module is to be used with DL_MESO in its most recently released version, version 2.7 (dating December 2018).

A variant of this module for use with a previous version of DL_MESO, version 2.6 (dating November 2015), can be found in the `old-v2.6` directory³.

The present module also requires the library [FFTW](#) (version 3.x) to be installed.

Testing

The present module `gen_moldipaf.f90` is compiled with the available Fortran 2003 compiler, e.g.:

```
gfortran -o gen_moldipaf.exe gen_moldipaf.f90 -I/usr/local/include -L/usr/local/lib -lfftw3 -lm
```

where `-I` indicates the location of the FFTW include file `fftw3.f03` and `-L` points to the directory containing the FFTW library files. The above command gives the most likely locations for these files, although these may need to be adjusted if FFTW has been installed somewhere else on your machine.

The executable must be in the same directory of the HISTORY file to be analyzed. The user is asked to provide the maximum number of snapshots to be used for the AFs (*naf*) and a switch for the Fourier transform: 1 for yes, 0 (or any other integer) for *no*.

To input these parameters one can either enter them from the keyboard or write them into a text file (say, *input.txt*), one per line in the right order, and run the program in this way:

```
gen_dipoleaf.exe < input.txt
```

Test: water in oil

As a test, we suggest considering a fluid made of harmonically bonded dimers $(+q, -q)$. Appropriately fixing the partial charges q and the Bjerrum length l_B , this system mimics water in an oil background as far as the dielectric properties are concerned. For more details about this model, please see the page [Test case: a dimer solvent](#).

Run DL_MESO_DPD using the following CONTROL file:

```
DL_MESO charged harmonic dimers with dpd repulsion
volume 64.0
```

(continues on next page)

³ A small change to specifying charge smearing schemes and lengths in CONTROL files has been made since version 2.6: the `old-v2.6` folder includes the CONTROL file for the test shown here that will work with this version of DL_MESO.

(continued from previous page)

```

temperature 1.0
cutoff 1.0

timestep 0.01
steps 70000
equilibration steps 20000
traj 20000 10
stats every 100
stack size 100
print every 100
job time 7200.0
close time 10.0

ensemble nvt mdvv

ewald sum 1.0 5 5 5
bjerrum 42.0
smear gauss
smear length 0.5 equal

finish

```

and the FIELD file:

```

DL_MESO charged harmonic dimers with dpd repulsion

SPECIES 2
solp 1.0 0.46 0
solm 1.0 -0.46 0

MOLECULES 1
DIMER
nummols 96
beads 2
solp 0.0 0.0 0.0
solm 0.1 0.0 0.0
bonds 1
harm 1 2 5.0 0.0

finish

INTERACTIONS 3
solp solp dpd 25.0 1.0 4.5
solm solm dpd 25.0 1.0 4.5
solp solm dpd 25.0 1.0 4.5

CLOSE

```

Analyzing the HISTORY file with *gen_moldipaf.exe* choosing *naf=100*, i.e., using this input.txt:

```

100
1

```

this output is printed to the standard output:

```

nchist:          0          96
Number of time steps in autocorrelation profile?

```

(continues on next page)

(continued from previous page)

```

100
switch for FFT computation? (1=yes, 0 or any other integer=no)
1

```

The first line shows a histogram of cluster sizes: in this case, it correctly gives 96 molecules of two beads. Since internally the module checks that each molecule is a connected cluster¹, this line should always give a histogram with the molecule sizes (shown up to the maximum number of beads per molecule).

The first fifteen lines of the *MDIPAFDAT* file are as follows:

```

DL_MESO charged harmonic dimers with dpd repulsion
      5001      100

DIMER
  0.000000E+00  1.415414E-01  1.000000E+00
  1.000000E-01  1.355954E-01  9.579910E-01
  2.000000E-01  1.212470E-01  8.566184E-01
  3.000000E-01  1.038903E-01  7.339924E-01
  4.000000E-01  8.670281E-02  6.125615E-01
  5.000000E-01  7.071286E-02  4.995913E-01
  6.000000E-01  5.627970E-02  3.976200E-01
  7.000000E-01  4.371192E-02  3.088278E-01
  8.000000E-01  3.315358E-02  2.342323E-01
  9.000000E-01  2.453714E-02  1.733566E-01

```

and the *MDIPAFFFT* file starts as follows:

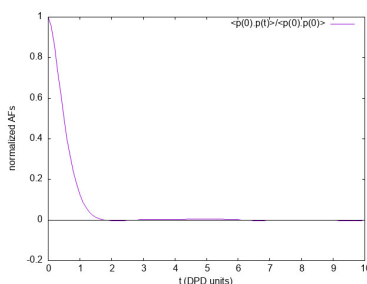
```

DL_MESO charged harmonic dimers with dpd repulsion
      5001      100

DIMER
  0.000000E+00  6.144240E+00  0.000000E+00
  6.283185E-01  5.756980E+00 -1.369256E+00
  1.256637E+00  5.201895E+00 -2.381423E+00
  1.884956E+00  4.104006E+00 -3.103102E+00
  2.513274E+00  3.144505E+00 -3.321907E+00
  3.141593E+00  2.259571E+00 -3.137872E+00
  3.769911E+00  1.681025E+00 -2.860935E+00
  4.398230E+00  1.291102E+00 -2.543543E+00
  5.026548E+00  1.038623E+00 -2.209089E+00
  5.654867E+00  8.770894E-01 -1.946351E+00

```

Below we show a plot of the normalized AF $\frac{\langle \vec{p}(0) \vec{p}(t) \rangle}{\langle \vec{p}(0) \vec{p}(0) \rangle}$ (obtained using the first and third columns of *MDIPAFDAT*).



Source Code

```

1 PROGRAM gen_moldipaf
2 !*****
3 ! module to compute autocorrelation functions of individual charge dipole moments in
4 ! DL_MESO_DPD
5 !
6 ! authors: m. a. seaton and s. chiacchiera, March 2017 (amended August 2017, January
7 !         2021)
8 !
9 !*****
10
11 USE, INTRINSIC :: iso_c_binding
12 IMPLICIT none
13 INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND (15, 307)
14 INTEGER, PARAMETER :: li = SELECTED_INT_KIND (12)
15 INTEGER, PARAMETER :: ntraj=10
16 INTEGER, PARAMETER :: endversion = 1
17 REAL(KIND=dp), PARAMETER :: pi=3.141592653589793_dp
18
19 CHARACTER(80) :: text
20 CHARACTER(8), ALLOCATABLE :: namspe (:), nammol (:)
21
22 INTEGER, ALLOCATABLE :: ltp (:), ltm (:), mole (:), bndtbl (:,:)
23 INTEGER, ALLOCATABLE :: nbdmol (:), readint (:)
24 INTEGER, ALLOCATABLE :: visit (:), from (:)
25 INTEGER :: nrtout
26 INTEGER :: chain, imol, ioerror, i, numtraj, j, k, l, nmoldef, nbdmolmx
27 INTEGER :: nspe, nbeads, nusyst, nmbeads, nsyst, numbond, global, species,
28
29 !molecule
30 INTEGER :: nummol, lfrzn, rnmol, keytrj, srfx, srfy, srfz
31 INTEGER :: nl
32 INTEGER :: naf, nsamp
33 INTEGER :: endver, Dlen, nstep, framesize, lend, leni
34 INTEGER(KIND=li) :: filesize, mypos, currentpos, lend_li, leni_li, framesizeli,
35
36 !numbeadsli
37
38 REAL(KIND=dp), ALLOCATABLE :: xxx (:), yyy (:), zzz (:), readdata (:)
39 REAL(KIND=dp), ALLOCATABLE :: nmol (:), chg (:), molchg (:)
40 REAL(KIND=dp), ALLOCATABLE :: dipx_box (:), dipy_box (:), dipz_box (:)
41 REAL(KIND=dp), ALLOCATABLE :: dipx (:), dipy (:), dipz (:)
42 REAL(KIND=dp), ALLOCATABLE :: mdipdata (:,:,), corrddata (:)
43 REAL(KIND=dp) :: dimx, dimy, dimz, shrdx, shrdy, shrdz
44 REAL(KIND=dp) :: amass, rcii
45 REAL(KIND=dp) :: time
46 REAL(KIND=dp) :: dt, time0, domega
47 REAL(KIND=dp) :: dx0, dy0, dz0
48
49 INTEGER :: nftpts
50 COMPLEX(C_DOUBLE_COMPLEX), ALLOCATABLE :: fftdata (:)
51
52 LOGICAL :: eof, lfft, swapend, bigend
53
54 ! determine number of bytes for selected double precision and integer kinds
55 ! (the default SELECTED_REAL_KIND (15, 307) should return 8 bytes)
56
57 lend = STORAGE_SIZE (1.0_dp) / 8

```

(continues on next page)

(continued from previous page)

```

52     leni = BIT_SIZE (1) / 8
53     lend_li = INT (lend, KIND=li)
54     leni_li = INT (leni, KIND=li)
55
56     !      check endianness of machine
57
58     bigend = (IACHAR(TRANSFER(1,"a"))==0)
59
60     !      determine if HISTORY file exists, which endianness to use,
61     !      if type of real is correct
62
63     INQUIRE (file = 'HISTORY', EXIST = eof)
64     IF (.NOT. eof) THEN
65         PRINT *, "ERROR: cannot find HISTORY file"
66         STOP
67     END IF
68
69     OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted', status_
↪= 'unknown')
70
71     swapend = .false.
72     READ (ntraj) endver, Dlen
73
74     IF (endver/=endversion) THEN
75         swapend = .true.
76         CLOSE (ntraj)
77         IF (bigend) THEN
78             OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',_
↪status = 'unknown', convert = 'little_endian')
79         ELSE
80             OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted',_
↪status = 'unknown', convert = 'big_endian')
81         END IF
82         READ (ntraj) endver, Dlen
83         IF (endver/=endversion) THEN
84             PRINT *, "ERROR: corrupted HISTORY file or created with incorrect version_
↪of DL_MESO"
85             STOP
86         END IF
87     END IF
88
89     IF (Dlen/=lend) THEN
90         PRINT *, "ERROR: incorrect type of real number used in HISTORY file"
91         PRINT *, "      recompile gen_dipole.f90 with reals of ", Dlen, " bytes"
92         STOP
93     END IF
94
95     !      read file size, number of frames and timestep numbers
96
97     READ (ntraj) filesize, numtraj, nstep
98
99     ! Read where the number of beads, molecules and bonds are determined
100    ! Arrays are filled with names of particles and molecules
101
102    READ (ntraj) text
103
104    READ (ntraj) nspe, nmoldef, nusyst, nsyst, numbond, keytrj, srfx, srfy, srfz

```

(continues on next page)

(continued from previous page)

```

105
106     IF (numbond==0) THEN
107         PRINT *, 'ERROR: no molecules in trajectory data!'
108         STOP
109     END IF
110
111     IF (srfx > 1 .OR. srfy > 1 .OR. srfz > 1) THEN
112         WRITE (*,*) "ERROR: Hard walls, electrostatics not implemented in DL_MESO_
↪DPD yet!"
113         STOP
114     END IF
115
116     IF (srfx == 1 .OR. srfy == 1 .OR. srfz == 1) THEN
117         WRITE (*,*) "ERROR: System under shear, not implemented yet!"
118         STOP
119     END IF
120
121     framesize = (keytrj+1) * 3
122     ALLOCATE (readint (1:nsyst), readdata (1:framesize))
123
124     ! get number of beads to be tracked when reading trajectory file (molecular beads)
125     nmbeads = nsyst - nusyst
126
127     ALLOCATE (namspe (nspe), nammol (nmoldef))
128     ALLOCATE (xxx (1:nmbeads), yyy (1:nmbeads), zzz (1:nmbeads))
129     ALLOCATE (ltp (1:nmbeads), ltm (1:nmbeads), mole (1:nmbeads))
130     ALLOCATE (nmol (1:nmoldef), nbdmol (1:nmoldef))
131     ALLOCATE (chg (nspe))
132     ALLOCATE (bndtbl (numbond, 2))
133     ALLOCATE (visit (nmbeads), from (nmbeads))
134
135     DO i = 1, nspe
136         READ (ntraj) namspe (i), amass, rcii, chg (i), lfrzn
137     END DO
138
139     DO i = 1, nmoldef
140         READ (ntraj) nammol (i)
141     END DO
142
143     ! reading of bead species and molecule types
144
145     nummol = 0 ! counter for number of molecules
146     ! ibond = 0 !counter for bonds
147
148     DO i = 1, nsyst
149         READ (ntraj) global, species, molecule, chain
150         IF (global>nusyst .AND. global<=nsyst) THEN
151             ltp (global-nusyst) = species
152             ltm (global-nusyst) = molecule
153             mole (global-nusyst) = chain
154             nummol = MAX (nummol, chain)
155         END IF
156     END DO
157
158     ! reading of bond tables
159
160     IF (numbond>0) THEN

```

(continues on next page)

(continued from previous page)

```

161      DO i = 1, numbond
162          READ (ntraj) bndtbl (i, 1), bndtbl (i, 2)
163      END DO
164  END IF
165
166  bndtbl = bndtbl - nusyst
167
168  !   reached end of header: find current position in file
169
170  INQUIRE (unit=ntraj, POS=currentpos)
171
172  ! find timestep size from times in first two frames
173
174  framesizeli = INT (framesize, KIND=li)
175  numbeadsli = INT (nsyst, KIND=li)
176
177  READ (ntraj, IOSTAT=ioerror, POS=currentpos) time0
178  mypos = currentpos + (numbeadsli + 1_li) * leni_li + (framesizeli * numbeadsli_
179  ↪ + 7_li) * lend_li
180  READ (ntraj, IOSTAT=ioerror, POS=mypos) time
181
182  dt = time - time0
183
184  ! determine numbers of molecules and beads per molecule type
185
186  nmol = 0.0_dp
187  nbdmol = 0
188  chain = 0
189  imol = 0 ! necessary to avoid out of bounds
190
191  DO i = 1, nmbeads
192      IF (mole (i) /= chain) THEN
193          chain = mole (i)
194          imol = ltm (i)
195          nmol (imol) = nmol (imol) + 1.0_dp
196      END IF
197      IF (imol > 0) nbdmol (imol) = nbdmol (imol) + 1
198  END DO
199
200  DO i = 1, nmoldef
201      rnmol = NINT (nmol (i))
202      IF (rnmol>0) THEN
203          nbdmol (i) = nbdmol (i) / rnmol
204      END IF
205  END DO
206
207  nbdmolmx = MAXVAL (nbdmol (1:nmoldef))
208
209  ! obtain connectivity information (needed only once)
210  CALL connect (nmbeads, numbond, bndtbl, nbdmolmx, visit, from)
211
212  ! Checking for charge neutrality of all molecules
213  ALLOCATE (molchg (nummol))
214
215  molchg (:) = 0.0_dp
216
217  DO i = 1, nmbeads

```

(continues on next page)

(continued from previous page)

```

217     imol = mole (i)
218     molchg (imol) = molchg (imol) + chg (ltp (i))
219 END DO
220
221 DO i = 1, nummol
222     IF (ABS (molchg (i))>1.0e-16_dp) THEN
223         WRITE (*,*) "molecule number ",i," is not neutral! (The dipole moment is,
↳frame-dependent)"
224         WRITE (*,*) "its charge is=", molchg (i)
225         WRITE (*,*) "its type is=", nammol (i)
226         STOP
227     ENDIF
228 END DO
229
230 CALL check_molecules ! checks that beads are labelled as expected
231
232 ! Get the maximum number of time steps for autocorrelation
↳
233 ! and adjust it if necessary
234
235 WRITE (*,*) "Number of time steps in autocorrelation profile? "
236 READ (*,*) naf
237 IF (naf<1 .OR. naf>numtraj) naf = numtraj
238
239 ! Get the switch for FFT computation
240
241 WRITE (*,*) "switch for FFT computation? (1=yes, 0 or any other integer=no)"
242 READ (*,*) n1
243 lfft = (n1 == 1)
244
245 ALLOCATE (mdipdata (4, nummol, numtraj))
246 ALLOCATE (dipx (nummol), dipy (nummol), dipz (nummol))
247
248 !reading trajectories and computing charge dipole moments
249 ALLOCATE (dipx_box (nmoldef), dipy_box (nmoldef), dipz_box (nmoldef))
250
251 eof = .false.
252
253 DO k = 1, numtraj
254
255     mypos = currentpos + INT (k-1, KIND=li) * ((numbeadsli + 1_li) * leni_li +
↳(framesizeli * numbeadsli + 7_li) * lend_li)
256     READ (ntraj, POS = mypos, IOSTAT=ioerror) time, nbeads, dimx, dimy, dimz,
↳shrdx, shrdy, shrdz
257
258     IF (ioerror/=0) THEN
259         eof = .true.
260         IF (k==1) THEN
261             PRINT *, 'ERROR: cannot find trajectory data in HISTORY files'
262             STOP
263         END IF
264         EXIT
265     END IF
266
267     READ (ntraj) readint (1:nsyst)
268     DO i = 1, nsyst
269         global = readint (i)

```

(continues on next page)

(continued from previous page)

```

270     READ (ntraj) readdata (1:framesize)
271     IF (global>nusyst .AND. global<=nsyst) THEN
272         xxx (global-nusyst) = readdata (1)
273         yyy (global-nusyst) = readdata (2)
274         zzz (global-nusyst) = readdata (3)
275     END IF
276 END DO
277
278 CALL compute_charge_dipoles (dipx_box, dipy_box, dipz_box, dipx, dipy, dipz)
279
280     ! the dipole components for each individual molecule are stored for all the_
↪ snapshots
281     DO j = 1, nummol
282         mdipdata (1, j, k) = dipx (j)
283         mdipdata (2, j, k) = dipy (j)
284         mdipdata (3, j, k) = dipz (j)
285         mdipdata (4, j, k) = time
286     END DO
287
288 END DO ! end of loop over trajectories
289
290 IF (k <= numtraj) THEN
291     WRITE (*,*) "ERROR: problem with the number of snapshots!"
292     STOP
293 END IF
294
295 nsamp = numtraj - naf + 1
296
297 ALLOCATE (corrdata (naf))
298
299 ! define FFT size if needed
300
301 IF (lfft) THEN
302     nftpts = naf ! modify here to change the size of the DFT
303     domega = 2 * pi / (dt * nftpts)
304     ALLOCATE (fftdata (nftpts))
305 END IF
306
307 ! Open output file, compute the autocorrelation and write it there
308
309 nrtout = ntraj + 1
310
311 IF (numtraj>0) THEN
312
313     OPEN (nrtout, file='MDIPAFDAT', status='replace')
314     WRITE (nrtout, '(a80)') text
315     WRITE (nrtout, '(2i10)') numtraj,naf
316     WRITE (nrtout, '(/)')
317
318     ! Open the FT output file if needed
319     IF (lfft) THEN
320         OPEN (nrtout+1, file='MDIPAFFFT', status='replace')
321         WRITE (nrtout+1, '(a80)') text
322         WRITE (nrtout+1, '(2i10)') numtraj,nftpts
323         WRITE (nrtout+1, '(/)')
324     END IF
325

```

(continues on next page)

(continued from previous page)

```

326     imol = 0 ! counter for molecules
327     DO j = 1, nmoldef
328         rnmol = NINT (nmol (j))
329         corrddata = 0.0_dp
330         WRITE (nrtout, '(a8)') nammol (j)
331         IF (lfft) WRITE (nrtout+1, '(a8)') nammol (j)
332         DO i = 1, nsamp
333             DO k = imol + 1, imol + rnmol
334                 dx0 = mdipdata (1, k, i)
335                 dy0 = mdipdata (2, k, i)
336                 dz0 = mdipdata (3, k, i)
337                 DO l = 1, naf
338                     corrddata (l) = corrddata (l) + mdipdata (1, k, i+l-1) * dx0 + mdipdata
↪ (2, k, i+l-1) * dy0 &
339                                     + mdipdata (3, k, i+l-1) * dz0
340                 END DO
341             END DO
342         END DO
343         corrddata = corrddata / (REAL (nsamp, KIND=dp) * nmol (j))
344         imol = imol + rnmol
345
346         DO i = 1, naf
347             WRITE (nrtout, '(1p,3e14.6)') REAL (i-1, KIND=dp)*dt, corrddata (i),
↪ corrddata (i)/corrddata(1)
348             END DO
349             WRITE (nrtout, '(/)')
350             IF (lfft) THEN
351                 fftdata (:) = corrddata (:) / corrddata (1) ! adapt here if nftpts differs
↪ from naf
352                 CALL fft (fftdata)
353                 DO i = 1, nftpts
354                     WRITE (nrtout+1, '(1p,3e14.6)') REAL (i-1, KIND=dp)*domega, fftdata (i)
355                 END DO
356                 WRITE (nrtout+1, '(/)')
357             END IF
358         END DO
359         corrddata = 0.0_dp
360         WRITE (nrtout, '("all species")')
361         IF (lfft) WRITE (nrtout+1, '("all species")')
362         DO i = 1, nsamp
363             DO k = 1, nummol
364                 dx0 = mdipdata (1, k, i)
365                 dy0 = mdipdata (2, k, i)
366                 dz0 = mdipdata (3, k, i)
367                 DO l = 1, naf
368                     corrddata (l) = corrddata (l) + mdipdata (1, k, i+l-1) * dx0 + mdipdata
↪ (2, k, i+l-1) * dy0 &
369                                     + mdipdata (3, k, i+l-1) * dz0
370                 END DO
371             END DO
372         END DO
373         corrddata = corrddata / (REAL (nsamp, KIND=dp) * nummol)
374
375         DO i = 1, naf
376             WRITE (nrtout, '(1p,3e14.6)') REAL (i-1, KIND=dp)*dt, corrddata (i),
↪ corrddata (i)/corrddata(1)
377             END DO

```

(continues on next page)

(continued from previous page)

```

378     WRITE (nrtout, '/')
379     IF (lfft) THEN
380         fftdata (:) = corrddata (:) / corrddata (1) ! adapt here if nftpts differs
↳ from naf
381         CALL fft (fftdata)
382         DO i = 1, nftpts
383             WRITE (nrtout+1, '(1p,3e14.6)') REAL (i-1, KIND=dp)*domega, fftdata (i)
384         END DO
385         WRITE (nrtout+1, '/')
386     END IF
387 END IF
388
389 ! Close the trajectory file
390 CLOSE (ntraj)
391
392 ! Close the output files
393 CLOSE (nrtout)
394 IF (lfft) CLOSE (nrtout+1)
395
396 DEALLOCATE (readint, readdata)
397 DEALLOCATE (namspe, nammol)
398 DEALLOCATE (xxx, yyy, zzz)
399 DEALLOCATE (ltp, ltm, mole)
400 DEALLOCATE (nmol, nbdmol)
401 DEALLOCATE (chg, molchg)
402 DEALLOCATE (dipx_box, dipy_box, dipz_box)
403 DEALLOCATE (bndtbl)
404 DEALLOCATE (visit, from)
405 DEALLOCATE (mdipdata, corrddata)
406 DEALLOCATE (dipx, dipy, dipz)
407 IF (lfft) DEALLOCATE (fftdata)
408
409 CONTAINS
410
411 SUBROUTINE check_molecules
412 !*****
413 ! subroutine to check molecular content and labelling
414 !
415 ! authors: s. chiacchiera, February 2017
416 !
↳ *****
↳
417     IMPLICIT NONE
418     INTEGER i, j, k, tm, tp, imol, im, ibd
419     INTEGER mxmolsize
420     INTEGER, ALLOCATABLE :: molbeads (:,:)
421
422     mxmolsize = 0
423     DO i = 1, nmoldef
424         mxmolsize = MAX (nbdmol(i), mxmolsize)
425     END DO
426     ALLOCATE (molbeads (nmoldef, mxmolsize))
427     molbeads (:,:) = 0
428
429     imol = 0
430     ibd = 0
431     DO i = 1, nmoldef

```

(continues on next page)

(continued from previous page)

```

432      DO j = 1, NINT (nmol(i))
433        imol = imol +1
434        DO k = 1, nbdmol(i)
435          ibd = ibd +1
436          tm = ltm (ibd)
437          tp = ltp (ibd)
438          im = mole (ibd)
439          IF (j==1) THEN
440            molbeads (i, k) = tp
441          ELSE
442            IF (molbeads (i, k) /= tp) THEN
443              WRITE (*,*) "ERROR: Problem with molecular content!"
444              STOP
445            ENDIF
446          ENDIF
447          IF (tm/=i.OR.im/=imol) THEN
448            WRITE (*,*) "ERROR: Problem with molecules labels!"
449            STOP
450          ENDIF
451        END DO
452      END DO
453    END DO
454    IF (imol/=nummol) THEN
455      WRITE (*,*) "ERROR: imol and nummol differ!"
456      STOP
457    ENDIF
458    DEALLOCATE (molbeads)
459    RETURN
460  END SUBROUTINE check_molecules

461  SUBROUTINE compute_charge_dipoles (dipx_box, dipy_box, dipz_box, px, py, pz)
462  ! *****
463  ! subroutine to compute charge dipole moments
464  !
465  ! authors: m. a. seaton and s. chiacchiera, February 2017
466  !
467  ! input: xxx, yyy, zzz (at a given time step) and chg
468  ! input: visit and from (obtained using connect)
469  ! output: the x,y,z components of the total dipole, for each molecule type and all
470  !         individual dipoles (at a given time step)
471  !
472  !
473  ! (NB: this is a slightly modified version, with different output)
474  !
475  ! *****
476  ! *****
477  IMPLICIT NONE
478  INTEGER i, j, k, tm, tp, imol, ibd, count, ipr
479  REAL(KIND=dp), DIMENSION(nmoldef) :: dipx_box, dipy_box, dipz_box
480  REAL(KIND=dp) :: x, y, z, dx, dy, dz, xpre, ypre, zpre
481  REAL(KIND=dp) :: dipx, dipy, dipz
482  REAL(KIND=dp), DIMENSION(nmbeads) :: xabs, yabs, zabs
483  REAL(KIND=dp), DIMENSION(nummol) :: px, py, pz
484
485  dipx_box (:) = 0._dp
486  dipy_box (:) = 0._dp
487  dipz_box (:) = 0._dp

```

(continues on next page)

(continued from previous page)

```

487     imol = 0
488     count = 0
489     ! xabs = 0._dp ! just to keep it clean
490     ! yabs = 0._dp
491     ! zabs = 0._dp
492
493     DO i = 1, nmoldef
494         tm = i
495         DO j = 1, NINT (nmol(i))
496             imol = imol + 1
497
498             dipx = 0._dp ! dipole of a SINGLE molecule
499             dipy = 0._dp
500             dipz = 0._dp
501
502             DO k = 1, nbdmol(i)
503                 count = count + 1
504                 ibd = visit (count)
505                 ipr = from (count)
506
507                 IF (ipr /= 0) THEN
508                     xpre = xabs (ipr)
509                     ypre = yabs (ipr)
510                     zpre = zabs (ipr)
511                 ELSE
512                     IF (k == 1) THEN
513                         xpre = 0._dp
514                         ypre = 0._dp
515                         zpre = 0._dp
516                     ELSE
517                         WRITE (*,*) "Unconnected molecule!"
518                         STOP
519                     ENDIF
520                 ENDIF
521
522                 tp = ltp (ibd)
523
524                 dx = xxx (ibd) - xpre
525                 dy = yyy (ibd) - ypre
526                 dz = zzz (ibd) - zpre
527
528                 dx = dx - dimx * ANINT (dx/dimx)
529                 dy = dy - dimy * ANINT (dy/dimy)
530                 dz = dz - dimz * ANINT (dz/dimz)
531
532                 x = xpre + dx
533                 y = ypre + dy
534                 z = zpre + dz
535
536
537                 dipx = dipx + x * chg (tp)
538                 dipy = dipy + y * chg (tp)
539                 dipz = dipz + z * chg (tp)
540
541                 xabs (ibd) = x
542                 yabs (ibd) = y
543                 zabs (ibd) = z

```

(continues on next page)

(continued from previous page)

```

544
545         END DO
546
547         ! storing dipole moments of individual molecules
548         px (imol) = dipx
549         py (imol) = dipy
550         pz (imol) = dipz
551
552         dipx_box (tm) = dipx_box (tm) + dipx
553         dipy_box (tm) = dipy_box (tm) + dipy
554         dipz_box (tm) = dipz_box (tm) + dipz
555
556     END DO
557 END DO
558
559 IF (imol/=nummol) THEN
560     WRITE (*,*) "ERROR: imol and nummol differ!"
561     STOP
562 ENDIF
563
564 RETURN
565 END SUBROUTINE compute_charge_dipoles
566
567 SUBROUTINE fft (x)
568 !
569 ! *****
570 ! Subroutine to call FFTW (v3) one-dimensional complex DFT.
571 ! Notice that the input array is overwritten with the its Discrete Fourier Transform.
572 !
573 ! author: s. chiacchiera, August 2017
574 ! amended: m. a. seaton, January 2021
575 !
576 ! *****
577
578 USE, INTRINSIC :: iso_c_binding
579 IMPLICIT none
580 INCLUDE 'fftw3.f03'
581 COMPLEX(C_DOUBLE_COMPLEX), INTENT(INOUT) :: x (:)
582 INTEGER :: n
583 TYPE(C_PTR) :: plan
584
585 n = SIZE (x)
586
587 plan = fftw_plan_dft_1d (n, x, x, FFTW_FORWARD, FFTW_ESTIMATE)
588 CALL fftw_execute_dft (plan, x, x)
589 CALL fftw_destroy_plan (plan)
590
591 RETURN
592
593 END SUBROUTINE fft
594
595 END PROGRAM gen_moldipaf
596
597 SUBROUTINE connect (nbeads, nbonds, bndtbl, mxmolsize, visit, from)
598 ! *****
599 ! Analyzes all the bonds (bndtbl) to obtain a schedule (visit, from)

```

(continues on next page)

(continued from previous page)

```

597 ! to visit the beads so that each cluster is visited along a connected
598 ! path. "visit" gives the order to include beads, "from" gives the bead
599 ! to attach them to.
600 ! (Note: vocabulary from infection propagation used to move along
601 ! clusters)
602 !
603 ! author: s. chiacchiera, February 2017
604 ! amended: m. a. seaton, January 2021
605 ! *****
606 IMPLICIT none
607 INTEGER, INTENT (IN) :: nbeads, nbonds
608 INTEGER, INTENT (IN) :: bndtbl (nbonds,2)
609 INTEGER, INTENT (IN) :: mxmolsize
610 INTEGER :: ic, i, k, nn, nclu, nper, lab, ref, count !j
611 INTEGER, ALLOCATABLE :: firstnn (:), lastnn (:), deg (:)
612 INTEGER, ALLOCATABLE :: labnn (:)
613 INTEGER, ALLOCATABLE :: state (:)
614 INTEGER, ALLOCATABLE :: perlab (:), perref (:)
615 INTEGER, ALLOCATABLE :: nchist (:)
616 INTEGER, INTENT (OUT) :: visit (nbeads), from (nbeads)
617
618 ALLOCATE (firstnn (nbeads), lastnn (nbeads), deg (nbeads), labnn (2*nbonds))
619 ALLOCATE (state (nbeads))
620 ALLOCATE (perlab (nbeads), perref (nbeads))
621 ALLOCATE (nchist (mxmolsize))
622 !-----
623 CALL organize (nbeads, nbonds, labnn, firstnn, lastnn, deg)
624 !-----
625 state (:) = 0
626 nchist (:) = 0
627 visit (:) = 0
628 from (:) = 0
629 count = 0
630 !-----
631 ic = 0
632 !-----
633 DO WHILE (ic < nbeads) ! ic = label of bead used to "grow" a cluster
634   ic = ic + 1
635   IF (state (ic) /= 0) THEN
636     WRITE (*,*) "ERROR: labels are not as expected!"
637     STOP
638   END IF
639   nclu = 1
640   count = count + 1
641   visit (ic) = ic
642   IF (deg (ic) == 0) THEN
643     state (ic) = -1
644     IF (nclu <= mxmolsize) nchist (nclu) = nchist (nclu) + 1
645     CYCLE
646   END IF
647   state (ic) = 1 ! ic is "infected"
648
649   ! nearest neighbours of ic are marked as "goint to be infected" -> a.k.a.
650   ↪perimeter
651   nper = 0
652   perlab (:) = 0
653   perref (:) = 0

```

(continues on next page)

(continued from previous page)

```

653     DO k = firstnn (ic), lastnn (ic)
654         nn = labnn (k)
655         IF ( state (nn) /= 0) THEN
656             WRITE (*,*) "ERROR: labels are not as expected!"
657             STOP
658         END IF
659         nper = nper + 1
660         perlab (nper) = nn !new bead in perimeter
661         perref (nper) = ic !its reference bead (origin of the link)
662         state (nn) = 2
663     END DO
664     state (ic) = 3 ! ic is "dead"
665
666     DO WHILE (nper > 0)
667         i = 1 ! pick a bead of "perimeter" to be analyzed
668         lab = perlab (i)
669         ref = perref (i)
670         perlab (i) = perlab (nper)
671         perref (i) = perref (nper)
672         nper = nper - 1
673         IF (state (lab) == 3) THEN
674             CYCLE
675         END IF
676         state (lab) = 1 ! "lab" is added to the cluster
677         nclu = nclu + 1
678         count = count + 1
679         visit (count) = lab
680         from (count) = ref
681
682         DO k = firstnn (lab), lastnn (lab) ! check nn of newly added
683             nn = labnn (k)
684             IF ( (state (nn) == 2) .OR. (state (nn) == 3)) CYCLE
685             nper = nper + 1
686             perlab (nper) = nn !new bead in perimeter
687             perref (nper) = lab !its reference bead (origin of the link)
688             state (nn) = 2
689         END DO
690         state (lab) = 3
691
692     END DO
693     nchist (nclu) = nchist (nclu) +1
694     ic = ic + nclu - 1 ! prepare ic for the next cluster
695 END DO
696 WRITE (*,*) "nchist: ", nchist
697 !-----
698 DEALLOCATE (firstnn, lastnn, deg, labnn)
699 DEALLOCATE (state)
700 DEALLOCATE (perlab, perref)
701 DEALLOCATE (nchist)
702 RETURN
703 !-----
704 CONTAINS
705 !-----
706 SUBROUTINE organize (N, NL, labnn, firstnn, lastnn, deg)
707 !*****
708 ! Analyzes the bonds (bndtbl) to obtain the degree (=number of bonds)
709 ! of each bead, and the nearest neighbours list.

```

(continues on next page)

(continued from previous page)

```

710      ! N in the number of beads (vertices) and NL of bonds (links).
711      !
712      ! author: s. chiacchiera, February 2017
713      ! *****
714      IMPLICIT none
715      INTEGER, INTENT(IN) :: N, NL
716      INTEGER :: i,l,count_lab, i1,i2
717      INTEGER, DIMENSION (N), INTENT(OUT) :: deg
718      INTEGER, DIMENSION (N), INTENT(OUT) :: firstnn, lastnn
719      INTEGER, DIMENSION (2*NL), intent(OUT) :: labnn
720
721      deg(:)=0
722      firstnn(:)=0
723      lastnn(:)=0
724      labnn(:)=0
725
726      count_lab=0
727
728      DO i=1,N
729          DO l=1,NL
730              IF (bndtbl(l,1).EQ.i) THEN
731                  deg(i)=deg(i)+1
732                  count_lab=count_lab+1
733                  labnn(count_lab)=bndtbl(l,2)
734              ENDIF
735              IF (bndtbl(l,2).EQ.i) THEN
736                  deg(i)=deg(i)+1
737                  count_lab=count_lab+1
738                  labnn(count_lab)=bndtbl(l,1)
739              ENDIF
740          END DO
741      END DO
742
743      i1=1
744      i2=0
745      DO i=1,N
746          IF (deg(i)==0) CYCLE
747          firstnn(i)=i1
748          i2=i1+deg(i)-1
749          lastnn(i)=i2
750          i1=i2+1
751      END DO
752
753      RETURN
754
755      END SUBROUTINE organize
756      ! -----
757      END SUBROUTINE connect
758

```

Consistency check of input files in DL_MESO_DPD

Software Technical Information**Language** FORTRAN 90**Licence** BSD**Documentation Tool** RST and LaTeX-generated .pdf file**Application Documentation** [Click to download the manual with more details](#)**Relevant Training Material** See the Testing section

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*

Purpose of Module

This module, `check_config.f90`, is a pre-processing utility for DL_MESO_DPD, the Dissipative Particle Dynamics (DPD) code from the [DL_MESO](#) package. It checks that the content of the optional configuration (CONFIG) file is consistent with that of the necessary input files (CONTROL and FIELD). In particular, it checks: the system dimensions, its composition and the bead content of all the molecules. In addition, in case hard walls are present, it checks that none of the stretching bonds between beads crosses a hard wall.

Background Information

The base code for this module is DL_MESO_DPD, the Dissipative Particle Dynamics code from the mesoscopic simulation package [DL_MESO](#), developed by M. Seaton at Daresbury Laboratory. This open source code is available from STFC under both academic (free) and commercial (paid) licenses. The module can be used with DL_MESO from version 2.6 (dated November 2015) onwards, including its currently released version, version 2.7 (dating December 2018).

Testing

The present module, `check_config.f90`, is compiled with the available Fortran90 compiler¹, e.g.:

```
gfortran -o check_config.exe check_config.f90
```

and the executable must be in the same directory of the three files to be analyzed (i.e., CONTROL, FIELD and CONFIG).

When running `check_config.f90`, the outcome of the different checks is sent to the standard output. The most important messages are: warnings, error messages and hints to fix them. For completeness, some information about the system size and composition is printed too.

We suggest as a test a very small system with three species of beads (A, B, C) and a total population of 24 beads. Of these, 6 are unbonded, while the others are grouped into 7 molecules of two types. In the first test, consistent input is

¹ Compilation has been tested with the GNU compiler GCC, version 10.2.0.

given. In the following ones, small changes rising warnings and errors are analyzed, to demonstrate the behaviour of the module.

Test 1

Use for the CONTROL file

```
Simple test

temperature 1.0
cutoff 1.0

timestep 0.01
steps 1100
equilibration steps 100
trajectory 100 100 0
stats every 100
stack size 100
print every 100
job time 100.0
close time 10.0

ensemble nvt mdvv

nfold 1 1 1
#vol 1.0
conf zero
#surface hard z

finish
```

for the FIELD file

```
Simple test

SPECIES 3
A 1.0 0.0 0
B 1.0 0.0 6
C 1.0 0.0 0

MOLECULES 2
ACB
nummols 4
beads 3
A 0.0 0.5 0.0
C 0.0 0.0 0.0
B 0.5 0.0 0.0
bonds 2
harm 1 2 10.0 0.0
harm 2 3 10.0 0.0
finish
BC
nummols 3
beads 2
B 0.0 0.0 0.0
C 0.5 0.0 0.0
bonds 1
harm 1 2 10.0 0.0
finish
```

(continues on next page)

(continued from previous page)

```

INTERACTIONS 3
A A dpd 25.0 1.0 4.0
B B dpd 25.0 1.0 4.0
C C dpd 25.0 1.0 4.0

CLOSE

```

and for the CONFIG file this (correct labelling) one, where the beads are randomly located in the cubic box

```

Simple test
      0      1
1.0000000000 0.0000000000 0.0000000000
0.0000000000 1.0000000000 0.0000000000
0.0000000000 0.0000000000 1.0000000000
B 1
0.4577200045 0.9001190080 0.3001750172
B 2
0.0415166244 0.7699064654 0.8179705041
B 3
0.6302680192 0.9146029274 0.7314079348
B 4
0.7731659040 0.5993543351 0.3483148324
B 5
0.1273913826 0.0669681234 0.5332509871
B 6
0.3493437595 0.4205682036 0.4898004159
A 7
0.0755944215 0.5154423406 0.0394230825
C 8
0.5837477096 0.0477604149 0.7092934456
B 9
0.0949452841 0.7453901460 0.7721903180
A 10
0.2026802865 0.4475765512 0.4191000671
C 11
0.8148312410 0.8686744347 0.8112311619
B 12
0.3621449634 0.5704018599 0.7440643976
A 13
0.4903370300 0.0944675650 0.7163648810
C 14
0.9445609725 0.2362723351 0.0291370763
B 15
0.7068423470 0.8993323711 0.0791676911
A 16
0.5713842548 0.2551756180 0.7366135404
C 17
0.2637800160 0.3507307479 0.7316829655
B 18
0.3956074216 0.9739386044 0.9861309514
B 19
0.9029085375 0.1837974484 0.0837168293
C 20
0.0287508243 0.2151038377 0.2502012593
B 21
0.1325665768 0.0464577116 0.8147593457

```

(continues on next page)

(continued from previous page)

```

C    22
    0.6603299794    0.1659685862    0.4340299834
B    23
    0.7419336708    0.6792113832    0.5057230908
C    24
    0.6879917453    0.0772687141    0.6552782347

```

Running the utility `check_config.f90`, this output is printed on the standard output

```

unit cell sizes      =      1.0000000000      1.0000000000      1.0000000000
nfoldx, nfoldy, nfoldz =          1          1          1
system sizes        =      1.0000000000      1.0000000000      1.0000000000
imcon               =          1
levcfg              =          0
lconfzero           =          T
srftype             =          0
nspe                 =          3
nmoldef             =          2
mxmolsize           =          3
mxbonds             =          2
nspec                =          0          6          0
nspecmol             =          4          7          7
numbond             =         11
for molecule ACB      :
mlstrtspe           =          1          3          2
for molecule BC       :
mlstrtspe           =          2          3          0

OK: CONFIG file is consistent with FIELD file
(composition and bead content of molecules)

```

Test 2

Instead, altering just two particle species in the CONFIG file given above:

- “B 3” changes into “A 3”
- “C 20” changes into “B 20”

an error message is given

```

error: problem with unbonded beads of species A      :          1 instead of      ↵
↪          0
error: problem with unbonded beads of species B      :          5 instead of      ↵
↪          6
error: problem with molecular beads of species B      :          8 instead of      ↵
↪          7
error: problem with molecular beads of species C      :          6 instead of      ↵
↪          7
error: problem with the molecular content of  BC      :          2 -th bead is B      ↵
↪ instead of C      (bead label =          20 )
error: CONFIG file is not consistent with FIELD file

```

Test 3

If instead these two lines of the CONFIG file are altered

- “A 10” into “C 10”

- “C 11” into “A 11”

the error message is

```
error: problem with the molecular content of  ACB      :      1 -th bead is C
↪      instead of A      (bead label =      10 )
error: problem with the molecular content of  ACB      :      2 -th bead is A
↪      instead of C      (bead label =      11 )

error: CONFIG file is not consistent with FIELD file
```

Test 4

Here instead we propose to add a hard wall orthogonal to the z axis: this is done by uncommenting the surface hard z line in the CONTROL file. Running the utility, one obtains

```
srftype =      1
srfx, srfy, srfz =      0      0      1
(composition and bead content of molecules)
error: bond between beads      7 and      8 crosses hard wall perp. to z
error: bond between beads     13 and     14 crosses hard wall perp. to z
```

Source Code

To download the source code for `check_config.f90`, [click here](#).

Analysis of local tetrahedral ordering for DL_MESO_DPD

Software Technical Information

Language Fortran 2003

Licence BSD

Documentation Tool RST

Application Documentation See the Source Code section

Relevant Training Material See the Testing section

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*

Purpose of Module

This module, `tetrahedral.f90`, is a postprocessing utility for DL_MESO_DPD, the Dissipative Particle Dynamics (DPD) code from the [DL_MESO](#) package. It processes trajectory (HISTORY) files and analyzes the local

tetrahedral ordering, a feature that is relevant, for example, in water-like systems.

The local ordering in liquid water can be assessed considering the coordinates of oxygen atoms [Duboue2015]. In particular, for each oxygen, its four nearest neighbouring oxygens are considered, whereas the hydrogens are disregarded. At the mesoscale level, the user will select one (appropriate) bead species and analyze its local ordering.

Given a particle j , we first find its *four nearest neighbours* (n.n.). Then, an *orientational tetrahedral order parameter* is built using $q = 1 - \frac{3}{8} \sum_{i=1}^3 \sum_{k=i+1}^4 \left(\cos \psi_{ik} + \frac{1}{3} \right)^2$, where i, k are n.n. of j and $\psi_{ik} = \theta_{ijk}$ is the angle¹ between the particles i, j and k . Of course, the quantity is then averaged over the central particle j and over time.

A *translational tetrahedral order parameter*, S_k , is defined as $S_k = 1 - \frac{1}{3} \sum_{i=1}^4 \frac{(r_i - \bar{r})^2}{4\bar{r}^2}$, where i is a n.n. of j and $\bar{r} = \frac{1}{4} \sum_{i=1}^4 r_i$.

Concerning the limiting values of these parameters: in a regular tetrahedron (if the four vertices are referred to the center of the solid) one has $q = S_k = 1$. In an ideal gas, where the angle ψ_{ik} is randomly distributed, $q \simeq 0$. On the other hand, $S_k \simeq 0$ if the density fluctuations are large enough.

As a result of the analysis, a file *TETRADAT* is produced, whose columns are snapshot index, q, S_k , the instantaneous values of the order parameters defined above. At the end of the file, the averages and standard errors (computed assuming the snapshots are uncorrelated) of both order parameters are given.

Background Information

The base code for this module is DL_MESO_DPD, the Dissipative Particle Dynamics code from the mesoscopic simulation package DL_MESO, developed by M. Seaton at Daresbury Laboratory. This open source code is available from STFC under both academic (free) and commercial (paid) licenses. The module is to be used with DL_MESO in its latest released version, version 2.7 (released December 2018). A variant of this module to be used with a previous version of DL_MESO, version 2.6 (dating November 2015), can be found in the `old-v2.6` directory.

Testing

The utility `tetrahedral.f90` is compiled with the available Fortran 2003 compiler², e.g.:

```
gfortran -o tetrahedral.exe tetrahedral.f90
```

and the executable must be in the same directory of the HISTORY file. The user is asked to provide the number of the species for which ordering has to be analyzed. To input the user-defined parameter, one can enter it interactively at runtime or write it into a text file (say, `input.txt`) and run the program in this way:

```
tetrahedral.exe < input.txt
```

Below we propose a test where a fluid is prepared in an ordered configuration (diamond cubic lattice) and rapidly goes into an orientationally disordered one.

Test

The sources used for this test are available to download.

Run the DL_MESO_DPD simulation on a single node (serial run) using the *CONTROL* file,

```
One species - starting as diamond cubic lattice
#volume 64.0
temperature 1.0
```

(continues on next page)

¹ The angle $\theta_{ijk} = \cos^{-1} \left\{ \frac{\vec{r}_{ij} \cdot \vec{r}_{kj}}{r_{ij} r_{kj}} \right\}$ where $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$ and $r = |\vec{r}|$.

² Compilation has been tested with the GNU compiler GCC, version 10.2.0.

(continued from previous page)

```

cutoff 1.0

timestep 0.01
steps 1000
traj 0 10 0
#traj 500 100 0
stats every 100
stack size 100
print every 100
job time 7200.0
close time 100.0

ensemble nvt mdvv

conf zero

nfold 2,2,2

finish

```

the *FIELD* file

```

One species - starting as diamond cubic lattice

SPECIES 1
A 1.0 0.0 8 0

INTERACTIONS 1
A A dpd 25.0 1.0 4.0

CLOSE

```

and the *CONFIG* file

```

One species - starting as diamond cubic lattice
      0      1
      4.0000000000  0.0000000000  0.0000000000
      0.0000000000  4.0000000000  0.0000000000
      0.0000000000  0.0000000000  4.0000000000
A      1
A      2
A      3
A      4
A      5
A      6
A      7
A      8

```

| | | |
|-----|-----|-----|
| 0.0 | 0.0 | 0.0 |
| 0.0 | 2.0 | 2.0 |
| 2.0 | 0.0 | 2.0 |
| 2.0 | 2.0 | 0.0 |
| 3.0 | 3.0 | 3.0 |
| 3.0 | 1.0 | 1.0 |
| 1.0 | 3.0 | 1.0 |
| 1.0 | 1.0 | 3.0 |

This configuration corresponds to a diamond cubic lattice³, while the `nfold` directive in the *CONTROL* file replicates the configuration twice in all three dimensions.

Analyzing the resulting trajectory (HISTORY) file with `tetrahedral.exe` (compiled as indicated above) and inputting 1 for the runtime argument, the following output is printed to the standard output:

```
Species          1 : A
Which species number has to be analyzed?
1
total number of beads:          64
number of beads by species:      64
number of analyzed beads:        64
<q> =      0.132069E+00
error =      0.159218E-01
<s_k> =      0.986906E+00
error =      0.243544E-03
```

The output file *TETRADAT*

```
# Local ordering for beads of species: A
# dimx, dimy, dimz=  0.0000000000000000      0.0000000000000000      0.
→0000000000000000
# snapshot number, q, sk
  1  1.000000E+00  1.000000E+00
  2  9.690709E-01  9.982591E-01
  3  8.476516E-01  9.935607E-01
  4  5.326193E-01  9.892793E-01
  5  3.297230E-01  9.868977E-01
  6  2.057773E-01  9.847919E-01
  7  9.033514E-02  9.829358E-01
  8  2.481253E-02  9.827499E-01
  9  4.596717E-02  9.831773E-01
 10  4.275300E-02  9.840823E-01
```

contains the values of q and S_k for each snapshot and their averages are also produced.

One can see that in the initial snapshot, both order parameters detect an ordered state (i.e., $S_k = q = 1$). With the evolution in time, since the system is a dilute fluid without bonds between particles, the orientational ordering is rapidly lost (i.e., $q \simeq 0$). On the other hand, the translational order parameter stays close to one since the density of the system is roughly uniform.

Source Code

You can directly download the source file `tetrahedral.f90` and we also include its contents below (as well as in the test tarball).

```
1 PROGRAM tetrahedral
2 ! *****
3 !
4 ! module to analyze tetrahedral ordering in dl_meso HISTORY files
5 !
6 ! authors - m. a. seaton & s. chiacchiera, january 2018 (tidied up and amended
```

(continues on next page)

³ The diamond cubic crystal lattice (https://en.wikipedia.org/wiki/Diamond_cubic) is a repeating pattern of 8 atoms. Their coordinates may be given as: $A = (0, 0, 0)$, $B = (0, 2, 2)$, $C = (2, 0, 2)$, $D = (2, 2, 0)$, $E = (3, 3, 3)$, $F = (3, 1, 1)$, $G = (1, 3, 1)$, and $H = (1, 1, 3)$ in a unit cubic cell of side $L = 4$. One can check that, with the minimum image convention, each particle has its 4 closest neighbours at a distance $\sqrt{3}$, and all the angles are $\arccos(-1/3)$. For this configuration (also if repeated periodically along the three Cartesian axis), $q = 1$ and $S_k = 1$.

(continued from previous page)

```

7  !           january 2021)
8  !
9  ! *****
10 IMPLICIT none
11 INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND (15, 307)
12 INTEGER, PARAMETER :: si = SELECTED_INT_KIND (8)
13 INTEGER, PARAMETER :: li = SELECTED_INT_KIND (12)
14 INTEGER, PARAMETER :: endversion = 1
15
16 REAL(KIND=dp), PARAMETER :: pi=3.141592653589793_dp
17
18 INTEGER, PARAMETER :: ntraj=10
19
20 CHARACTER(80) :: text
21 CHARACTER(8), ALLOCATABLE :: namspe (:), nammol (:)
22
23 INTEGER, ALLOCATABLE :: ltp (:), nspec (:), readint (:)
24 INTEGER :: nrtout
25 INTEGER :: chain, ioerror, i, numtraj, j, k, nmoldef, ibond
26 INTEGER :: nspe, nbears, nusyst, nsyst, numbond, global, species, molecule
27 INTEGER :: lfrzn, keytrj, srfx, srfy, srfz
28 INTEGER :: nav
29 INTEGER :: bead1, bead2
30 INTEGER :: endver, Dlen, nstep, framesize, lend
31 INTEGER(KIND=li) :: filesize
32
33 REAL(KIND=dp), ALLOCATABLE :: xxx (:), yyy (:), zzz (:), readdata (:)
34 REAL(KIND=dp) :: dimx, dimy, dimz, shrdx, shrdy, shrdz
35 REAL(KIND=dp) :: amass, rcii, chg
36 REAL(KIND=dp) :: time
37
38 LOGICAL :: eof, swapend, bigend
39
40 ! Variables for tetrahedral ordering
41 INTEGER :: nnlab(4), npart, sp, count
42 REAL(KIND=dp) :: qtetra, stetra
43 REAL(KIND=dp) :: q, sk, q_sum, sk_sum, q_ave, sk_ave
44 REAL(KIND=dp) :: q2_sum, sk2_sum, q2_ave, sk2_ave
45
46 !-----
47 ↪-----
48
49 ! determine number of bytes for selected double precision kind
50 ! (the default SELECTED_REAL_KIND (15, 307) should return 8 bytes)
51
52 lend = STORAGE_SIZE (1.0_dp) / 8
53
54 ! check endianness of machine
55
56 bigend = (IACHAR(TRANSFER(1,"a"))==0)
57
58 ! Determine if HISTORY file exists, which endianness to use,
59 ! if type of real is correct
60
61 INQUIRE (file = 'HISTORY', EXIST = eof)
62 IF (.NOT. eof) THEN
63     PRINT *, "ERROR: cannot find HISTORY file"

```

(continues on next page)

(continued from previous page)

```

63     STOP
64 END IF
65
66 OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted', status =
↪ 'unknown')
67
68 swapend = .false.
69 READ (ntraj) endver, Dlen
70
71 IF (endver/=endversion) THEN
72     swapend = .true.
73     CLOSE (ntraj)
74     IF (bigend) THEN
75         OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted', status_
↪ = 'unknown', convert = 'little_endian')
76     ELSE
77         OPEN (ntraj, file = 'HISTORY', access = 'stream', form = 'unformatted', status_
↪ = 'unknown', convert = 'big_endian')
78     END IF
79     READ (ntraj) endver, Dlen
80     IF (endver/=endversion) THEN
81         PRINT *, "ERROR: corrupted HISTORY file or created with incorrect version of DL_
↪ MESO"
82     STOP
83     END IF
84 END IF
85
86 IF (Dlen/=lend) THEN
87     PRINT *, "ERROR: incorrect type of real number used in HISTORY file"
88     PRINT *, "      recompile tetrahedral.f90 with reals of ", Dlen, " bytes"
89     STOP
90 END IF
91
92 ! read file size, number of trajectory frames and timestep numbers
93
94 READ (ntraj) filesize, numtraj, nstep
95
96 ! Read the number of beads, molecules and bonds
97 ! Arrays are filled with names of particles and molecules: if checking molecules,
98 ! arrays for species, molecule types etc. also filled
99
100 READ (ntraj) text
101 READ (ntraj) nspe, nmoldef, nusyst, nsyst, numbond, keytrj, srfx, srfy, srfz
102
103 ALLOCATE (namspe (nspe), nammol (nmoldef), nspec (nspe)) ! NB: nspec here counts_
↪ ALL beads of a type, not only unbonded ones
104 ALLOCATE (xxx (1:nsyst), yyy (1:nsyst), zzz (1:nsyst))
105 ALLOCATE (ltp (1:nsyst))
106
107 framesize = (keytrj+1) * 3
108 ALLOCATE (readint (1:nsyst), readdata (1:framesize))
109
110 DO i = 1, nspe
111     READ (ntraj) namspe (i), amass, rcii, chg, lfrzn
112 END DO
113
114 DO i = 1, nmoldef

```

(continues on next page)

(continued from previous page)

```

115     READ (ntraj) nammol (i)
116 END DO
117
118 ! Read properties of beads and molecules
119
120 nspec (:) = 0 ! populations
121 ibond = 0 !counter for bonds
122
123 DO i = 1, nsyst
124     READ (ntraj) global, species, molecule, chain
125     ltp (global) = species
126     nspec (species) = nspec (species) + 1
127 END DO
128
129 IF (numbond>0) THEN
130     DO i = 1, numbond
131         READ (ntraj) bead1, bead2
132     END DO
133 END IF
134
135 ! Find number of beads for which trajectories are needed
136
137 DO i = 1, nspe
138     WRITE(*,*) "Species ",i,": ",namspe (i)
139 END DO
140 WRITE(*,*) "Which species number has to be analyzed?"
141 READ(*,*) sp
142 IF (sp<0 .OR. sp>nspe) THEN
143     WRITE(*,*) "error: undefined species!"
144     STOP
145 END IF
146
147 npart = nspec (sp)
148
149 WRITE(*,*) "total number of beads:      ", nsyst
150 WRITE(*,*) "number of beads by species: ", nspec
151 WRITE(*,*) "number of analyzed beads:   ", npart
152
153 ! Open and write output file
154
155 nrtout = ntraj + 1
156 OPEN (nrtout, file = 'TETRADAT', status = 'replace')
157 WRITE (nrtout,*) "# Local ordering for beads of species: ", namspe (sp)
158 WRITE (nrtout,*) "# dimx, dimy, dimz=", dimx, dimy, dimz
159 WRITE (nrtout,*) "# snapshot number, q, sk"
160
161 eof = .false.
162 k = 0
163 nav = 0
164
165 q_sum = 0
166 sk_sum = 0
167 q2_sum = 0
168 sk2_sum = 0
169
170 ! Read snapshots of trajectories
171

```

(continues on next page)

(continued from previous page)

```

172 DO k = 1, numtraj
173   READ (ntraj, IOSTAT=ioerror) time, nbeads, dimx, dimy, dimz, shrdx, shrdy, shrdz
174   IF (ioerror/=0) THEN
175     eof = .true.
176     IF (k==1) THEN
177       WRITE (*,*) 'ERROR: cannot find trajectory data in HISTORY files'
178       STOP
179     END IF
180     EXIT
181   END IF
182
183   nav = nav + 1
184
185   ! The full coordinate arrays are used to avoid re-labelling, but they are filled_
186   ! ↳*only* for particles of species "sp"
187   xxx (:) = 0.0_dp
188   yyy (:) = 0.0_dp
189   zzz (:) = 0.0_dp
190
191   count = 0
192
193   READ (ntraj) readint (1:nbeads)
194   DO i = 1, nbeads
195     global = readint (i)
196     READ (ntraj) readdata (1:framesize)
197     IF (ltp (global) == sp) THEN
198       xxx (global) = readdata (1)
199       yyy (global) = readdata (2)
200       zzz (global) = readdata (3)
201       count = count + 1
202     END IF
203   END DO
204
205   IF (count /= npart) THEN
206     WRITE (*,*) " Number of particles of species ",sp," differs from expected!"
207     STOP
208   END IF
209
210   ! ... Analyze the trajectories (snapshot by snapshot) ...
211   q = 0.0_dp
212   sk = 0.0_dp
213
214   DO i = 1, nsyst
215     IF (ltp (i) /= sp) CYCLE
216     CALL closest4 (i, nnlab, npart)
217     ! WRITE (*,*) i, nnlab ! uncomment to see nn labels
218     CALL compute_tetra_label (i, nnlab, qtetra, stetra)
219     ! print*, "q=", qtetra ! uncomment to print q for each single set of 5 particles
220     ! print*, "s=", stetra ! uncomment to print sk for each single set of 5 particles
221     q = q + qtetra
222     sk = sk + stetra
223   END DO
224
225   q = q / REAL(npart, KIND=dp)
226   sk = sk / REAL(npart, KIND=dp)
227
228   WRITE (nrtout, '(1p,I8,2(2x,e14.6))') nav, q, sk

```

(continues on next page)

(continued from previous page)

```

228     q_sum = q_sum + q
229     sk_sum = sk_sum + sk
230     q2_sum = q2_sum + q * q
231     sk2_sum = sk2_sum + sk * sk
232
233     ! ...
234     END DO ! end of loop over trajectories
235
236     q_ave = q_sum / REAL(nav, KIND=dp) ! average over snapshots
237     sk_ave = sk_sum / REAL(nav, KIND=dp)
238
239     q2_ave = q2_sum / REAL(nav, KIND=dp) ! average over snapshots
240     sk2_ave = sk2_sum / REAL(nav, KIND=dp)
241
242     WRITE (nrtout,*)
243     WRITE (nrtout,*)
244
245     WRITE (*, ' (A9,2x,e14.6) ') " <q>   = ", q_ave
246     WRITE (*, ' (A9,2x,e14.6) ') " error = ", SQRT ((q2_ave - q_ave * q_ave)/REAL(nav,
↪KIND=dp))
247     WRITE (*, ' (A9,2x,e14.6) ') " <s_k> = ", sk_ave
248     WRITE (*, ' (A9,2x,e14.6) ') " error = ", SQRT ((sk2_ave - sk_ave * sk_ave)/REAL(nav,
↪KIND=dp))
249
250     WRITE (nrtout, ' (A11,2x,e14.6) ') " # <q>   = ", q_ave
251     WRITE (nrtout, ' (A11,2x,e14.6) ') " # error = ", SQRT ((q2_ave - q_ave * q_ave)/
↪REAL(nav, KIND=dp))
252     WRITE (nrtout, ' (A11,2x,e14.6) ') " # <s_k> = ", sk_ave
253     WRITE (nrtout, ' (A11,2x,e14.6) ') " # error = ", SQRT ((sk2_ave - sk_ave * sk_ave)/
↪REAL(nav, KIND=dp))
254
255     ! Close the trajectory file
256     CLOSE (ntraj)
257
258     !close output file
259     CLOSE (nrtout)
260
261     DEALLOCATE (namspe, nammol, nspec)
262     DEALLOCATE (xxx, yyy, zzz)
263     DEALLOCATE (ltp)
264
265     !-----
↪-----
266     CONTAINS
267
268     SUBROUTINE compute_tetra_label (gb0, nnlab, qtetra, stetra)
269     ! *****
270     ! subroutine to compute q and sk for five particles given their global labels
271     ! (a central one and its four nearest neighbours)
272
273     ! authors: s. chiacchiera, january 2018
274     !
↪ *****
↪
275
276     IMPLICIT none
277     ! NB: I should finally recover the use of subroutine "images"

```

(continues on next page)

(continued from previous page)

```

278  INTEGER, INTENT(IN) :: gb0, nnlab (4)
279
280  REAL(KIND=dp), INTENT(OUT) :: qtetra, stetra
281  REAL(KIND=dp) :: theta, ctheta!, angle_ave, cangle_ave ! can be uncommented for_
↪ checks
282
283  REAL(KIND=dp) :: xab, yab, zab, rab, rrab, xcb, ycb, zcb, rcb, rrcb
284  REAL(KIND=dp) :: r_ave, r2_ave
285
286  INTEGER :: nn1, nn2, i,j,k !change if needed
287
288  !-----
↪ ----
289  qtetra = 0.0_dp
290  ! angle_ave = 0._dp
291  ! cangle_ave = 0._dp
292
293  j = gb0 ! central particle for angle computations
294  DO nn1 = 1, 3
295      i = nnlab (nn1)
296      DO nn2 =nn1+1, 4
297          k = nnlab (nn2)
298  !-----
↪ ----
299  ! part to compute the ijk angle (from bond_module.f90)
300      xab = xxx (i) - xxx (j)
301      yab = yyy (i) - yyy (j)
302      zab = zzz (i) - zzz (j)
303  !!!
304  ! CALL images (xab, yab, zab, dimx, dimy, dimz, srfx, srfy, srfz, shrdx, shrdy,
↪ shrdz)
305      xab = xab - dimx * ANINT (xab/dimx)
306      yab = yab - dimy * ANINT (yab/dimy)
307      zab = zab - dimz * ANINT (zab/dimz)
308  !!!
309      rab = SQRT(xab * xab + yab * yab + zab * zab)
310      rrab = MAX (rab, 1e-10_dp)
311      rrab = 1.0_dp / rrab
312      xab = xab * rrab
313      yab = yab * rrab
314      zab = zab * rrab
315
316      xcb = xxx (k) - xxx (j)
317      ycb = yyy (k) - yyy (j)
318      zcb = zzz (k) - zzz (j)
319  !!!
320      ! CALL images (xcb, ycb, zcb, dimx, dimy, dimz, srfx, srfy, srfz, shrdx,
↪ shrdy, shrdz)
321      xcb = xcb - dimx * ANINT (xcb/dimx)
322      ycb = ycb - dimy * ANINT (ycb/dimy)
323      zcb = zcb - dimz * ANINT (zcb/dimz)
324  !!!
325      rcb = SQRT(xcb * xcb + ycb * ycb + zcb * zcb)
326      rrcb = MAX (rcb, 1e-10_dp)
327      rrcb = 1.0_dp / rrcb
328      xcb = xcb * rrcb
329      ycb = ycb * rrcb

```

(continues on next page)

(continued from previous page)

```

330      zcb = zcb * rrcb
331
332      ctheta = xab * xcb + yab * ycb + zab * zcb
333      IF (ABS(ctheta)>1.0_dp) ctheta = SIGN(1.0_dp, ctheta) ! could add a check of_
↪how much >1 it is
334      theta = ACOS (ctheta)
335      !-----
↪-----
336      qtetra = qtetra + (ctheta + 1.0_dp/3.0_dp) * (ctheta + 1.0_dp/3.0_dp)
337      ! angle_ave = angle_ave + theta
338      ! cangle_ave = cangle_ave + ctheta
339      !-----
↪-----
340      !      WRITE(*, '(i2,1x,i2,1x,f13.6,1x,f13.6)') nn1, nn2, ctheta, ACOS(ctheta)/pi*180
341      END DO
342      END DO
343
344      qtetra = 1.0_dp - 0.375_dp * qtetra
345
346      ! angle_ave = angle_ave/ 6.
347      ! cangle_ave = cangle_ave/ 6.
348
349      ! print*, "average angle=", angle_ave
350      ! print*, "average cosine angle=", cangle_ave, "-> angle", ACOS(cangle_ave), " and in_
↪degrees ", ACOS(cangle_ave)/pi*180
351      !-----
↪-----
352
353      r_ave = 0.0_dp
354      r2_ave = 0.0_dp
355
356      j = gb0 ! central particle for distance computations
357      DO nn1 = 1, 4
358          i = nnlab (nn1)
359
360          xab = xxx (i) - xxx (j)
361          yab = yyy (i) - yyy (j)
362          zab = zzz (i) - zzz (j)
363      !!!
364      ! CALL images (xab, yab, zab, dimx, dimy, dimz, srfx, srfy, srfz, shrdx, shrdy,
↪shrdz)
365      xab = xab - dimx * ANINT (xab/dimx)
366      yab = yab - dimy * ANINT (yab/dimy)
367      zab = zab - dimz * ANINT (zab/dimz)
368      !!!
369      rab = SQRT(xab * xab + yab * yab + zab * zab)
370
371      r_ave = r_ave + rab
372      r2_ave = r2_ave + rab * rab
373
374      END DO
375
376      r_ave = 0.25_dp * r_ave
377      r2_ave = 0.25_dp * r2_ave
378
379      stetra = 1.0_dp - 1.0_dp/(3.0_dp*r_ave*r_ave) * (r2_ave - r_ave * r_ave)
380

```

(continues on next page)

(continued from previous page)

```

381  RETURN
382
383  END SUBROUTINE compute_tetra_label
384
385  SUBROUTINE closest4 (gb0, sorted, npart)
386  !*****
387  ! subroutine to find the four closest particles of a given species to a given particle
388  !
389  ! authors: s. chiacchiera, january 2018
390  !
391  !*****
392  !
393  ! input: - all the coordinates of beads of species "sp" (the others are set to "0")
394  !         - one selected particle of species "sp"
395  ! output: the (ordered by distance) labels of the four closest "sp" particles to it
396  IMPLICIT none
397
398  INTEGER, INTENT(IN) :: gb0, npart
399  INTEGER, INTENT(OUT) :: sorted (4)
400  INTEGER :: i, count, ncut, indx, size
401
402  REAL(KIND=dp) :: x, y, z, r, volm
403  REAL(KIND=dp) :: rcut, rmin, sorted_r (4)
404  REAL(KIND=dp), ALLOCATABLE :: list (:,:)
405
406  ncut = 15 !10 ! a bit more than 4, to be safe.
407
408  volm = dimx*dimy*dimz
409  size = MIN (npart - 1, 2 * ncut)
410  rcut = (0.75_dp/pi * ncut / npart * volm) ** (1.0_dp/3.0_dp)
411  count = 0
412
413  ALLOCATE (list (size, 2))
414
415  list = 0.0_dp
416
417  DO i = 1, nsyst
418    IF (i == gb0) CYCLE
419    IF (ltp (i) /= sp) CYCLE
420    x = xxx (i) - xxx (gb0)
421    y = yyy (i) - yyy (gb0)
422    z = zzz (i) - zzz (gb0)
423
424    !!!
425    ! CALL images (xab, yab, zab, dimx, dimy, dimz, srfx, srfy, srfz, shrdx, shrdy,
426    ! shrdz)
427    x = x - dimx * ANINT (x/dimx)
428    y = y - dimy * ANINT (y/dimy)
429    z = z - dimz * ANINT (z/dimz)
430
431    !!!
432    r = SQRT(x * x + y * y + z * z)
433
434    IF (r > rcut) CYCLE
435    count = count + 1
436    IF (count>size) THEN
437      WRITE(*,*) "error: too many particles!"
438      STOP

```

(continues on next page)

(continued from previous page)

```

435     END IF
436     list (count, 1) = REAL(i, KIND=dp) ! store the global index
437     list (count, 2) = r ! store the distance to gb0
438     END DO
439
440     ! WRITE (*,*) "rcut=", rcut ! uncomment to see radius of search region
441     ! WRITE (*,*) gb0, count ! uncomment to see the number of particles within it
442
443     IF (count < 4) THEN
444         WRITE (*,*) "error: fewer than 4 neighbours - ", count, " - found! Increase the_
↪searched volume (-> ncut)"
445         WRITE (*,*) "time=", time
446         STOP
447     END IF
448
449     ! sorting by distance
450     sorted (:) = 0
451     sorted_r (:) = 0.0_dp
452     DO j = 1, 4
453         rmin = rcut
454         indx = 0
455         DO i = 1, count
456             IF ((NINT(list (i,1)) == sorted (1)) .OR. (NINT(list (i,1)) == sorted (2)) .
↪OR. &
457                 (NINT(list (i,1)) == sorted (3)) .OR. (NINT(list (i,1)) == sorted (4)))_
↪CYCLE
458             IF (list (i,2) < rmin) THEN
459                 rmin = list (i,2)
460                 indx = NINT(list (i,1))
461             END IF
462         END DO
463         sorted (j) = indx
464         sorted_r (j) = rmin
465     END DO
466     ! WRITE (*, '(4(1x,I6))') sorted ! uncomment to see the labels of nn of gb0 (sorted_
↪by distance)
467     ! WRITE (*, '(4(1x,f13.6))') sorted_r ! uncomment to see the corresponding distances
468
469     DEALLOCATE (list) ! for fixed size, could allocate/deallocate in the main
470     RETURN
471 END SUBROUTINE closest4
472
473 END PROGRAM tetrahedral

```

Multi-GPU version of DL_MESO_DPD

Software Technical Information

The information in this section describes the DL_MESO_DPD GPU versions as a whole.

Language Fortran/CUDA-C (cuda toolkit 7.5)

Documentation Tool ReST files

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Licence BSD, v. 2.7 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Performance*
- *Examples*
- *Source Code*

Authors: Jony Castagna

This module implements the first version of the D_MESO_DPD code with multiple NVidia Graphical Processing Units (GPUs). More details about it can be found in the following sections.

Purpose of Module

In this module the main framework of a multi-GPU version of the DL_MESO_DPD code has been developed. The exchange of data between GPUs overlaps with the computation of the forces for the internal cells of each partition (a domain decomposition approach based on the MPI parallel version of DL_MESO_DPD has been followed). The current implementation is a proof of concept only and relies on slow transfers of data from the GPU to the host and vice-versa. Faster implementations will be explored in future modules.

In particular, the transfer of data occurs in 3 steps: x-y planes first, x-z planes with halo data (i.e. the values which will fill the ghost cells) from the previous swap and finally the y-z planes with all halos. This avoid the problems of the corner cells, which usually requires a separate communication reducing the number of send/receive calls from 14 to 6. The multi-GPU version has been currently tested with 8 GPUs and successfully reproduce the same results as a single GPU within machine accuracy resolution.

Future plans include benchmarking of the code with different data transfer implementations other than the current (trivial) GPU-host-GPU transfer mechanism. These are: of Peer To Peer communication within a node, CUDA-aware MPI, and CUDA-aware MPI with Direct Remote Memory Access (DRMA).

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Testing

The DL_MESO code is developed using git version control. Currently the GPU version is under a branch named `add_gpu_version`. After downloading the code, checkout the GPU branch and look into the `DPD/gpu_version` folder, i.e:

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout gpu_version
cd ./DPD/gpu_version
make all
```

To compile and run the code you need to have installed the CUDA-toolkit (≥ 8.0) and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>). For the MPI library the OpenMPI 3.1.0 has been used.

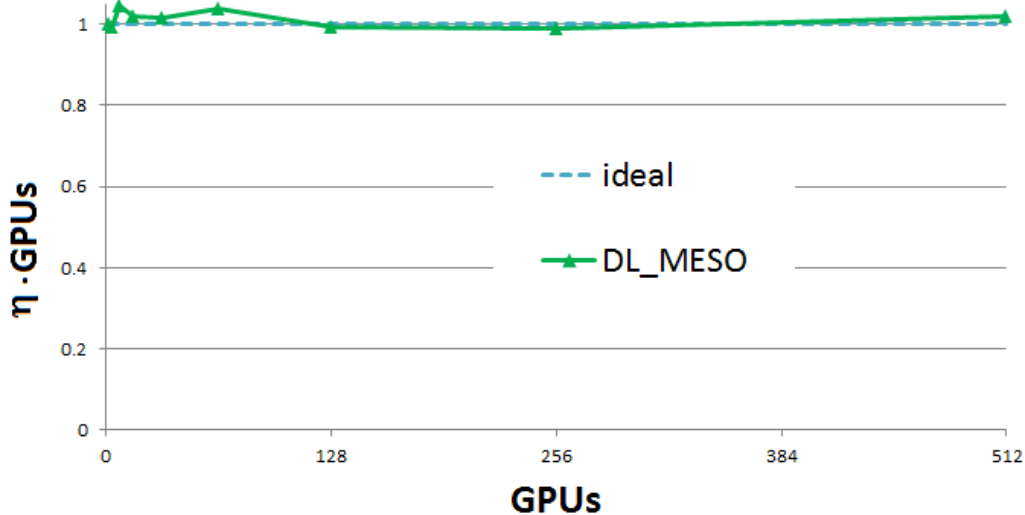
The current version has been tested ONLY for the `Mixture_Large` test case available in the `DEMO/DPD` folder. To run the case, compile the code using the `make all` command from the `bin` directory, copy the `FIELD` and `CONTROL` files in this directory and run `./dpd_gpu.exe`.

Attention: the `HISTORY` file produced is currently NOT compatible with the serial version, because this is written in the C binary data format (Fortran files are organised in records, while C are not. See <https://scipy.github.io/old-wiki/pages/Cookbook/FortranIO.html>).

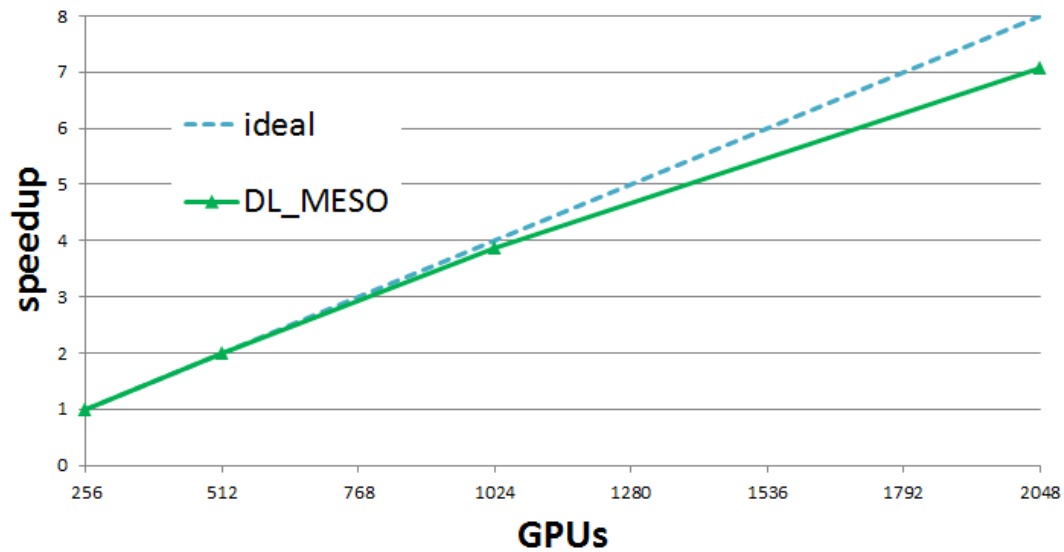
However, you can compare the `OUTPUT` and the `export` files to verify your results. For more details see the `README.rst` file in the `gpu_version` folder.

Performance

A test case a two phase mixture separation with 1.8 billion particles has been used and run for 100 time steps without IO operations. A weak scaling efficiency (η) plot up to 512 GPUs (1.2 billion particles) is presented below. This plot is obtained by taking the ratio between the wall time for the GPU count and a reference walltime of two GPUs (the singleGPU version uses a non-scalable, faster, alternative implementation which would skew the results). As can be seen, the result ($\eta * GPU_s$) oscillates near perfect scalability.



Strong scaling results are obtained using 1.8 billion particles for 256 to 2048 GPUs. Results show very good scaling, with efficiency always above 89% for 2048 GPUs (note that 2048 P100 GPUs on PizDaint is equivalent to almost 10 Petaflops of raw double precision compute performance).



Examples

See the `Mixture_Large` case in the DL_MESO manual.

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/7f3e7abe7bb1c8010dd6a5baa0de4907ffe2f003

Using SIONlib (parallel I/O library) to write/read HISTORY files in DL_MESO_DPD

Software Technical Information

Language FORTRAN 90

Licence BSD / DL_MESO Licence for the base code

Documentation Tool RST

Application Documentation See the Source Code section

Relevant Training Material See the Testing section

- *Purpose of Module*
- *Background Information*
- *Testing*

- [Source Code](#)

Purpose of Module

This module proposes to use the [SIONlib](#) library to write/read the trajectory (HISTORY) files in DL_MESO_DPD, the Dissipative Particle Dynamics (DPD) code from the [DL_MESO](#) package. In the last release (2.6, dating November 2015), the MPI version of DL_MESO_DPD generates *multiple* trajectory files, one for each MPI task. The use of [SIONlib](#) allows to minimally modify the writing so that just *one* physical file (history.sion) is produced. An analogous modification has to be implemented in the post-processing utilities that read the HISTORY files. As an example, here the modifications are implemented for one specific utility, `format_history_sion.f90`, a formatting tool analogous to `format_history.f90` (see [Formatting the HISTORY files of DL_MESO_DPD](#)). Beside showing how to adapt the reading, this allows a robust check of the implementation, since the output is human readable, contains the full trajectories, and can be readily compared with that obtained using `format_history.f90` with the standard version of DL_MESO_DPD.

Notice that the next released version of DL_MESO_DPD (in development) will tackle the writing of files differently, producing a single trajectory file from the start. However, the interface proposed here provides this feature to the users of version 2.6, and represents an alternative solution for the handling of the trajectories.

The implementation presented here is meant to show the feasibility of the interfacing, not to deal with all the possible cases. We therefore restrict in this module to the relevant case in which: i) the simulation is run in parallel using MPI, ii) a single SIONlib-type physical file is produced, and iii) the post-processing is done by a single process.

Finally, we would like to underline that, while [SIONlib](#) is optimized for a large number of MPI tasks, the reduction from several output files to just one is in any case a benefit, for example when it comes to the maintenance of the simulation output.

Background Information

The base code for this module is DL_MESO_DPD, the Dissipative Particle Dynamics code from the mesoscopic simulation package [DL_MESO](#), developed by M. Seaton at Daresbury Laboratory. This open source code is available from STFC under both academic (free) and commercial (paid) licenses. The module is to be used with [DL_MESO](#) in its last released version, version 2.6 (dating November 2015).

The present module requires the [SIONlib](#) library to be installed. Its last released version is number 1.7.1 (dating November 2016).

Testing

The version of DL_MESO_DPD including [SIONlib](#) (see below) is compiled using the corresponding makefile (Makefile-MPI). Two pre-processing flags can be used when compiling: `-D DEBUG`, to print information for any SIONlib-related action, and `-D STDTRAJ`, to recover the standard printing of trajectories as HISTORY* files.

The utility `format_history_sion.f90` is compiled with the available Fortran90+MPI compiler, and using appropriate flags for the [SIONlib](#) library, e.g:

```
mpifort -c -cpp format_history_sion.f90 ` /home/user/sionlib/bin/sionconfig --cflags --
↪f77 --mpi --threadsafe --64`
mpifort -o format_history_sion.exe format_history_sion.o ` /home/user/sionlib/bin/
↪sionconfig --libs --f77 --mpi --threadsafe --64`
```

and the executable must be in the same directory of the history.sion file. It is assumed that [SIONlib](#) has been installed in the `/home/user/sionlib/` directory, where of course the *user* name has to be adapted. If the pre-processing flag `-D`

DEBUG is used when compiling, the result of each read statement is printed to the standard output and an eventual mismatch in the number of read elements is signaled.

To test the writing/reading of the trajectories, the user can choose any simulation run using DL_MESO_DPD, then analyze the trajectories with both `format_history.f90` (which reads standard DL_MESO_DPD binary HISTORY* files) and `format_history_sion.f90` (which reads the SIONlib-type history.sion file): the formatted files so obtained, HISTORY*-F and sion*-F, respectively, should coincide.

However, for completeness, we provide the input files for a possible test: the CONTROL file

```
Simple test

vol 1000.0
temperature 1.0
cutoff 1.0

timestep 0.01
steps 1000
equilibration steps 0
traj 0 100 0
stats every 100
stack size 100
print every 100
job time 1000.0
close time 10.0

ensemble nvt mdvv

nfold 1 1 1
global bonds

finish
```

and the FIELD file

```
Simple example

SPECIES 2
A 1.0 0.0 0
B 1.0 0.0 0

MOLECULES 1
AB
nummols 1500
beads 2
A 0.0 0.0 0.0
B 0.1 0.0 0.0
bonds 1
harm 1 2 10.0 0.0
finish

INTERACTIONS 2
A A dpd 25.0 1.0 4.0
B B dpd 25.0 1.0 4.0

CLOSE
```

Source Code

A number of DL_MESO_DPD modules have to be slightly modified to use [SIONlib](#) when writing the trajectories, namely: `variables.f90`, `constants.f90`, `start_module.f90`, `dlmesodpd.f90`, `error_module.f90` and the `Makefile-MPI`. As an example of the post-processing of a SIONlib-type trajectory, we provide the formatting utility `format_history_sion.f90`, analogous to `format_history.f90` (see [Formatting the HISTORY files of DL_MESO_DPD](#)): it reads the SIONlib trajectory file (`history.sion`) and produces multiple formatted trajectory files (`sion*-F`).

In the following we give the needed changes in the form of patches¹: in the *git diff*, *a* is the branch with the standard version (version 2.6, revision 15²), *b* the SIONlib one.

The patch for `Makefile-MPI` is

```

1 diff --git a/Makefile-MPI b/Makefile-MPI
2 index 462de59..0078f94 100644
3 --- a/Makefile-MPI
4 +++ b/Makefile-MPI
5 @@ -1,10 +1,13 @@
6  MF=      Makefile
7
8  +SCFLAGS = `/home/user/sionlib/bin/sionconfig --cflags --f77 --mpi --threadsafe --64`
9  +SLFLAGS = `/home/user/sionlib/bin/sionconfig --libs --f77 --mpi --threadsafe --64`
10 +
11  FC=      mpifort
12  -FFLAGS= -O3
13  +FFLAGS= -O3 -cpp
14  LFLAGS=  $(FFLAGS)
15
16  -EXE=     dpd.exe
17  +EXE=     dpd-MPI-sion.exe
18
19  VPATH=    ../DPD/
20
21 @@ -38,12 +41,12 @@ SRC= \
22  OBJ=      $(SRC:.f90=.o)
23
24  .f90.o:
25  -      $(FC) $(FFLAGS) -c $<
26  +      $(FC) $(FFLAGS) -c $< $(SCFLAGS)
27
28  all:      $(EXE)
29
30  $(EXE):   $(OBJ)
31  -      $(FC) $(LFLAGS) -o $@ $(OBJ)
32  +      $(FC) $(LFLAGS) -o $@ $(OBJ) $(SLFLAGS)
33
34  $(OBJ):   $(MF)
35

```

The patch for `variables.f90` is

```

1 diff --git a/variables.f90 b/variables.f90
2 index 3aef25a..3f6a109 100644

```

(continues on next page)

¹ If patching is done with GNU *patch* command, the *-l* option (ignoring whitespaces) has to be active.

² On [CCPForge](#), a software development framework where, in particular, the different versions of DL_MESO_DPD are stored, version 2.6 in its revision 15 corresponds to the commit number 48e9a42a51f4cb450eb9c39dcfb6eb4a38c7cd32.

(continued from previous page)

```

3 --- a/variables.f90
4 +++ b/variables.f90
5 @@ -326,4 +326,18 @@ MODULE variables
6      !           Allocated in config_module.f90
7          REAL(KIND=dp), ALLOCATABLE, SAVE, TARGET :: commsinbuf(:,:), commsoutbuf(:,:)
8
9      +!         variables needed by SIONlib
10     +         INTEGER*8 sierr
11     +         CHARACTER(len=255) :: filename
12     +         CHARACTER(len=255) :: newfname
13     +         INTEGER:: nfiles
14     +         INTEGER:: gComm, lComm, sid
15     +         INTEGER:: fsblksize
16     +         INTEGER*8 :: chunksize
17     +         INTEGER*8 :: size, nelem
18     +         INTEGER :: seof
19     +         INTEGER :: buffer_i(6)
20     +         REAL(KIND=dp) :: buffer_r(10)
21     +         CHARACTER(LEN=8) :: buffer_c
22     +
23     END MODULE

```

The patch for constants.f90 is

```

1 diff --git a/constants.f90 b/constants.f90
2 index 65bbec4..82c2641 100644
3 --- a/constants.f90
4 +++ b/constants.f90
5 @@ -13,5 +13,7 @@ MODULE constants
6      REAL(KIND=dp), PARAMETER :: fkt=2.0_dp/3.0_dp
7      REAL(KIND=dp), PARAMETER :: rt12=3.464101615377546_dp
8      REAL(KIND=dp), PARAMETER :: langepsilon=1.0e-6_dp
9      +!         for SIONlib
10     +         INTEGER, PARAMETER :: nsion = 13
11
12     END MODULE

```

The patch for dlmesodpd.f90 is

```

1 diff --git a/dlmesodpd.f90 b/dlmesodpd.f90
2 index 062c26c..90600f2 100644
3 --- a/dlmesodpd.f90
4 +++ b/dlmesodpd.f90
5 @@ -189,8 +189,15 @@ PROGRAM dlmesodpd
6      END IF
7
8      !         close files, deallocate arrays and close down MPI
9      -
10     +!ifdef STDTRAJ
11         IF (ltraj) CLOSE (nhist)
12     +!endif
13     +!!! SIONlib 3: close SIONlib file
14     +     IF (ltraj) call fsion_parclose_mpi(sid,sierr)
15     +!ifdef DEBUG
16     +     WRITE (nprint, *) "sierr=", sierr , "on idnode=", idnode
17     +!endif
18     +!!!

```

(continues on next page)

(continued from previous page)

```

19      CALL free_memory
20      IF (.NOT. l_scr) CLOSE (nprint)
21      CALL exitcomms ()

```

The patch for `error_module.f90` is

```

1  diff --git a/error_module.f90 b/error_module.f90
2  index cb19b28..f8c3c3b 100644
3  --- a/error_module.f90
4  +++ b/error_module.f90
5  @@ -589,6 +589,11 @@ CONTAINS
6      CASE (1198)
7          WRITE (nprint,"(/,lx,'error: deallocation failure in field_module ->_
      ↪plcfor_stoyanov')")
8
9  +!      sionlib
10 +      CASE (1500)
11 +          WRITE (nprint,"(/,lx,'error: this version (using sionlib) does not support_
      ↪restart')")
12 +      CASE (1501)
13 +          WRITE (nprint,"(/,lx,'error: problem in writing SIONfile, mismatched_
      ↪number of items',i10)") value
14
15      CASE DEFAULT
16          WRITE (nprint,"(/,lx,'error: undefined error code found')")
17  @@ -605,7 +610,12 @@ CONTAINS
18      !      close all i/o channels
19
20      IF (idnode==0) CLOSE (nprint)
21  +#ifdef STDTRAJ
22      CLOSE (nhist)
23  +#endif
24  +!!! SIONlib: close file
25  +      IF (ltraj) call fsion_parclose_mpi(sid,sierr)
26  +!!!
27      IF (idnode==0) CLOSE (nsave)
28
29      !      shut down MPI and stop program

```

The patch for `start_module.f90` is

```

1  diff --git a/start_module.f90 b/start_module.f90
2  index a7f0233..81c3b24 100644
3  --- a/start_module.f90
4  +++ b/start_module.f90
5  @@ -95,6 +95,9 @@ CONTAINS
6      INTEGER :: fail (4)
7      INTEGER, ALLOCATABLE :: localmolmap(:)
8
9  +!      SIONlib 0: set sionlib filename
10 +      filename = 'history.sion'
11 +
12      !      set restart filename
13
14      WRITE (chan, '(i6.6)') idnode
15  @@ -296,6 +299,10 @@ CONTAINS
16

```

(continues on next page)

(continued from previous page)

```

17         IF (nstep>0) THEN
18
19 +!!! SIONlib 1a: give error for resart
20 +         CALL error (idnode, 1500, 1)
21 +!!!
22 +#ifdef STDTRAJ
23         IF (nodes>1) THEN
24             OPEN (nhist, file='HISTORY'//chan, access = 'sequential', form =
25 ↪ 'unformatted', status = 'unknown', &
26                 & position = 'append')
27 @@ -303,45 +310,184 @@ CONTAINS
28             OPEN (nhist, file='HISTORY', access = 'sequential', form = 'unformatted
29 ↪ ', status = 'unknown', &
30                 & position = 'append')
31         END IF
32 -
33 +#endif
34 ELSE
35
36 +!!! SIONlib 1b: define and open
37 +         gcomm = MPI_COMM_WORLD
38 +         lcomm = MPI_COMM_WORLD
39 +         fsblksize = -1
40 +         chunksize = 100
41 +         nfiles = 1
42 +         call fsion_paropen_mpi(trim(filename), 'bw', nfiles, gComm, lComm, &
43 +             chunksize, fsblksize, idnode, newfname, sid)
44 +#ifdef DEBUG
45 +         WRITE (6,*) "opened sionfile on node=", idnode, "; sid=", sid
46 +         WRITE (6,*) "input chunksize (if needed, will be internally corrected)=",
47 ↪ chunksize, "; fsblksize=", fsblksize
48 +#endif
49 +!!!
50 +#ifdef STDTRAJ
51         IF (nodes>1) THEN
52             OPEN (nhist, file='HISTORY'//chan, access = 'sequential', form =
53 ↪ 'unformatted', status = 'unknown')
54         ELSE
55             OPEN (nhist, file='HISTORY', access = 'sequential', form = 'unformatted',
56 ↪ status = 'unknown')
57         END IF
58 -
59 -
60 +#endif
61 IF (lgbnd .AND. idnode>0) THEN
62 +#ifdef STDTRAJ
63         WRITE (nhist) nspe, nmoldef, nusyst, nsyst, nbeads, 0
64 +#endif
65 +!!! SIONlib 2a: write into SION file
66 +         nelem=6
67 +         size=4
68 +         buffer_i (1:6) = (/ nspe, nmoldef, nusyst, nsyst, nbeads, 0 /)
69 +         call fsion_write(buffer_i, size, nelem, sid, sierr)
70 +#ifdef DEBUG
71 +         IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
72 +         WRITE (6,*) "a written in sionfile on node=", idnode, "; # elements", sierr
73 +#endif

```

(continues on next page)

(continued from previous page)

```

69 +!!!
70         ELSE
71 +#ifdef STDTRAJ
72         WRITE (nhist) nspe, nmoldef, nusyst, nsyst, nbeads, nbonds
73 -         END IF
74 -
75 +#endif
76 +!!! SIONlib 2b: write into SION file
77 +         nelem=6
78 +         size=4
79 +         buffer_i (1:6) = (/ nspe, nmoldef, nusyst, nsyst, nbeads, nbonds /)
80 +         call fsion_write(buffer_i,size,nelem,sid,sierr)
81 +#ifdef DEBUG
82 +         IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
83 +         WRITE (6,*) "b written in sionfile on node=",idnode ,"; # elements",sierr
84 +#endif
85 +!!!
86 +         END IF
87 +#ifdef STDTRAJ
88         WRITE (nhist) dimx, dimy, dimz, volm
89 +#endif
90 +!!! SIONlib 2c: write into SION file
91 +         nelem=4
92 +         size=8
93 +         buffer_r (1:4) = (/ dimx, dimy, dimz, volm /)
94 +         call fsion_write(buffer_r,size,nelem,sid,sierr)
95 +#ifdef DEBUG
96 +         IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
97 +         WRITE (6,*) "c written in sionfile on node=",idnode ,"; # elements",sierr
98 +#endif
99 +!!!
100 +#ifdef STDTRAJ
101         WRITE (nhist) keytrj, srftype*srfx, srftype*srfy, srftype*srfz
102 +#endif
103 +!!! SIONlib 2d: write into SION file
104 +         nelem=4
105 +         size=4
106 +         buffer_i (1:4) = (/ keytrj, srftype*srfx, srftype*srfy, srftype*srfz /)
107 +         call fsion_write(buffer_i,size,nelem,sid,sierr)
108 +#ifdef DEBUG
109 +         IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
110 +         WRITE (6,*) "d written in sionfile on node=",idnode ,"; # elements",sierr
111 +#endif
112 +!!!
113
114 !         write species information
115         DO i = 1, nspe
116             k = (i * (i + 1)) / 2
117             SELECT CASE (ktype (k))
118             CASE (0:2)
119 +#ifdef STDTRAJ
120                 WRITE (nhist) namspe (i), amass (i), vvv (2, k), lfrzn (i)
121 +#endif
122 +!!! SIONlib 2e: write into SION file
123 +         nelem=1
124 +         size=8
125 +         buffer_c = namspe (i)

```

(continues on next page)

(continued from previous page)

```

126 +          call fsion_write(buffer_c,size,nelem,sid,sierr)
127 +#ifdef DEBUG
128 +          IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
129 +          WRITE (6,*) "e1 written in sionfile on node=",idnode ,"; # elements",
    ↪sierr
130 +#endif
131 +          nelem=2
132 +          size=8
133 +          buffer_r (1:2) = (/ amass (i), vvv (2, k) /)
134 +          call fsion_write(buffer_r,size,nelem,sid,sierr)
135 +#ifdef DEBUG
136 +          IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
137 +          WRITE (6,*) "e2 written in sionfile on node=",idnode ,"; # elements",
    ↪sierr
138 +#endif
139 +          nelem=1
140 +          size=4
141 +          buffer_i (1) = lfrzn (i)
142 +          call fsion_write(buffer_i,size,nelem,sid,sierr)
143 +#ifdef DEBUG
144 +          IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
145 +          WRITE (6,*) "e3 written in sionfile on node=",idnode ,"; # elements",
    ↪sierr
146 +#endif
147 +!!!
148 +          CASE (3)
149 +#ifdef STDTRAJ
150 +          WRITE (nhist) namspe (i), amass (i), vvv (6, k), lfrzn (i)
151 +#endif
152 +!!! SIONlib 2f: write into SION file
153 +          nelem=1
154 +          size=8
155 +          buffer_c = namspe (i)
156 +          call fsion_write(buffer_c,size,nelem,sid,sierr)
157 +#ifdef DEBUG
158 +          IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
159 +          WRITE (6,*) "f1 written in sionfile on node=",idnode ,"; # elements",
    ↪sierr
160 +#endif
161 +          nelem=2
162 +          size=8
163 +          buffer_r (1:2) = (/ amass (i), vvv (6, k) /)
164 +          call fsion_write(buffer_r,size,nelem,sid,sierr)
165 +#ifdef DEBUG
166 +          IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
167 +          WRITE (6,*) "f2 written in sionfile on node=",idnode ,"; # elements",
    ↪sierr
168 +#endif
169 +          nelem=1
170 +          size=4
171 +          buffer_i (1) = lfrzn (i)
172 +          call fsion_write(buffer_i,size,nelem,sid,sierr)
173 +#ifdef DEBUG
174 +          IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
175 +          WRITE (6,*) "f3 written in sionfile on node=",idnode ,"; # elements",
    ↪sierr
176 +#endif

```

(continues on next page)

(continued from previous page)

```

177 +!!!
178         END SELECT
179         END DO
180
181     !         write molecule names
182         IF (nmoldef>0) THEN
183             DO i = 1, nmoldef
184 +#ifdef STDTRAJ
185                 WRITE (nhist) nammol (i)
186 +#endif
187 +!!! SIONlib 2g: write into SION file
188 +
189         +         nelem=1
190         +         size=8
191         +         buffer_c = nammol (i)
192         +         call fsion_write(buffer_c,size,nelem,sid,sierr)
193 +#ifdef DEBUG
194 +         IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
195 +         WRITE (6,*) "g written in sionfile on node=",idnode ,"; # elements",
196 +         ↪sierr
197 +#endif
198 +!!!
199         END DO
200         END IF
201
202     !         write name of calculation
203 +#ifdef STDTRAJ
204         WRITE (nhist) text
205 +#endif
206 +!!! SIONlib 2h: write into SION file
207 +
208         +         nelem=1
209         +         size=80
210         +         call fsion_write(text,size,nelem,sid,sierr)
211 +#ifdef DEBUG
212 +         IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
213 +         WRITE (6,*) "h written in sionfile on node=",idnode ,"; # elements",
214 +         ↪sierr
215 +#endif
216 +!!!
217
218     !         create map of local bead numbers to molecule numbers
219         ALLOCATE (localmolmap (nbeads), STAT=fail(1))
220 @@ -351,7 +497,19 @@ CONTAINS
221     !         write bead information (including molecule numbers)
222         DO i = 1, nbeads
223             imol = localmolmap(i)
224 +#ifdef STDTRAJ
225             WRITE (nhist) lab (i), ltp (i), ltm (i), imol
226 +#endif
227 +!!! SIONlib 2i: write into SION file
228 +
229         +         nelem=4
230         +         size=4
231         +         buffer_i (1:4) = (/ lab (i), ltp (i), ltm (i), imol /)
232         +         call fsion_write(buffer_i,size,nelem,sid,sierr)
233 +#ifdef DEBUG
234 +         IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
235 +         WRITE (6,*) "i written in sionfile on node=",idnode ,"; # elements",sierr
236 +#endif

```

(continues on next page)

(continued from previous page)

```

232 +!!!
233         END DO
234
235         DEALLOCATE (localmolmap, STAT=fail(1))
236 @@ -360,7 +518,19 @@ CONTAINS
237 !       write bonds between beads
238         IF (nbonds>0 .AND. ((.NOT. lgbnd) .OR. idnode==0)) THEN
239             DO j = 1, nbonds
240 +#ifdef STDTRAJ
241                 WRITE (nhist) bndtbl (j, 1), bndtbl (j, 2)
242 +#endif
243 +!!! SIONlib 2j: write into SION file
244 +       nelem=2
245 +       size=4
246 +       buffer_i (1:2) = (/ bndtbl (j, 1), bndtbl (j, 2) /)
247 +       call fsion_write(buffer_i,size,nelem,sid,sierr)
248 +#ifdef DEBUG
249 +       IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
250 +       WRITE (6,*) "j written in sionfile on node=",idnode ,"; # elements",sierr
251 +#endif
252 +!!!
253         END DO
254     END IF
255

```

These changes only affect one subroutine (start) within the start_module.f90. The user can either implement the changes shown above, or replace the second part of the subroutine start with the file provided (downloadable version of the second part of subroutine start).

The patch for statistics_module.f90 is

```

1 diff --git a/statistics_module.f90 b/statistics_module.f90
2 index e2d5e79..e283d16 100644
3 --- a/statistics_module.f90
4 +++ b/statistics_module.f90
5 @@ -11,6 +11,7 @@ MODULE statistics_module
6
7     USE constants
8     USE variables
9 +   USE error_module
10    IMPLICIT none
11
12    CONTAINS
13 @@ -621,41 +622,103 @@ CONTAINS
14     REAL(KIND=dp) :: time
15
16     !       write out data
17     -
18 +#ifdef STDTRAJ
19     WRITE (nhist) time, REAL (nbeads, KIND=dp), dimx, dimy, dimz, shrdx, shrdy,
20 ↪ shrdz
21 +#endif
22 +!!! SIONlib 2k: write into SION file
23 +   nelem=8
24 +   size=8
25 +   buffer_r (1:8) = (/ time, REAL (nbeads, KIND=dp), dimx, dimy, dimz, shrdx,
26 ↪ shrdy, shrdz /)

```

(continues on next page)

(continued from previous page)

```

25 +      call fsion_write(buffer_r,size,nelem,sid,sierr)
26 +#ifdef DEBUG
27 +      IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
28 +      WRITE (6,*) "k written in sionfile on node=",idnode ,"; # elements",sierr
29 +#endif
30 +!!!
31
32      SELECT CASE (keytrj)
33      CASE (0)
34          ! positions
35          DO i = 1, nbeads
36 +#ifdef STDTRAJ
37              WRITE (nhist) REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) + dely, zzz_
↪ (i) + delz
38 +#endif
39 +!!! SIONlib 2l: write into SION file
40 +      nelem=4
41 +      size=8
42 +      buffer_r (1:4) = (/ REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) + dely,
↪ zzz (i) + delz /)
43 +      call fsion_write(buffer_r,size,nelem,sid,sierr)
44 +#ifdef DEBUG
45 +      IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
46 +      WRITE (6,*) "l written in sionfile on node=",idnode ,"; # elements",sierr
47 +#endif
48 +!!!
49      END DO
50      CASE (1)
51          ! positions and velocities
52          DO i = 1, nbeads
53 +#ifdef STDTRAJ
54              WRITE (nhist) REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) + dely, zzz_
↪ (i) + delz, &
55                  &vxx (i), vyy (i), vzz (i)
56 +#endif
57 +!!! SIONlib 2m: write into SION file
58 +      nelem=7
59 +      size=8
60 +      buffer_r (1:7) = (/ REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) + dely,
↪ zzz (i) + delz, &
61 +          &vxx (i), vyy (i), vzz (i) /)
62 +      call fsion_write(buffer_r,size,nelem,sid,sierr)
63 +#ifdef DEBUG
64 +      IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
65 +      WRITE (6,*) "m written in sionfile on node=",idnode ,"; # elements",sierr
66 +#endif
67 +!!!
68      END DO
69      CASE (2)
70          ! positions, velocities and forces
71          IF (itype==1) THEN
72              DO i = 1, nbeads
73 +#ifdef STDTRAJ
74              WRITE (nhist) REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) + dely,
↪ zzz (i) + delz, &
75                  &vxx (i), vyy (i), vzz (i), (fxx(i)+fvx(i)), (fyy(i)+fvy(i)),
↪ (fzz(i)+fvz(i))

```

(continues on next page)

(continued from previous page)

```

76  + #endif
77  + !!! SIONlib 2n: write into SION file
78  +      nelem=10
79  +      size=8
80  +      buffer_r (1:10) = (/ REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) +
81  +      ↪ dely, zzz (i) + delz, &
82  +      ↪ &vxx (i), vyy (i), vzz (i), (fxx(i)+fvx(i)), (fyy(i)+fvy(i)),
83  +      ↪ (fzz(i)+fvz(i)) /)
84  +      call fsion_write(buffer_r,size,nelem,sid,sierr)
85  + #ifdef DEBUG
86  +      IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
87  +      WRITE (6,*) "n written in sionfile on node=",idnode ,"; # elements",sierr
88  + #endif
89  + !!!
90  +      END DO
91  +      ELSE
92  +      DO i = 1, nbeads
93  + #ifdef STDTRAJ
94  +      WRITE (nhist) REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) + dely,
95  +      ↪ zzz (i) + delz, &
96  +      ↪ &vxx (i), vyy (i), vzz (i), fxx (i), fyy (i), fzz (i)
97  + #endif
98  + !!! SIONlib 2p: write into SION file
99  +      nelem=10
100 +      size=8
101 +      buffer_r (1:10) = (/ REAL (lab(i), KIND=dp), xxx (i) + delx, yyy (i) +
102 +      ↪ dely, zzz (i) + delz, &
103 +      ↪ &vxx (i), vyy (i), vzz (i), fxx (i), fyy (i), fzz (i) /)
104 +      call fsion_write(buffer_r,size,nelem,sid,sierr)
105 + #ifdef DEBUG
106 +      IF (sierr.ne.nelem) CALL error (idnode, 1501, INT (sierr - nelem))
107 +      WRITE (6,*) "p written in sionfile on node=",idnode ,"; # elements",sierr
108 + #endif
109 + !!!
110 +      END DO
111 +      END IF
112 +      END SELECT
113 +
114 +      !      clear buffers in case of job failure
115 +      -
116 + #ifdef STDTRAJ
117 +      ENDFILE (nhist)
118 +      BACKSPACE (nhist)
119 +      -
120 + #endif
121 +      RETURN
122 +      END SUBROUTINE histout

```

Also here the changes only affect one subroutine (histout) within the statistics_module.f90. The user can either implement the changes shown above, or replace the subroutine histout with the file provided (downloadable version of the subroutine histout).

Finally, the formatting utility format_history_sion.f90 is

```

1  PROGRAM format_history_sion
2  ! *****

```

(continues on next page)

(continued from previous page)

```

3  !
4  ! module to format dl_meso HISTORY files written using SIONlib library
5  !
6  ! authors - m. a. seaton & s. chiacchiera, february 2017
7  ! adapted to use SIONlib: march 2018
8  !*****
9
10 IMPLICIT none
11 INCLUDE "mpif.h"
12
13     INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND (15, 307)
14     INTEGER, PARAMETER :: ntraj=10,nuser=5
15
16     CHARACTER(80) :: text
17     CHARACTER(8), ALLOCATABLE :: namspe (:), nammol (:)
18     CHARACTER(6) :: chan
19
20     INTEGER, ALLOCATABLE :: ltp (:), ltm (:), mole (:), beads (:), bonds (:),
↳ bndtbl (:,:)
21     INTEGER, ALLOCATABLE :: nbdmol (:), nbomol (:)
22     INTEGER :: chain, imol, ioerror, i, k, j, nmoldef, ibond
23     INTEGER :: nspe, nbeads, nusyst, nsyst, nbonds, global, species, molecule,
↳ numnodes, numbond
24     INTEGER :: nummol, lfrzn, rnmol, keytrj, srfx, srfy, srfz
25     INTEGER :: bead1, bead2
26     INTEGER :: nform
27
28     REAL(KIND=dp), ALLOCATABLE :: nmol (:)
29     REAL(KIND=dp) :: volm, dimx, dimy, dimz, shrdx, shrdy, shrdz
30     REAL(KIND=dp) :: amass, rcii
31     REAL(KIND=dp) :: time, mbeads, mglobal, x, y, z, vx, vy, vz, fx, fy, fz
32
33     LOGICAL :: eof, lcomm, lmcheck
34 !   for SIONlib
35     CHARACTER*(*) :: fname, file_mode
36     INTEGER :: numfiles, ntasks, fsblksize, sid
37     INTEGER, ALLOCATABLE :: globalranks (:)
38     INTEGER*8, ALLOCATABLE :: chunksize (:)
39     INTEGER*8 :: chunksize_input
40     INTEGER*8 :: sierr
41     INTEGER :: nformsion
42     INTEGER*8 :: size, nelem
43     INTEGER :: buffer_i(6)
44     REAL(KIND=dp) :: buffer_r(10)
45     CHARACTER(LEN=8) :: buffer_c
46     INTEGER :: rank, chunknum
47     INTEGER*8 :: posinchunk
48     INTEGER*8, ALLOCATABLE :: pos_d (:)
49     INTEGER, ALLOCATABLE :: chun_d (:)
50     INTEGER :: seof
51     PARAMETER (fname = 'history.sion')
52     PARAMETER (file_mode= 'br')
53
54     ! Switches for commenting and checking molecules
55
56     lcomm = .TRUE.
57     lmcheck = .TRUE.

```

(continues on next page)

(continued from previous page)

```

58      ! Get number of nodes
59
60
61      WRITE (*,*) "Number of nodes used in calculations ?"
62      READ (*,*) numnodes
63
64      ! Get chunksize used to write
65      WRITE (*,*) "Chunksize used to write history.sion?"
66      READ (*,*) chunksize_input
67
68      ALLOCATE (beads (numnodes), bonds (numnodes))
69
70      ! SIONlib: Determine if history.sion file exists
71      INQUIRE (file = fname, EXIST = eof)
72      IF (.NOT. eof) THEN
73          WRITE (*,*) "ERROR: cannot find history.sion file"
74          STOP
75      END IF
76
77      ! SIONlib: serial open
78      numfiles = 1
79      fsblksize = -1
80      ALLOCATE (chunksizes (numnodes), globalranks (numnodes))
81      chunksizes (:) = -1
82      globalranks (:) = -1
83      call FSION_OPEN (fname, file_mode, ntasks, numfiles, &
84          chunksizes, fsblksize, globalranks, sid)
85      IF (ntasks.ne.numnodes) THEN
86          WRITE (6,*) "Number of tasks used to write is different from given! -",
↪ ntasks
87          STOP
88      END IF
89      ! WRITE(6,*) "chunksizes=", chunksizes !not read as it should. Why?
90      WRITE(6,*) "fsblksize=", fsblksize
91      ! WRITE(6,*) "globalranks=", globalranks !not read as it should. Why?
92      WRITE(6,*) "sid=", sid
93      ! Set *by hand* the values of chunksizes and globalranks
94      DO j = 1, ntasks
95          globalranks (j) = j-1
96          chunksizes (j) = 0
97          DO WHILE (chunksizes (j) < chunksize_input)
98              chunksizes (j) = chunksizes (j) + fsblksize
99          END DO
100      END DO
101      WRITE(6,*) "(set by hand) chunksizes=", chunksizes
102      WRITE(6,*) "(set by hand) globalranks=", globalranks
103
104      ! variables to track positions within the .sion file
105      ALLOCATE (pos_d (numnodes), chun_d (numnodes))
106
107      ! Open the output files
108      nform = ntraj + numnodes
109      nformsion = nform + numnodes
110      DO j = 1, numnodes
111          IF (numnodes>1) THEN
112              WRITE (chan, '(i6.6)') j-1
113              OPEN (nformsion+j-1, file = 'sion'//chan//'-F', status = 'replace')

```

(continues on next page)

(continued from previous page)

```

114         ELSE
115             OPEN (nformsion+j-1, file = 'sion-F', status = 'replace')
116         END IF
117     END DO
118
119     ! SIONLib: reading the header of history.sion
120     ! Here the number of beads, molecules and bonds are determined
121     ! Arrays are filled with names of particles and molecules
122
123     numbond = 0
124
125     DO j = 1, numnodes
126         seof = 0
127         call fsion_feof (sid, seof)
128         IF (seof /= 0) THEN
129 #ifdef DEBUG
130             WRITE (6,*) "rank ", j-1, ": End of file !"
131 #endif
132         CYCLE
133     END IF
134
135     rank = j - 1
136     chunknum = 0
137     posinchunk = 0
138     CALL FSION_SEEK (sid, rank, chunknum, posinchunk, sierr)
139
140     ! lines a and b
141     nelem=6
142     size=4
143     buffer_i (1:6) = 0
144     CALL FSION_READ(buffer_i,size,nelem,sid,sierr)
145 #ifdef DEBUG
146     WRITE (6,*) "(a/b) in sion file, rank ", rank, ": buffer_i=",buffer_i
147     CALL READ_CHECK (sierr, nelem)
148 #endif
149     CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
150     IF (j==1) THEN
151         nspe = buffer_i (1)
152         nmoldef = buffer_i (2)
153         nusyst = buffer_i (3)
154         nsyst = buffer_i (4)
155     END IF
156     nbonds = buffer_i (5)
157     nbonds = buffer_i (6)
158     ! line c
159     nelem=4
160     size=8
161     buffer_r (1:4) = 0
162     CALL FSION_READ(buffer_r,size,nelem,sid,sierr)
163 #ifdef DEBUG
164     WRITE (6,*) "(c) in sion file, rank ", rank, ": buffer_r(1:4)=",buffer_r(1:4)
165     CALL READ_CHECK (sierr, nelem)
166 #endif
167     CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
168     IF (j==1) THEN
169         dimx = buffer_r (1)
170         dimy = buffer_r (2)

```

(continues on next page)

(continued from previous page)

```

171         dimz = buffer_r (3)
172         volm = buffer_r (4)
173     END IF
174     !      line d
175     nelem=4
176     size=4
177     buffer_i (1:4) = 0
178     CALL FSION_READ(buffer_i,size,nelem,sid,sierr)
179 #ifdef DEBUG
180     WRITE (6,*) "(d) in sion file, rank ", rank, ": buffer_i(1:4)=",buffer_i(1:4)
181     CALL READ_CHECK (sierr, nelem)
182 #endif
183     CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
184     IF (j==1) THEN
185         keytrj = buffer_i (1)
186         srfx = buffer_i (2)
187         srfy = buffer_i (3)
188         srfz = buffer_i (4)
189     END IF
190     beads (j) = nbeads
191     bonds (j) = nbonds
192     numbond = numbond + nbonds
193
194     IF (lcomm) WRITE (nformsion+j-1,*) "# nspe, nmoldef, nusyst, nsyst, nbeads, ↵
↵nbonds"
195     WRITE (nformsion+j-1,*) nspe, nmoldef, nusyst, nsyst, nbeads, nbonds
196     IF (lcomm) WRITE (nformsion+j-1,*) "# dimx, dimy, dimz, volm"
197     WRITE (nformsion+j-1,97) dimx, dimy, dimz, volm
198     IF (lcomm) WRITE (nformsion+j-1,*) "# keytrj, srfx, srfy, srfz"
199     WRITE (nformsion+j-1,*) keytrj, srfx, srfy, srfz
200
201     chun_d (j) = chunknum
202     pos_d (j) = posinchunk
203     END DO ! end of loop over nodes
204     !!!
205     ALLOCATE (namspe (nspe), nammol (nmoldef))
206     IF (lmcheck) THEN
207         ALLOCATE (ltp (1:nsyst), ltm (1:nsyst), mole (1:nsyst))
208         ALLOCATE (nmol (1:nmoldef), nbdmol (1:nmoldef), nbomol (1:nmoldef))
209         ALLOCATE (bndtbl (numbond, 2))
210     ENDIF
211
212     DO j = 1, numnodes
213         rank = j - 1
214         chunknum = chun_d (j)
215         posinchunk = pos_d (j)
216         CALL FSION_SEEK (sid, rank, chunknum, posinchunk, sierr)
217     !!!
218     IF (lcomm) WRITE (nformsion+j-1,*) "# SPECIES:"
219     IF (lcomm) WRITE (nformsion+j-1,*) "# namspe, amass, rcii, lfrzn"
220
221     DO i = 1, nspe
222         !      line e
223         nelem = 1
224         size = 8
225         buffer_c = ' '
226         CALL FSION_READ(buffer_c,size,nelem,sid,sierr)

```

(continues on next page)

(continued from previous page)

```

227 #ifdef DEBUG
228     WRITE (6,*) "(e1) in sion file, rank ", rank, ": buffer_c=",buffer_c
229     CALL READ_CHECK (sierr, nelelem)
230 #endif
231     CALL DETERMINE_POS (rank, nelelem*size, chunknum, posinchunk)
232     IF (j==1) THEN
233         namspe (i) = buffer_c
234     END IF
235
236     nelelem=2
237     size=8
238     buffer_r (1:2) = 0
239     CALL FSION_READ(buffer_r,size,nelelem,sid,sierr)
240 #ifdef DEBUG
241     WRITE (6,*) "(e2) in sion file, rank ", rank, ": buffer_r (1:2)=",buffer_
242     ↪ r (1:2)
243     CALL READ_CHECK (sierr, nelelem)
244 #endif
245     CALL DETERMINE_POS (rank, nelelem*size, chunknum, posinchunk)
246     amass = buffer_r (1)
247     rcii = buffer_r (2)
248
249     nelelem=1
250     size=4
251     buffer_i (1) = 0
252     CALL FSION_READ(buffer_i,size,nelelem,sid,sierr)
253 #ifdef DEBUG
254     WRITE (6,*) "(e3) in sion file, rank ", rank, ": buffer_i (1)=",buffer_i_
255     ↪ (1)
256     CALL READ_CHECK (sierr, nelelem)
257 #endif
258     CALL DETERMINE_POS (rank, nelelem*size, chunknum, posinchunk)
259     lfrzn = buffer_i (1)
260     WRITE (nformsion+j-1,96) namspe (i), amass, rcii, lfrzn
261     END DO
262
263     IF (nmoldef>0) THEN
264         IF (lcomm) WRITE (nformsion+j-1,*) "# MOLECULES:"
265         IF (lcomm) WRITE (nformsion+j-1,*) "# nammol"
266
267         DO i = 1, nmoldef
268             ! line g
269             nelelem=1
270             size=8
271             buffer_c = ' '
272             CALL FSION_READ(buffer_c,size,nelelem,sid,sierr)
273             IF (j==1) THEN
274                 nammol (i) = buffer_c
275             END IF
276 #ifdef DEBUG
277             WRITE (6,*) "(g) in sion file, rank ", rank, ": buffer_c=",buffer_c
278             CALL READ_CHECK (sierr, nelelem)
279 #endif
280             CALL DETERMINE_POS (rank, nelelem*size, chunknum, posinchunk)
281             WRITE (nformsion+j-1,*) nammol (i)
282         END DO
283     END IF

```

(continues on next page)

(continued from previous page)

```

282      !      line h
283      nelem=1
284      size=80
285      text = '
286      '
287      CALL FSION_READ(text,size,nelem,sid,sierr)
288      #ifdef DEBUG
289      WRITE (6,*) "(h) in sion file, rank ", rank, ": text=", text
290      CALL READ_CHECK (sierr, nelem)
291      #endif
292      CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
293
294      IF (lcomm) WRITE (nformsion+j-1,*) "# Simulation name:"
295      WRITE (nformsion+j-1,*) text
296
297      IF (j==1) THEN
298          nummol = 0 !counter for number of molecules
299          ibond = 0 !counter for bonds
300      END IF
301
302      ! Here one could close and open again the loop over nodes, as in the std
303      ! version of the utility: in case, pos_d and chunk_d must be updated too
304
305      !      fill in arrays for beads and bonds
306      rank = j - 1
307
308      IF (lcomm) WRITE (nformsion+j-1,*) "# BEADS:"
309      IF (lcomm) WRITE (nformsion+j-1,*) "# global, species, molecule, chain"
310
311      IF (lmcheck) THEN
312          !Build ltp, ltm, mole
313          DO i = 1, beads (j)
314              !      line i
315              nelem = 4
316              size=4
317              buffer_i (1:4) = 0
318              CALL FSION_READ(buffer_i,size,nelem,sid,sierr)
319              #ifdef DEBUG
320              WRITE (6,*) "(i) in sion file, rank ", rank, ": buffer_i(1:4)=",buffer_
321              i(1:4)
322              CALL READ_CHECK (sierr, nelem)
323              #endif
324              CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
325
326              global = buffer_i (1)
327              species = buffer_i (2)
328              molecule = buffer_i (3)
329              chain = buffer_i (4)
330
331              ltp (global) = species
332              ltm (global) = molecule
333              mole (global) = chain
334              nummol = MAX (nummol, chain)
335              WRITE (nformsion+j-1,*) global, species, molecule, chain
336          END DO
337      ELSE

```

(continues on next page)

(continued from previous page)

```

337     DO i = 1, beads (j)
338         !      line i
339         nelem = 4
340         size=4
341         buffer_i (1:4) = 0
342         CALL FSION_READ(buffer_i,size,nelem,sid,sierr)
343 #ifdef DEBUG
344         WRITE (6,*) "(i) in sion file, rank ", rank, ": buffer_i(1:4)=",buffer_
345         ↪ i(1:4)
346         CALL READ_CHECK (sierr, nelem)
347 #endif
348         CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
349         global = buffer_i (1)
350         species = buffer_i (2)
351         molecule = buffer_i (3)
352         chain = buffer_i (4)
353         WRITE (nformsion+j-1,*) global, species, molecule, chain
354     END DO
355 ENDIF
356
357 IF (bonds (j)>0) THEN
358     IF (lcomm) WRITE (nformsion+j-1,*) "# BONDS:"
359     IF (lcomm) WRITE (nformsion+j-1,*) "# extremes of the bond"
360
361     IF (lmcheck) THEN
362         ! Build bndtbl
363         DO i = 1, bonds (j)
364             ibond = ibond + 1
365             !      line j
366             nelem=2
367             size=4
368             buffer_i (1:2) = 0
369             CALL FSION_READ(buffer_i,size,nelem,sid,sierr)
370 #ifdef DEBUG
371             WRITE (6,*) "(j) in sion file, rank ", rank, ": buffer_i(1:2)=",
372             ↪ buffer_i(1:2)
373             CALL READ_CHECK (sierr, nelem)
374 #endif
375             CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
376             bead1 = buffer_i (1)
377             bead2 = buffer_i (2)
378             bndtbl (ibond, 1) = bead1
379             bndtbl (ibond, 2) = bead2
380             WRITE (nformsion+j-1,*) bead1, bead2
381         END DO
382     ELSE
383         DO i = 1, bonds (j)
384             !      line j
385             nelem=2
386             size=4
387             buffer_i (1:2) = 0
388             CALL FSION_READ(buffer_i,size,nelem,sid,sierr)
389 #ifdef DEBUG
390             WRITE (6,*) "(j) in sion file, rank ", rank, ": buffer_i(1:2)=",
391             ↪ buffer_i(1:2)
392             CALL READ_CHECK (sierr, nelem)
393 #endif

```

(continues on next page)

(continued from previous page)

```

391         CALL DETERMINE_POS (rank, nelem*size, chunknum, posinchunk)
392         bead1 = buffer_i (1)
393         bead2 = buffer_i (2)
394         WRITE (nformsion+j-1,*) bead1, bead2
395     END DO
396 END IF
397 END IF
398 chun_d (j) = chunknum
399 pos_d (j) = posinchunk
400
401 END DO ! over nodes
402
403 IF (lmcheck) THEN
404     ! determine numbers of molecules, beads and bonds per molecule type
405     nmol = 0.0_dp
406     nbdmol = 0
407     nbomol = 0
408     chain = 0
409     imol = 0 !necessary to avoid out of bounds
410
411     DO i = 1, nsyst
412         IF (mole (i) /= chain) THEN
413             chain = mole (i)
414             imol = ltm (i)
415             nmol (imol) = nmol (imol) + 1.0_dp
416         END IF
417         IF (imol > 0) nbdmol (imol) = nbdmol (imol) + 1
418     END DO
419
420     DO i = 1, numbond
421         imol = ltm (bndtbl (i,1))
422         nbomol (imol) = nbomol (imol) + 1
423     END DO
424
425     DO i = 1, nmoldef
426         rnmol = NINT (nmol (i))
427         IF (rnmol>0) THEN
428             nbdmol (i) = nbdmol (i) / rnmol
429             nbomol (i) = nbomol (i) / rnmol
430         END IF
431     END DO
432
433     ! Write to std output the arrays built
434     WRITE (*,*) "# Check of beads: i, ltp(i), ltm(i), mole(i)"
435     DO i = 1, nsyst
436         WRITE(*,*) i, ltp (i), ltm (i), mole (i)
437     END DO
438
439     !Check of molecule beads and numbers
440     IF (nmoldef>0) THEN
441         WRITE (*,*) "# Check of molecules: nammol(i), nbdmol(i), nbomol(i),
↪nmol(i) "
442         DO i = 1, nmoldef
443             WRITE (*,*) nammol (i), nbdmol (i), nbomol (i), NINT(nmol(i))
444         END DO
445         WRITE (*,*) "# Total number of molecules = ",nummol
446     END IF

```

(continues on next page)

(continued from previous page)

```

447      ! Write to std output bndtbl
448      IF (numbond > 0) THEN
449          WRITE (*,*) "# Check of bonds: bndtbl(i,1), bndtbl(i,2)"
450          DO i = 1, numbond
451              WRITE (*,*) bndtbl (i,1), bndtbl (i,2)
452          END DO
453      END IF
454  END IF
455  END IF
456
457  !reading trajectories
458  DO j = 1, numnodes
459
460      seof = 0
461      k = 0
462
463      IF (lcomm) WRITE (nformsion+j-1,*) "# --- TRAJECTORIES --- (key =", keytrj,")
464      ↪ "
465      SELECT CASE (keytrj)
466      CASE (0)
467          IF (lcomm) WRITE (nformsion+j-1,*) "# mglobal, x, y, z"
468      CASE(1)
469          IF (lcomm) WRITE (nformsion+j-1,*) "# mglobal, x, y, z, vx, vy, vz"
470      CASE(2)
471          IF (lcomm) WRITE (nformsion+j-1,*) "# mglobal, x, y, z, vx, vy, vz, fx, ↪
472      ↪ fy, fz"
473      END SELECT
474
475      rank = j - 1
476      chunknum = chun_d (j)
477      posinchunk = pos_d (j)
478      CALL FSION_SEEK (sid, rank, chunknum, posinchunk, sierr)
479      !!!
480      DO WHILE (.true.)
481          call fsion_feof (sid, seof)
482          IF (seof /= 0) THEN
483              #ifdef DEBUG
484                  WRITE (6,*) "rank ", rank, ": End of file !"
485              #endif
486              IF (k==0) THEN
487                  PRINT *, 'ERROR: cannot find trajectory data in history.sion file'
488                  STOP
489              END IF
490              EXIT
491          END IF
492
493          ! line k
494          nelem=8
495          size=8
496          buffer_r (1:8) = 0
497          CALL FSION_READ(buffer_r,size,nelem,sid,sierr)
498          #ifdef DEBUG
499              WRITE (6,*) "(k) in sion file, rank ", rank, ": buffer_r(1:8)=",buffer_
500              ↪ r(1:8)
501              CALL READ_CHECK (sierr, nelem)
502          #endif

```

(continues on next page)

(continued from previous page)

```

501     time = buffer_r (1)
502     mbeads = buffer_r (2)
503     dimx = buffer_r (3)
504     dimy = buffer_r (4)
505     dimz = buffer_r (5)
506     shrdx = buffer_r (6)
507     shrdy = buffer_r (7)
508     shrdz = buffer_r (8)
509     !
510     IF (lcomm) WRITE (nformsion+j-1,*) "# time, mbeads, dimx, dimy, dimz,
↪shrdx, shrdy, shrdz"
511     WRITE (nformsion+j-1,98) time, mbeads, dimx, dimy, dimz, shrdx, shrdy,
↪shrdz
512
513     k = k + 1
514
515     IF (lcomm) WRITE (nformsion+j-1,*) "# snapshot number:", k
516
517     nbeads = NINT (mbeads)
518
519     SELECT CASE (keytrj)
520     CASE (0)
521         DO i = 1, nbeads
522             ! line 1
523             nelem=4
524             size=8
525             buffer_r (1:4) = 0
526             CALL FSION_READ(buffer_r,size,nelem,sid,sierr)
527 #ifdef DEBUG
528             WRITE (6,*) "(1) in sion file, rank ", rank, ": buffer_r(1:4)=",
↪buffer_r(1:4)
529             CALL READ_CHECK (sierr, nelem)
530 #endif
531             mglobal = buffer_r (1)
532             x = buffer_r (2)
533             y = buffer_r (3)
534             z = buffer_r (4)
535             !
536             WRITE (nformsion+j-1,99) mglobal, x, y, z
537         END DO
538     CASE (1)
539         DO i = 1, nbeads
540             ! line m
541             nelem=7
542             size=8
543             buffer_r (1:7) = 0
544             CALL FSION_READ(buffer_r,size,nelem,sid,sierr)
545 #ifdef DEBUG
546             WRITE (6,*) "(m) in sion file, rank ", rank, ": buffer_r(1:7)=",
↪buffer_r(1:7)
547             CALL READ_CHECK (sierr, nelem)
548 #endif
549             mglobal = buffer_r (1)
550             x = buffer_r (2)
551             y = buffer_r (3)
552             z = buffer_r (4)
553             vx = buffer_r (5)

```

(continues on next page)

(continued from previous page)

```

554         vy = buffer_r (6)
555         vz = buffer_r (7)
556         !
557         WRITE (nformsion+j-1,99) mglobal, x, y, z, vx, vy, vz
558     END DO
559     CASE (2)
560     DO i = 1, nbeads
561         ! line n and p
562         nelem=10
563         size=8
564         buffer_r (1:10) = 0
565         CALL FSIION_READ(buffer_r,size,nelem,sid,sierr)
566 #ifdef DEBUG
567         WRITE (6,*) "(1) in sion file, rank ", rank, ": buffer_r(1:10)=",
568         ↪buffer_r(1:10)
569         CALL READ_CHECK (sierr, nelem)
570 #endif
571         mglobal = buffer_r (1)
572         x = buffer_r (2)
573         y = buffer_r (3)
574         z = buffer_r (4)
575         vx = buffer_r (5)
576         vy = buffer_r (6)
577         vz = buffer_r (7)
578         fx = buffer_r (8)
579         fy = buffer_r (9)
580         fz = buffer_r (10)
581         !
582         WRITE (nformsion+j-1,99) mglobal, x, y, z, vx, vy, vz, fx, fy, fz
583     END DO
584     END SELECT
585 END DO
586
587 ! close the output files
588 DO j = 1, numnodes
589     CLOSE (nformsion+j-1)
590 END DO
591
592 ! SIONlib serial close
593 call FSIION_CLOSE (sid, sierr)
594 CLOSE (nformsion)
595
596 #ifdef DEBUG
597 WRITE (*,*) "The final chunknumbers and positions within chunks are:"
598 WRITE (*,*) "chun_d=", chun_d
599 WRITE (*,*) "pos_d=", pos_d
600 #endif
601
602 DEALLOCATE (beads, bonds)
603 DEALLOCATE (namspe, nammol)
604 IF (lmcheck) DEALLOCATE (ltp, ltm, mole, nmol, nbdmol, bndtbl, nbomol)
605 ! SIONlib related quantities
606 DEALLOCATE (chunksizes, globalranks, pos_d, chun_d)
607
608 99    FORMAT(f10.1,2x,1p,9(e13.6,3x))

```

(continues on next page)

(continued from previous page)

```

610  98      FORMAT(8(f10.3,3x))
611  97      FORMAT(4(f10.3,3x))
612  96      FORMAT(A9,3x,2(f10.3,3x),I2)
613
614      CONTAINS
615
616      SUBROUTINE DETERMINE_POS (rank, nbytes, chunk, pos)
617      !*****
618      ! routine to determine the position in the .sion file at each write statement
619      !
620      ! author - s. chiacchiera, march 2018
621      !*****
622      ! It is assumed that each rank has its chunks numbered as 0, 1, etc
623      IMPLICIT none
624      INTEGER, INTENT (OUT) :: rank
625      INTEGER, INTENT (INOUT) :: chunk
626      INTEGER*8, INTENT (INOUT) :: pos
627      INTEGER*8, INTENT (IN) :: nbytes
628      INTEGER*8 :: pos_try
629      IF (nbytes > chunksizes (rank + 1)) THEN
630          WRITE (*,*) "error: too large record (hint: increase chunksize)"
631          STOP
632      END IF
633      pos_try = pos + nbytes
634      IF (pos_try <= chunksizes (rank + 1)) THEN
635          pos = pos_try
636      ELSE
637          chunk = chunk + 1
638          pos = nbytes
639      END IF
640      RETURN
641      END SUBROUTINE DETERMINE_POS
642
643      SUBROUTINE READ_CHECK (sierr, nelem)
644      !*****
645      ! routine to signal eventual mismatch when reading the .sion file
646      !
647      ! author - s. chiacchiera, march 2018
648      !*****
649
650      IMPLICIT none
651      INTEGER*8, INTENT (IN) :: sierr, nelem
652      IF (sierr.ne.nelem) THEN
653          WRITE (6,*) "error: number of elements read differs from expected!_
654      ↪ mismatch is ", sierr-nelem
655          STOP
656      END IF
657      END SUBROUTINE READ_CHECK
658
659      END PROGRAM format_history_sion

```

Additional information

Using the modifications proposed above, the trajectories will be written by DL_MESO_DPD in the SIONlib format (history.sion file). For comparison purposes, the standard HISTORY* files can be also written at the same time, using the `-D STDTRAJ` flag for compilation in Makefile-MPI.

To be able to use SIONlib, the writing statements for which the records are formed by inhomogeneous items (e.g., two

8-byte strings and a 4-byte integer) have to be split into different records, hence the increased number of write/read statements. To help the reader, comments have been added to label all the SIONlib-related commands, namely: “SIONlib 0, 1a, 1b, 2a, 2b, ..., 2p, 3”. The writing statements are labelled “2a, 2b, etc”, and each one corresponds to the writing of single record in the standard version of DL_MESO_DPD. The SIONlib file definition, opening and closing statements have been labelled 0, 1 and 3 in the comments.

Important SIONlib variables:

- `fsblksize`: file system block size in bytes. If set to -1, it is read by SIONlib. (Typically, this value is 4096.)
- `chunksize`: size in bytes of the data written by a task in a single write call. It is internally increased by SIONlib to the next multiple of the filesystem block size. (For DL_MESO_DPD, the largest record has size 80 bytes, hence we choose `chunksize` = 100, which, typically, will be internally increased to 4096.)
- `nfiles`: number of physical files produced by SIONlib (set to 1 here).

Acknowledgements

We are very grateful to Dr. Wolfgang Frings for kind support concerning the usage of the Fortran version of [SIONlib](#).

Software Technical Information

Name DL_MESO

Language Fortran/CUDA-C

Licence BSD, v. 2.7 or later

Documentation Tool Fortran/C comments

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

Surface boundary conditions on DL_MESO_DPD multi-GPUs

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module implement the solid surfaces boundary conditions on the multi-GPU version of DL_MESO_DPD.

Purpose of Module

The single GPU version contains already the wall surface boundary conditions. The following module is an implementation on the multi GPU version.

Real cases often involve complex geometries and require the implementation of solid walls as boundary conditions. A typical example is the flow in microchannels used for example in the production of Graphene. The interaction between fluid and surface create a different and unique profile of velocities which has a strong impact on the fluid dynamic,

especially in case of non-Newtonian fluids (i.e. where the shear stress is a non linear function of the velocity gradient) like shampoo and other body care products.

This module will allow to study such phenomena reducing the computational cost and time and scaling up to larger systems.

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

To compile and run the code you need to have installed the CUDA-toolkit (≥ 8.0) and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>). For the MPI library the OpenMPI 3.1.0 has been used.

The DL_MESO code is developed using git version control. Currently, the multi GPU version is under a branch named `multi_GPU_version`. After downloading the code, checkout the GPU branch and move to the `DPD/gpu_version/bin` folder. Modify the Makefile to use the correct GPU architecture (`sm_XX`) and check if the CPP flags are supported (i.e.: `-DAWARE_MPI` for `CUDA_aware_MPI` support, `-DOPENMPI` for OpenMPI library, `-DMVAPICH` for `MVAPICH` library and `-DHWLOC` for `hwloc` support). Make sure `nvcc` is installed (or CUDA toolkit module loaded). Then, compile using `make all`. In short:

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout multi_GPU_version
cd ./DPD/gpu_version/bin
# Modify the Makefile according to your device and libraries
make all
```

To run the test case, copy the `FIELD` and `CONTROL` files from the “`./tests/Poiseuille`” directory and run using `mpirun -np 8 ./dpd_gpu.exe` on a job partition with 1 GPU available per MPI process. The test case consists in simulating the Poiseuille flow, using 8 GPUs, obtained between two parallel plane surfaces. Being the flow laminar, the solution has to match with the analytic parabolic profile of the velocity field. Compare the `OUTPUT` and the export files to verify your results. Do not worry about the problem with `total_nbeads` warning message.

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/c10992eb39815029bbe485f18e05230cd098afab

Software Technical Information

Name DL_MESO

Language Fortran/CUDA-C

Licence BSD, v. 2.7 or later

Documentation Tool Fortran/C comments

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

Load balancing for multi-GPU DL_MESO

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module concerns the implementation of the ALL library for load balancing in the multi-GPU version of DL_MESO_DPD.

Purpose of Module

The intention is to allow for better performance when modelling complex systems, like large proteins or lipid bilayers, redistributing the work load across the GPUs. The A Load Balancing (ALL) library, developed at Julich Supercomputer Center, provides several scheme to find the ideal split of the work load: from the simplest orthogonal non staggered domain decomposition, to the more fancy Voronoi mesh scheme.

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

The ALL library has been introduced during the ECAM Extended Software Development Workshop for Meso and Multi Scale Modelling hosted in Julich in June 2019.

Building and Testing

The DL_MESO code is developed using git version control. Currently, the multi GPU version is under a branch named `multi_GPU_version`. After downloading the code, checkout the GPU branch and look into the `DPD/gpu_version` folder, i.e:

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout multi_GPU_version
cd ../DPD/gpu_version/bin
make all
```

To compile and run the code you need to have installed the CUDA-toolkit (≥ 8.0) and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>). For the MPI library, OpenMPI 3.1.0 has been used in testing.

To run the case, copy the FIELD and CONTROL files from the “./tests” directory and run using `mpirun -np NP ./dpd_gpu.exe`. Compare the OUTPUT and the export files to verify your results.

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/7f3e7abe7bb1c8010dd6a5baa0de4907ffe2f003

The ALL library is available (after registration) at:

- <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing>

Software Technical Information

Name DL_MESO

Language Fortran/CUDA-C

Licence BSD, v. 2.7 or later

Documentation Tool Fortran/C comments

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

Many body DPD on DL_MESO_DPD single-GPU

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module implement the many body DPD boundary conditions on the single-GPU version of DL_MESO_DPD.

Purpose of Module

One of the main weak point of DPD simulation is in its equation of state. For example, compressible gas or vapor liquid mixtures are difficult, if not impossible, to correctly be simulated. The many-body DPD approach allows us to overcome this limit by extending the potential of neighbour beads to a large cut off radius.

This module consists in the implementation of many-body DPD on the single GPU version of DL_MESO_DPD. The new feature will allow the simulation of complex systems such as liquid drop, phase interactions, etc.

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

To compile and run the code you need to have installed the CUDA-toolkit (≥ 8.0) and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>). For the MPI library, OpenMPI 3.1.0 has been used during testing.

The DL_MESO code is developed using git version control. Currently, the single GPU version is under a branch named `single_GPU_version`. After downloading the code, checkout the GPU branch and move to the `DPD/gpu_version/bin` folder. Modify the Makefile to use the correct GPU architecture (`sm_XX`) and check if the CPP flags are supported (i.e.: `-DAWARE_MPI` for `CUDA_aware_MPI` support, `-DOPENMPI` for OpenMPI library, `-DMVAPICH` for `MVAPICH` library and `-DHWLOC` for `hwloc` support). Make sure `nvcc` is installed (or CUDA toolkit module loaded). Then, compile using `make all`. In short,

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout single_GPU_version
cd ./DPD/gpu_version/bin
# Modify Makefile according to your device and libraries
make all
```

There are two test cases for this module: * a water drop between two surfaces * a vapour-liquid interface.

To run the water drop case, copy the `FIELD` and `CONTROL` files from the “`./tests/SurfaceDrop`” directory and run using `./dpd_gpu.exe`. The test case consists of simulating the evolution of fully-dispersed water droplets which will eventual agglomerate and deposit on a surface.

To run the vapour-liquid interface, copy the `FIELD` and `CONTROL` files from the “`./tests/VapourLiquid`” directory and run using `./dpd_gpu.exe`. The test case consists of simulating the interface between vapour and liquid starting from water particle dispersed into the vapour.

For both test cases, compare the `OUTPUT` and the `export` files to verify your results.

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/09c62778b1d5e6928b74570191f037a54c80fab

Software Technical Information

Name DL_MESO

Language Fortran/CUDA-C

Licence BSD, v. 2.7 or later

Documentation Tool Fortran/C comments

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

Long Integer on DL_MESO_DPD multi-GPU

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module extends the capability of the DL_MESO code to run simulations with more than 1.8 billion particles using long integer arrays.

Purpose of Module

The current version of DL_MESO_DPD on multi GPU cannot run systems larger than 1.8 billion particles due to the INTEGER type used in Fortran for the particle arrays.

This module addresses this problem replacing, only where needed, the INTEGER with LONG INTEGER type in the Fortran arrays. This allows us to run simulations with more than 1.8 billion particles and, as a result, more complex systems.

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

The DL_MESO code is developed using git version control. Currently, the multi GPU version is under a branch named `multi_GPU_version`. After downloading the code, checkout the GPU branch and look into the `DPD/gpu_version` folder, i.e:

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout multi_GPU_version
cd ../DPD/gpu_version/bin
make all
```

To compile and run the code you need to have installed the CUDA-toolkit (≥ 8.0) and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>). For testing, the MPI library the OpenMPI 3.1.0 has been used.

To run the separation test case, copy the `FIELD` and `CONTROL` files from the “`../tests/LargeSeparation`” directory and run using `mpirun -np NP ./dpd_gpu.exe`. The test case consists in simulating a binary mixture of 24 billion particles on 4096 GPUs (tested on PizDaint CSCS supercomputer).

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/7f3e7abe7bb1c8010dd6a5baa0de4907ffe2f003

Software Technical Information

Name DL_MESO

Language Fortran/CUDA-C

Licence BSD, v. 2.7 or later

Documentation Tool Fortran/C comments

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

Many body DPD on DL_MESO_DPD multi-GPU

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

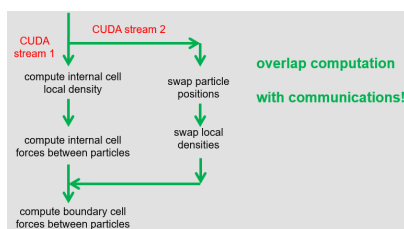
This module implements the many body DPD algorithm on the multi-GPU version of DL_MESO_DPD.

Purpose of Module

One of the main weak points of the DPD simulation is in its equation of state. For example, compressible gas or vapor liquid mixtures are difficult, if not impossible, to be simulated correctly. The many-body DPD approach allows us to overcome this limitation by extending the potential of neighbour beads to a large cut off radius.

This module consists in the implementation of many-body DPD on the multi GPU version of DL_MESO_DPD. The new feature will allow us to simulate complex systems liquid drops, phase interactions, etc.

From an implementation point of view, the algorithm requires us to first loop over the internal cells of a typical domain to calculate the local densities, followed by a second loop to find the forces acting between particles. To achieve good scaling across multiple GPUs, we must allow the overlap of the computation of local densities and forces with the swapping of the particle's positions and local densities. This is achieved with a partial sum of the forces based on the internal particles first and then adding the forces from the border particles later. A flowchart of the algorithm is presented in the figure below.



Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

The DL_MESO code is developed using git version control. Currently, the multi GPU version is under a branch named `multi_GPU_version`. After downloading the code, checkout the GPU branch and look into the `DPD/gpu_version` folder, i.e:

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout multi_GPU_version
cd ./DPD/gpu_version/bin
```

To compile and run the code you need to have installed the CUDA-toolkit (≥ 8.0) and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>). For the MPI library, OpenMPI 3.1.0 has been used. Install `hwloc` if you want to set the GPU affinity between devices and CPU cores, otherwise remove the `-DHWLOC` flag in the Makefile.

Finally, you need to install the ALL library and make sure the ALL path is set correctly. ALL handles the load balancing among the GPUs, for details see [Load balancing for multi-GPU DL_MESO](#).

Use `make all` to compile, resulting in the executable `dpd_gpu.exe`.

A testloop is added in the `tests` folder. Type `./Tesloop_All` followed by option 2 to and specify 8 as number of GPUs. Verify the results with option 3. No difference should appear in the statistical values and final stress values (the final printed positions are randomly particles chosen and can be different at every run).

For the current module, the `test/SurfaceDrop` test case is a good example of combining manybody DPD and load balanced as presented in [Load balancing for multi-GPU DL_MESO](#). Below is an snapshot from the simulation based on the same input (but large system) using 8 GPUs and for 35k time steps.

Source Code

This module has been merged into `DL_MESO` code. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/-/commit/5d5db87433f21e31afcb61343f500728af52cd0a

while for the GPU version we have:

First version of DL_MESO_DPD code for NVidia GPU

Software Technical Information

The information in this section describes the `DL_MESO_DPD GPU` versions as a whole.

Language Fortran/CUDA-C (cuda toolkit 7.5)

Documentation Tool ReST files

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Licence BSD, v. 2.7 or later

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Performance*
- *Examples*
- *Source Code*

This module implements the first version of the `DL_MESO_DPD` code on NVidia Graphical Processing Unit (GPU). More details about it can be found in the following sections.

Purpose of Module

In order to accelerate the `DL_MESO_DPD` code on the latest and future exascale hardware, a first version for NVidia GPU has been developed. This is only a starting point, it does NOT cover all the possible cases and it does NOT yet support multiple GPUs. However, it represent an HPC milestone for the application, complementing the already present parallel versions developed for shared and distributed memory (MPI/OpenMP).

In this version, the full computational workload is offloaded to the GPUs with the “H_MAINLOOP” call present in the “dlmesodpd.f90” file. In this way, the initialisation IO operation are left unaltered and fully compatible with the serial version. A major change compared to the serial version is in the algorithm used to find the particle-particle interaction forces: in order to guarantee better coalescent access for the CUDA-threads, the algorithm has been re-adapted to the GPU architecture reordering the cell-linked list arrays.

The FORTRAN files (mainly unaltered from the serial version) are saved in the “src” folder, while the CUDA files with their corresponding headers files are in stored in the “src_cuda” folder and “headers”, respectively. The folder “bin” contains the Makefile. This arrangement of the folders allow to use the hidden Eclipse IDE project files (.cproject, .project, .settings).

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- <http://www.scd.stfc.ac.uk/SCD/support/40694.aspx>
- <http://www.scd.stfc.ac.uk/SCD/resources/PDF/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://ccpforge.cse.rl.ac.uk/gf/>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Testing

The DL_MESO code is developed using git version control. Currently the GPU version is under a branch named “add_gpu_version”. After downloading the code, checkout to the GPU branch and look into the “DPD/gpu_version” folder, i.e:

- `git clone DL_MESO_repository_path`
- `cd dl_meso`
- `git checkout gpu_version`
- `cd ./DPD/gpu_version`
- `make all`

To compile and run the code you need to have installed the CUDA-toolkit and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>).

The current version has been tested ONLY for the Mixture_Large test case available in the DEMO/DPD folder. To run the case, compile the code using the “make all” command from the “bin” directory, copy the “FIELD” and “CONTROL” files in this directory and run “./dpd_gpu.exe”.

Attention: the HISTORY file produced is currently NOT compatible with the serial version, because this is written in the C binary data format (Fortran files are organised in records, while C not. See <https://scipy.github.io/old-wiki/pages/Cookbook/FortranIO.html>).

However, you can compare the “OUTPUT” and the “export” files to verify your results. For more details see the README.rst file in the “gpu_version” folder.

Performance

Below is a table about the performance of the Mixture_Large case on different GPU cards compared to the serial version on a single core:

| CPU or GPU card | compute capability | time per cycle [s] | speedup |
|------------------------------------|--------------------|--------------------|---------|
| Intel Ivy Bridge E5-2697v2 @2.7GHz | none | 0.4740 | 1.0 |
| NVidia Tesla C1060 | 1.3 | 0.2280 | 2.1 |
| NVidia Tesla C2075 | 2.0 | 0.1830 | 2.6 |
| NVidia Tesla K40 | 3.5 | 0.1011 | 4.7 |
| NVidia Tesla K80 | 3.7 | 0.0898 | 5.3 |
| NVidia Tesla M60 | 5.2 | 0.0978 | 4.8 |
| NVidia Tesla P100 | 6.0 | 0.0390 | 12.2 |

Examples

See the Mixture_Large case in the DL_MESO manual.

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be registered as developer):

- https://ccpforge.cse.rl.ac.uk/gf/project/dl_meso/scmgit/?action=ScmCommitDetail&scm_commit_id=110906
- https://ccpforge.cse.rl.ac.uk/gf/project/dl_meso/scmgit/?action=ScmCommitDetail&scm_commit_id=111357

Software Technical Information

Name DL_MESO (DPD).

Language Fortran, CUDA-C.

Licence BSD, v. 2.7 or later

Documentation Tool ReST files

Application Documentation See the DL_MESO Manual

Relevant Training Material See DL_MESO webpage

Software Module Developed by Jony Castagna

Ewald on DL_MESO_DPD (GPU version)

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The electrostatic force calculation represent usually the main computational costs in systems where even a small amount of charged particles is present ($>1\%$). The Smooth Particle Ewald Mesh cite{SPME} splits the electrostatic forces in two parts: a short range, solved in the real space, and a long range, solved in the Fourier space. An error

weight function combines the two contributes. For the long range force the electrical charges are spread on a virtual particle mesh using a B-spline interpolation function.

Porting to GPU the full short and long range interactions allowed to maintain the speedup factor of $\times 4$ when compared to the a traditional Intel 12-core.

One of the main applications which included electrical charges are the simulations of plasma.

Purpose of Module

The Ewald summation method above described scales with $N^{1.5}$ at best, where N is the number of charged particles. The SPME allows a better scaling, $N \log(N)$, but requires a stencil domain decomposition (i.e. decomposing the domain along one direction only) to allow the FFTW library scaling with more than 1 core. If this is not used, as in the current master version of DL_MESO_DPD, the FFTW becomes rapidly a bottleneck for scaling across several nodes. On the other side, the porting to a single GPU does not need domain decomposition and the same speedup factor ($\times 4$ compared to 12-core Intel) is maintained.

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at https://gitlab.stfc.ac.uk/dl_meso/dl_meso. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

The DL_MESO code is developed using git version control. Currently the GPU version is under a branch named “add_gpu_version”. After downloading the code, checkout to the GPU branch and look into the “DPD/gpu_version” folder, i.e:

- `git clone DL_MESO_repository_path`
- `cd dl_meso`
- `git checkout gpu_version`
- `cd /DPD/gpu_version`
- `make all`

To compile and run the code you need to have installed the CUDA-toolkit and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>).

To run the case, compile the code using the “make all” command from the “bin” directory, copy the “FIELD” and “CONTROL” files in this directory and run “./dpd_gpu.exe”. The DL_MESO code is developed using git version control. Currently the GPU version is under a branch named “add_gpu_version”. After downloading the code, checkout to the GPU branch and look into the “DPD/gpu_version” folder, i.e:

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be register as developer):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/9962103c7821634a17ecb5da5460183a68951a0b

Software Technical Information

Name DL_MESO (DPD).

Language Fortran, CUDA-C.

Licence BSD, v. 2.7 or later

Documentation Tool ReST files

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

SPME on DL_MESO_DPD (GPU version)

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The electrostatic force calculation usually represents the main computational costs in systems where even a small amount of charged particles are present (>1%). The Smooth Particle Mesh Ewald [SPME] splits the electrostatic forces in two parts: a short range, solved in the real space, and a long range, solved in the Fourier space. An error weight function combines the two contributions. For the long range force the electrical charges are spread on a virtual particle mesh using a B-spline interpolation function.

Porting the full short and long range interactions to GPUs allowed us to achieve a speedup factor of 4x when compared to a traditional 12-core Intel CPU.

One of the main applications which includes electrical charges are the simulations of plasma.

Purpose of Module

The Ewald summation method scales with $N^{1.5}$ at best, where N is the number of charged particles. The SPME method allows for improved scaling, $N * \log(N)$, but requires a stencil domain decomposition (i.e. decomposing the domain along one direction only) to allow the FFTW library to scale with more than 1 core. If this is not used, as in the current master version of DL_MESO_DPD, FFTW rapidly becomes a bottleneck for scaling across several nodes. On the other hand, the porting to a single GPU does not need domain decomposition and the same speedup factor (4x compared to 12-core Intel) is maintained.

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx

- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at https://gitlab.stfc.ac.uk/dl_meso/dl_meso. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

The DL_MESO code is developed using git version control. Currently the GPU version is under a branch named “add_gpu_version”. After downloading the code, checkout the GPU branch and look into the “DPD/gpu_version” folder, i.e:

- git clone DL_MESO_repository_path
- cd dl_meso
- git checkout gpu_version
- cd /DPD/gpu_version
- make all

To compile and run the code you need to have installed the CUDA-toolkit and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>).

To run the case, compile the code using the “make all” command from the “bin” directory, copy the “FIELD” and “CONTROL” files in this directory and run “./dpd_gpu.exe”.

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be register as developer):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/34a652fe62cadbac5e8a037b57ee9be64dcf4187

Software Technical Information

Name DL_MESO (DPD).

Language Fortran, CUDA-C.

Licence BSD, v. 2.7 or later

Documentation Tool ReST files

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

Improved overlap computation communication in DL_MESO_DPD (multi-GPU version)

- *Purpose of Module*
- *Background Information*

- *Building and Testing*
- *Source Code*

The following module present an improved overlap between communication and computation for the DL_MESO_DPD package on multi-GPUs.

A binary mixture phase separation test case up to 1.8 billion particles has been used for weak and strong benchmarks. The results show good scaling in both cases up to 1024 GPUs. After that the scaling without improved overlap quickly tails off while the other shows good efficiency (>85%) up to 4096 GPUs.

Purpose of Module

The previous multi-GPU version of DL_MESO_DPD was not correctly setting the order of the CUDA streams dedicated to computation and communication. This was preventing their overlap and drastically reduce the overall performance and scalability. The current module fixes this problem and present weak and strong scaling on the Piz Daint Supercomputer (see <https://user.cscs.ch/>) using up to 4096 GPUs. The previous performance is presented for comparison.

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- <http://www.scd.stfc.ac.uk/SCD/support/40694.aspx>
- <http://www.scd.stfc.ac.uk/SCD/resources/PDF/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk/>. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

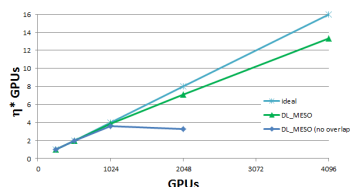
Building and Testing

The DL_MESO code is developed using git version control. Currently the GPU version is under a branch named “multi_GPU_version”. After downloading the code, checkout to the GPU branch and look into the “DPD/gpu_version” folder, i.e:

- `git clone DL_MESO_repository_path`
- `cd dl_meso`
- `git checkout multi_GPU_version`
- `cd /DPD/gpu_version/bin`
- `make all`

To compile and run the code you need to have installed the CUDA-toolkit, a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>), a fortran compiler (like GCC gfortran, Intel Fortran, Cray ftn) and MPI library. Moreover, the code uses CUDA_aware_MPI which is part of GPU Direct Technologies. Please make sure your cluster support CUDA_aware_MPI!

The current version has been tested ONLY for the Mixture_Large test case available in the DEMO/DPD folder. To run the case, compile the code using the “make all” command from the “bin” directory, copy the “FIELD” and “CONTROL” files in this directory and run “`mpirun -np N ./dpd_gpu.exe`”. For a the strong scaling test we used 1.8 billion particles keeping the density ratio particles/volume=5. Below is a plot of the strong scaling with and without improved overlap.



Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be registered as developer):

- https://gitlab.stfc.ac.uk/srb73435/dl_meso/commit/90701a3ad97d53dc0555d0b79862e0db3134f83c

Software Technical Information

Name DL_MESO (DPD).

Language Fortran, CUDA-C.

Licence BSD, v. 2.7 or later

Documentation Tool ReST files

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

Bond forces on DL_MESO_DPD (single GPU)

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

This module add the bond forces to the single GPU version of DL_MESO (DPD). These take in account in the interactions between different chemical species which allow to create complex molecules more representative of real systems. An example of an application is the ternary solution where a primary component is bonds interacting with the other two phases.

Purpose of Module

The algorithm used is the same of the DL_MESO serial version, but of course adapted for SIMT (Single Instruction Multiple Threads) architecture. The module includes also the angle and dihedral forces, all divided according a classical orthogonal domain decomposition.

Considering that in a real case the number of bounds is usually much lower than the total number of particles, different CUDA streams for the three kernels (`k_findBondForce`, `k_findAngleForce` and

`k_findDihedralForce`) are used. This allow to launch them in parallel to improve the performance of the overall simulation.

Note: If the module is an ingredient for a more general workflow (e.g. the module was the necessary foundation for later code; the module is part of a group of modules that will be used to calculate certain property or have certain application, etc.) mention this, and point to the place where you specify the applications of the more general workflow (that could be in another module, in another section of this repository, an application's website, etc.).

Note: If you are a post-doc who works in E-CAM, an obvious application for the module (or for the group of modules that this one is part of) is your pilot project. In this case, you could point to the pilot project page on the main website (and you must ensure that this module is linked there).

Background Information

This module is part of the DL_MESO_DPD code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at https://gitlab.stfc.ac.uk/dl_meso/dl_meso. Please contact Dr. Micheal Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

The DL_MESO code is developed using git version control. Currently there are a single GPU version and a multi GPU version is under a two different branches. After downloading the code, checkout the `single_GPU_version` branch and look into the `DPD/gpu_version` folder, i.e:

```
git clone DL_MESO_repository_path
cd dl_meso
git checkout single_GPU_version
cd /DPD/gpu_version
make all
```

To compile and run the code you need to have installed the CUDA-toolkit and have a CUDA enabled GPU device (see <http://docs.nvidia.com/cuda/#axzz4ZPtFifjw>).

To run the case, compile the code using the `make all` command from the `bin` directory, copy the `FIELD` and `CONTROL` files from the `gpu_version/test/Solvent` folder in this directory and run `./dpd_gpu.exe`.

Source Code

This module has been merged into DL_MESO code. It is composed of the following commits (you need to be register as developer):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/c787c4bc4f56634c2c6c6730b98b75093907ce57
- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/8b3dd9c071a60cbfb6e5fc285e82049efcc603f7
- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/commit/9c6452d2340c53dc68d1eabbee37992d14e071b5

The DL_MESO code is developed using git version control. Currently the GPU version is under a branch named “add_gpu_version”.

Software Technical Information

Name DL_MESO

Language Fortran/C++

Licence BSD, v. 2.7 or later

Documentation Tool Fortran/C comments

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

DL_MESO (DPD) on Kokkos: Verlet Velocity step 1

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Performance*
- *Experience in porting to Kokkos*
- *Source Code*

This module is the first version of a performance portable version of DL_MESO (DPD) using the [Kokkos library](#). Its focus is on the first loop of the Verlet Velocity (VV) scheme for the time marching scheme.

Purpose of Module

In this module we present a first version of DL_MESO (DPD) with [Kokkos](#) library which offloads one of the main steps of the time marching scheme during the force integration. This allows us to run DL_MESO on NVidia GPUs as well as on other GPUs or architectures (many-core hardware like KNL), allowing performance portability as well as a separation of concerns between computational science and HPC.

The VV scheme is made of 3 steps:

- 1) a first velocity and particle positions integration by $\Delta t/2$,
- 2) a force calculation, and
- 3) a second velocity integration by $\Delta t/2$.

In this module we are porting to Kokkos the first step.

Note: Kokkos is a C++ library, while DL_MESO (DPD) is in Fortran90 Language. The current implementation requires a transfer between Fortran to C++, due to the use of Fortran pointers not bound using the `ISO_C_BINDING` standard. This constraint will be removed in future versions of DL_MESO.

Background Information

With the advent of heterogeneous hardware, achieving performance portability across different architectures is one of the main challenges in HPC. In fact, while specific languages, like CUDA, can give best performance for the NVIDIA hardware, they cannot be used with different GPU vendors limiting the usage across supercomputers world wide.

In this module we use Kokkos, developed at Sandia National Laboratories, which consist of several C++ templated libraries able to offload the workload to several different architectures, taking care of the memory layout and transfer between host and device.

To install Kokkos follow the instructions at: [Kokkos Tutorial](#). This module has been built with a Kokkos installation using the following flags:

```
cmake      ../      -CMAKE_CXX_COMPILER=$HOME/Kokkos/kokkos/bin/nvcc_wrapper
-DKokkos_ENABLE_CUDA=ON      -DKokkos_ENABLE_OPENMP=ON      -
DKokkos_ENABLE_CUDA_LAMBDA=ON -DCMAKE_INSTALL_PREFIX=$HOME/Kokkos/kokkos
```

which allows for the translation of the Kokkos kernel to CUDA for running on NVidia GPUs.

This module is part of the DL_MESO (DPD) code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Michael Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

To compile and run the code you need to have installed a Fortran and C++ compiler (GCC>7.x), CMake (>3.11.4) and Kokkos.

The DL_MESO code is developed using git version control. Currently, the Kokkos GPU version is under a branch named Kokkos_version. After downloading the code, checkout the Kokkos branch and move to the DPD folder. Use cmake to build and compile the executable:

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout kokkos_version
cd ./DPD
mkdir build
cd build
cmake ../
cmake --build .
```

Use the files FIELD and CONTROL files in DEMO/DPD folders to test your code on different architectures. Compare the OUTPUT and the export files to verify your results.

Performance

We timed the execution for the VV kernel using Kokkos and compared to the same loop written in CUDA language (see [DL_MESO GPU version modules](#)) using a Volta V100 NVidia card.

For 5.12 million particles of the Large Mixture test case, we get a 0.00114s per kernel execution with both versions, which indicate no loss of performance in using Kokkos compared to native CUDA code. However, the data transfer between host and device currently occurs at every time step in the Kokkos version, taking 0.5721s and therefore a negative impact on the overall performance.

For a fair comparison, this data should be transferred upstream to the time marching loop as done in the CUDA version.

Experience in porting to Kokkos

Compared to other paradigms used for GPU programming, like OpenACC or OpenMP, Kokkos has quite a steep learning curve. This is due to several concepts which needs to be considered by the programmer before starting the porting. Some of these concepts are familiar to C++ programmers, like the use of lambda functions and function objects (commonly known as functors). Another important concept is the *Memory Space* which is different according to the hardware used. The transfer between host and device is based around the concept of the *View* layout in the Memory Space, an array of one or more dimensions which can be set at compile time or runtime. Programmers familiar with CUDA will easily recognize some similarity when porting to GPU, like the concepts of host and device asynchronism and the Unified Memory memory space. After following the first [on line tutorial](#), the porting of a simple loop should be straight forward. However, more advanced concepts are required for more complex scientific kernels to achieve performance portability. Finally, the error messages are not always very easy to interpret and the online threads to similar issues are still relatively few. With time, wider usage of the library will definitely improve the usability and make it more user friendly (and less verbose).

Source Code

This module has been pushed into DL_MESO git repository. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/-/commit/6b58be6b23d823ef4224b06ac5b1ca089fea56ef
- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/-/commit/9f9ea4563986cf43562af67b8a60a5cdf9615016
- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/-/commit/a285d3c93492ac540d342025d5c3f0ca61f8b295
- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/-/commit/a315ef48d44f7924e51c27748e0e9761adebea5

Software Technical Information

Name DL_MESO

Language Fortran/C++

Licence BSD, v. 2.7 or later

Documentation Tool Fortran/C comments

Application Documentation See the [DL_MESO Manual](#)

Relevant Training Material See [DL_MESO webpage](#)

Software Module Developed by Jony Castagna

DL_MESO (DPD) on Kokkos: Verlet Velocity step 2

- *Purpose of Module*
- *Background Information*
- *Building and Testing*

- [Performance](#)
- [Source Code](#)

This module is related to the implementation in DL_MESO (DPD) [Kokkos library](#) to achieve performance portability. It's focus is on the second loop of the Verlet Velocity (VV) scheme for the time marching scheme.

Purpose of Module

In this module we are porting to DL_MESO (DPD) the second loop of the Verlet Velocity scheme to [Kokkos](#). This allows to run DL_MESO on NVidia GPUs as well as on other GPUs or architectures (many-core hardware like KNL), allowing performance portability as well as separation of concern between computational science and HPC.

The VV scheme is made of 3 steps: 1) a first velocity and particle positions integration by $\Delta t/2$, 2) a force calculation and 3) a second velocity integration by $\Delta t/2$. We have already offloaded the first loop in another module (*DL_MESO (DPD) on Kokkos: Verlet Velocity step 1*). The main difference here is that we are using a `parallel_reduce` loop rather than `parallel_for` as we need to calculate also the stress tensor via reduction operations.

Note: Kokkos is a C++ library, while DL_MESO (DPD) is in Fortran90 Language. The current implementation requires a transfer between Fortran and C++, due to the use of Fortran pointers not binded via the ISO_C_BINDING standard. This constraint will be removed in future versions.

Background Information

With the advent of heterogeneous hardware, achieving performance portability across different architectures is one of the main challenges in HPC. In fact, while specific languages, like CUDA, can give best performance for the NVidia hardware, they cannot be used with different GPU vendors limiting the usage across supercomputers worldwide.

In this module we use Kokkos, developed at Sandia National Laboratories, which consist of several C++ templated libraries which provide the capability to offload a workload to several different architectures, taking care of the memory layout and transfer between host and device.

To install Kokkos follow the instructions at: [Kokkos Tutorial](#). This module has been built on a Kokkos installation using the following flags:

```
cmake ../ -CMAKE_CXX_COMPILER=$HOME/Kokkos/kokkos/bin/nvcc_wrapper -DKokkos_ENABLE_
↪CUDA=ON \
  -DKokkos_ENABLE_OPENMP=ON -DKokkos_ENABLE_CUDA_LAMBDA=ON -DCMAKE_INSTALL_PREFIX=
↪$HOME/Kokkos/kokkos
```

which allows us to translate the Kokkos kernel to CUDA language and run on NVidia GPUs.

This module is part of the DL_MESO (DPD) code. Full support and documentation is available at:

- https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
- <https://www.scd.stfc.ac.uk/Pages/USRMAN.pdf>

To download the DL_MESO_DPD code you need to register at <https://gitlab.stfc.ac.uk>. Please contact Dr. Michael Seaton at Daresbury Laboratory (STFC) for further details.

Building and Testing

To compile and run the code you need to have installed a Fortran and C++ compiler (GCC>7.x), CMake (>3.11.4) and Kokkos.

The DL_MESO code is developed using git version control. Currently, the Kokkos GPU version is under a branch named `Kokkos_version`. After downloading the code, checkout the Kokkos branch and move to the `DPD` folder. Use `cmake` to build and compile the executable:

```
git clone https://gitlab.stfc.ac.uk/dl_meso.git
cd dl_meso
git checkout kokkos_version
cd ./DPD
mkdir build
cd build
cmake ../
cmake --build .
```

Use the files `FIELD` and `CONTROL` files in `DEMO/DPD` folders to test your code on different architectures. Compare the `OUTPUT` and the `export` files to verify your results.

Performance

We timed the execution for the VV second step kernel using Kokkos and compared to the same loop written in CUDA language (see [DL_MESO GPU version modules](#)) using a Volta V100 NVidia card. For a 5.12 million particles of the Large Mixture test case, we get a 0.00117s (very close to the fist loop, despite the reduction operations) per kernel execution with both versions, which indicate no loss of performance in using Kokkos compared to native CUDA code. However, the data transfer between host and device currently occurs at every time step in the Kokkos version, taking 0.4689s and then with a negative impact on the overall performance. For a fair comparison, this data should be transferred upstream to the time marching loop as done in the CUDA version.

Source Code

This module has been pushed into DL_MESO git repository. It is composed of the following commits (you need to be registered as collaborator):

- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/-/commit/4d32671264648b4252f71c2f98d0164ab0843f46
- https://gitlab.stfc.ac.uk/dl_meso/dl_meso/-/commit/457509dc8727d30b49f4bb70a4bec98126866447

4.3.2 ESPResSo++

The following modules connected to the ESPResSo++ code have been produced so far in the context of an [associated Pilot Project](#):

Hierarchical Strategy for Simple One-Component Polymer Melts: fixed-local-tuple

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Languages: Python (2.7) and C++

Documentation Tools: Sphinx and Doxygen

Application Documentation: <http://espressopp.github.io/>

Relevant Training Material: <https://github.com/esspressopp/esspressopp/tree/master/examples>

Licence GNU General Public License

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

Reference¹ describes the principles of a hierarchical strategy to equilibrate simple one-component polymer melts described in terms of atomistic or coarse-grained (bead-spring) models. The present module is part of our implementation of this method in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module is part of a suite of programs which realize this method within the framework of the simulation package ESPResSO++.

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. One thus obtains a configuration that is equilibrated on large scales but does not provide information about the structure on smaller (i.e. more fine-grained (FG)) scales.

To obtain the latter, the resolution is step-by-step increased by recursively applying a fine-graining procedure to the previous (more coarse-grained) level. In such a fine-graining step, each CG polymer chain is replaced with a more fine-grained chain, by dividing a CG blob into several FG blobs. In the last step, microscopic monomers are reinserted into their CG blobs.

The resulting set of FG blobs is set up in such a way that its conformation is consistent with the conformation at the more coarse-grained level. After this setup, the local FG conformation is relaxed into a local equilibrium, again consistent with the (fixed) CG blobs.

This procedure needs a data structure where tuples of particles are stored in lists. In contrast to simple Molecular Dynamics, which is based only on pairs of particles, we here need to allow tuples of arbitrary size. The present module provides this more general data structure. The list can contain both real and ghost particles.

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://esspressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/esspressopp/esspressopp>

¹ <http://pubs.acs.org/doi/abs/10.1021/mz5000015>, preprint available via <https://arxiv.org/abs/1610.07511>

Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested by a Python script found under the following link:

- <https://github.com/espressopp/espressopp/tree/master/testsuite/FixedLocalTuple>

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espressopp/espressopp/pull/171>

References

Hierarchical Strategy for Simple One-Component Polymer Melts: md-softblob

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Languages: Python (2.7) and C++

Documentation Tools: Sphinx and Doxygen

Application Documentation: <http://espressopp.github.io/>

Relevant Training Material: <https://github.com/espressopp/espressopp/tree/master/examples>

Licence GNU General Public License

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

Reference¹ describes the principles of a hierarchical strategy to equilibrate simple one-component polymer melts described in terms of atomistic or coarse-grained (bead-spring) models. The present module is the central part of our implementation of this method in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation

¹ : <http://pubs.acs.org/doi/abs/10.1021/mz5000015>, preprint available via <https://arxiv.org/abs/1610.07511>

theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module is the central part that controls a suite of programs which realize this method within the framework of the simulation package ESPResSO++.

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. One thus obtains a configuration that is equilibrated on large scales but does not provide information about the structure on smaller (i.e. more fine-grained (FG)) scales.

To obtain the latter, the resolution is step-by-step increased by recursively applying a fine-graining procedure to the previous (more coarse-grained) level. In such a fine-graining step, each CG polymer chain is replaced with a more fine-grained chain, by dividing a CG blob into several FG blobs. In the last step, microscopic monomers are reinserted into their CG blobs.

The resulting set of FG blobs is set up in such a way that its conformation is consistent with the conformation at the more coarse-grained level. After this setup, the local FG conformation is relaxed into a local equilibrium, again consistent with the (fixed) CG blobs.

The present module implements the actual coarse-graining step, which is therefore described in more detail:

A polymer chain, originally consisting of N monomers, is replaced by a coarse-grained (CG) chain consisting of N/N_b soft blobs linked by a harmonic bond potential, $V_{bond} = 3k_B T d^2 / 2b_{CG}^2$, and an angular bond-bending potential $V_{bend} = k_B T k_{bend} (1 + \cos(\theta)) / 2$. Here d is the distance and θ is the angle between consecutive bonds. The interactions between non-bonded soft blobs are taken into account by a repulsive pair potential $V_{nb} = k_B T \epsilon U_G(r_{ij})$. Here r_{ij} is the center-to-center distance between the two blobs, $U_G(r_{ij})$ is a Gaussian function with variance $\bar{\sigma}^2 = \sigma_i^2 + \sigma_j^2$ and σ_i is the gyration radius of blob number i . The gyration radius σ is in turn fluctuating. This fluctuation is controlled by the potential $V_{sphere} = k_B T (a_1 N_b^3 \sigma^{-6} + a_2 N_b^{-1} \sigma^2 + a_3 \sigma^{-3})$. Equilibrated configurations of soft blobs are generated by Molecular Dynamics (MD) simulations based on the above model.

Within the module, the following classes have been implemented or modified:

- A `VSpherePair` class for calculating $V_{nb} = k_B T \epsilon U_G(r_{ij})$
- A `VSphereSelf` class for calculating $V_{sphere} = k_B T (a_1 N_b^3 \sigma^{-6} + a_2 N_b^{-1} \sigma^2 + a_3 \sigma^{-3})$
- A `Particle` class for communicating the property “radius” between different cores
- A `LangevinThermostatOnRadius` class for simulating the fluctuations of the radii of the blobs

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested by a Python script found under the following link:

- https://github.com/espressopp/espressopp/tree/master/testsuite/langevin_thermostat_on_radius

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espresopp/espresopp/pull/168>
- <https://github.com/espresopp/espresopp/pull/169>
- <https://github.com/espresopp/espresopp/pull/170>
- <https://github.com/espresopp/espresopp/pull/176>

References

Minimize Energy : A Component of the Hierarchical Equilibration Strategy for Polymer Melts

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Languages: Python (2.7) and C++

Documentation Tools: Sphinx and Doxygen

Application Documentation: <http://espresopp.github.io/>

Relevant Training Material: <https://github.com/espresopp/espresopp/tree/master/examples>

Licence GNU General Public License

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

The module is an implementation of a part of a hierarchical strategy¹ for the equilibration of simple one-component polymer melts in ESPResSo++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module provides a part of that method.

¹ : <http://pubs.acs.org/doi/abs/10.1021/mz5000015>

When the microscopic monomers are re-inserted into the soft blobs, the polymer configurations should satisfy a local energy minimum to avoid overlap between particles. This module provides a steepest-descent method <https://en.wikipedia.org/wiki/Gradient_descent> which is a typical energy minimization method.

This module is a modification of the already existing class *espressopp.integrator.MinimizeEnergy*. The modifications are as follows:

1. Corrected the procedure of particle redistribution to cells.
2. A variable relaxation of the energy per step γ is implemented. In this case, the position of particles is updated following the equation: $p_{i+1} = p_i + (d_{\max}/f_{\max}) F_i$ where d_{\max} is the maximum update of particle coordinates in a single steepest-descent step and γ is adjusted via $\gamma = d_{\max}/f_{\max}$ where f_{\max} is the maximum force in a single step.

These modifications significantly stabilize the procedure of redistributing particles to cells, and any value d_{\max} less than half the skin parameter of the Verlet list can be used.

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested by a Python script found under the following link:

- https://github.com/espressopp/espressopp/tree/master/testsuite/minimize_energy

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espressopp/espressopp/pull/89>
- <https://github.com/espressopp/espressopp/pull/90>

References

Hierarchical Strategy for Simple One-Component Polymer Melts: constrain-com

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Languages: Python (2.7) and C++

Documentation Tools: Sphinx and Doxygen

Application Documentation: <http://espressopp.github.io/>

Relevant Training Material: <https://github.com/espressopp/espressopp/tree/master/examples>

Licence GNU General Public License

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

Reference¹ describes the principles of a hierarchical strategy to equilibrate simple one-component polymer melts described in terms of atomistic or coarse-grained (bead-spring) models. The present module is part of our implementation of this method in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module is part of a suite of programs which realize this method within the framework of the simulation package ESPResSO++.

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. One thus obtains a configuration that is equilibrated on large scales but does not provide information about the structure on smaller (i.e. more fine-grained (FG)) scales.

To obtain the latter, the resolution is step-by-step increased by recursively applying a fine-graining procedure to the previous (more coarse-grained) level. In such a fine-graining step, each CG polymer chain is replaced with a more fine-grained chain, by dividing a CG blob into several FG blobs. In the last step, microscopic monomers are reinserted into CG blobs.

The resulting set of FG blobs is set up in such a way that its conformation is consistent with the conformation at the more coarse-grained level. After this setup, the local FG conformation is relaxed into a local equilibrium, again consistent with the (fixed) CG blobs. Consistency here means that the center of mass (COM) of the set of FG blobs coincides with the center of the corresponding CG blob, during an initial period which is long enough that nearly perfect local equilibrium is reached. After that, the constraint is lifted.

The present module provides the C++ class for applying a suitable constraint that conserves the position of the COM of N FG blobs.

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ via the following links:

¹ <http://pubs.acs.org/doi/abs/10.1021/mz5000015>, preprint available via <https://arxiv.org/abs/1610.07511>

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested by a Python script found under the following link:

- https://github.com/espressopp/espressopp/tree/master/testsuite/constrain_com

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espressopp/espressopp/pull/178>

References

Hierarchical Strategy for Simple One-Component Polymer Melts: constrain-rg

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Languages: Python (2.7) and C++

Documentation Tools: Sphinx and Doxygen

Application Documentation: <http://espressopp.github.io/>

Relevant Training Material: <https://github.com/espressopp/espressopp/tree/master/examples>

Licence GNU General Public License

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

Reference¹ describes the principles of a hierarchical strategy to equilibrate simple one-component polymer melts described in terms of atomistic or coarse-grained (bead-spring) models. The present module is part of our implementation of this method in ESPResSo++.

¹ <http://pubs.acs.org/doi/abs/10.1021/mz5000015>, preprint available via <https://arxiv.org/abs/1610.07511>

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module is part of a suite of programs which realize this method within the framework of the simulation package ESPResSO++.

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. One thus obtains a configuration that is equilibrated on large scales but does not provide information about the structure on smaller (i.e. more fine-grained (FG)) scales.

To obtain the latter, the resolution is step-by-step increased by recursively applying a fine-graining procedure to the previous (more coarse-grained) level. In such a fine-graining step, each CG polymer chain is replaced with a more fine-grained chain, by dividing a CG blob into several FG blobs. In the last step, microscopic monomers are reinserted into their CG blobs.

The resulting set of FG blobs is set up in such a way that its conformation is consistent with the conformation at the more coarse-grained level. After this setup, the local FG conformation is relaxed into a local equilibrium, again consistent with the (fixed) CG blobs.

In the last step (the reinsertion of microscopic monomers) it is useful to avoid an initial overstretching of microscopic bonds as much as possible. To this end, the algorithm makes sure that the gyration radii of the subchains coincide with the gyration radii of the corresponding blobs, during an initial equilibration period.

The present module provides the C++ class for applying a suitable constraint that conserves the gyration radius of N microscopic monomers.

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested by a Python script found under the following link:

- https://github.com/espressopp/espressopp/tree/master/testsuite/constrain_rg

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espressopp/espressopp/pull/182>

References

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Name Feedback control mechanism for one component melts

Language Python (2.7) and C++

Licence GPL <<https://opensource.org/licenses/gpl-license>>

Documentation Tool Sphinx and Doxygen

Application Documentation <http://espressopp.github.io/>

Relevant Training Material <https://github.com/espressopp/espressopp/tree/master/examples>

Feedback control mechanism: A Component of the Hierarchical Equilibration Strategy for Polymer Melts

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*
- *References*

The module is an implementation of the existing hierarchical strategy¹ for the equilibration of simple one-component polymer melts in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module provides a part of that method described below.

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. One thus obtains a configuration that is equilibrated on large scales but does not provide information about the structure on smaller (i.e. more fine-grained (FG)) scales.

To obtain the latter, the resolution is step-by-step increased by recursively applying a fine-graining procedure to the previous (more coarse-grained) level. In such a fine-graining step, each CG polymer chain is replaced with a more fine-grained chain, by dividing a CG blob into several FG blobs.

¹ <http://pubs.acs.org/doi/abs/10.1021/mz5000015>

In the last step, microscopic monomers are reinserted into CG blobs. This reinsertion procedure is divided into 2 parts. Firstly, monomers are treated as mass points without non-bonded interaction. Starting from this state, repulsive non-bonded interactions are gradually introduced according to the feedback control mechanism explained in Ref.². This procedure makes sure that the final fine-grained conformation is consistent with the conformation at the more coarse-grained level.

The present module provides the python script which performs the feedback control mechanism. The implementation detail is in following below.

1. The microscopic configuration of N polymers consisted of M monomers is prepared. The system size L is determined by the number of density $\rho = (N \times M)/L^3 \approx 0.85$. m and σ stands for the mass and the diameter of monomers.

We presupposed that a configuration is already equilibrated at a coarse-grained level and is not equilibrated at a microscopic level.

2. NVT MD simulation is carried out with bonding potential V_{FENE} and force-capped-LJ potential $V_{\text{fc-LJ}}$ defined as

$$V_{\text{fc-LJ}} = (r - r_{fc})V_{\text{LJ}}(r_{fc}) + V_{\text{LJ}}(r_{fc}) \text{ for } r < r_{fc},$$

$$V_{\text{fc-LJ}} = V_{\text{LJ}}(r) \text{ for } r > r_{fc},$$

for preventing too strong repulsive forces. At first we set $r_{fc} = 2^{1/6}\sigma$.

3. The excluded volume interaction can be gradually introduced with gradually decreasing r_{fc} . In order to obtain the equilibrated structure of polymer melts, r_{fc} is controlled by the difference between the mean-square internal distances of the current configuration and that of the ideal curve in the intermediate region. This difference is defined as

$$I = \int_{20}^{50} [(\langle R^2(n) \rangle / n)_{\text{ideal}} - (\langle R^2(n) \rangle / n)_{\text{current}}] dn,$$

where $R(n)$ is an internal distance of chain segment of length n . For $I < 0$, r_{fc} is increased. In contrast, for $I > 0$, r_{fc} is decreased.

4. After performing during 650τ , MD simulation is finished. Where $\tau = \sqrt{m\sigma^2/k_B T}$.

More detail of this feedback control mechanism is explained in Ref².

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Building and Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested according to the README file found under the following link:

- https://github.com/espressopp/espressopp/tree/master/examples/hierarchical_strategy_for_one-component/

² <http://onlinelibrary.wiley.com/doi/10.1002/mats.201500013/full>

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espressopp/espressopp/pull/213>

References

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Name Reinsertion procedure for one component melts

Language Python (2.7) and C++

Licence GPL <<https://opensource.org/licenses/gpl-license>>

Documentation Tool Sphinx and Doxygen

Application Documentation <http://espressopp.github.io/>

Relevant Training Material <https://github.com/espressopp/espressopp/tree/master/examples>

Reinsertion: A Component of the Hierarchical Equilibration Strategy for Polymer Melts

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*
- *References*

The module is an implementation of the existing hierarchical strategy¹ for the equilibration of simple one-component polymer melts in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module provides a part of that method described below.

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. One thus obtains a configuration that is equilibrated on large scales but does not provide information about the structure on smaller (i.e. more fine-grained (FG)) scales.

¹ <http://pubs.acs.org/doi/abs/10.1021/mz5000015>

To obtain the latter, the resolution is step-by-step increased by recursively applying a fine-graining procedure to the previous (more coarse-grained) level. In such a fine-graining step, each CG polymer chain is replaced with a more fine-grained chain, by dividing a CG blob into several FG blobs.

In the last step, microscopic monomers are reinserted into CG blobs. This reinsertion procedure is divided into 2 parts. Firstly, monomers are treated as mass points without non-bonded interaction. Starting from this state, repulsive non-bonded interactions are gradually introduced according to the feedback control mechanism explained in Ref.². This procedure makes sure that the final fine-grained conformation is consistent with the conformation at the more coarse-grained level.

The present module provides the python script which performs the reinsertion procedure. The implementation detail is in following below.

1. The microscopic configuration of N polymers consisted of M monomers is prepared. The system size L is determined by the number of density $\rho = (N \times M)/L^3 \approx 0.85$. m and σ stands for the mass and the diameter of monomers.

We presuppose that equilibrated CG chain at $N_b = 25$ is already obtained.

2. 25 monomers of microscopic model are reinserted into all softblobs at $N_b = 25$ randomly.
3. NVT MD simulation is carried out with bonding potential V_{FENE} , non bonding potential V_{LJ} only for connected monomers,

the constrain potential for the position described as

$$V_{\text{com}}(\mathbf{r}_{\text{com}}^i) = k_{\text{com}}(\mathbf{r}_{\text{com}}^i - \mathbf{R}_i^{N_b})^2, \text{ where } \mathbf{r}_{\text{com}}^i \equiv \frac{1}{25} \sum_{j=1}^{25} \mathbf{r}_{(j+25i)},$$

and the constrain potential for the radius of gyration defined as

$$V_{R_g}(\rho_i) = k_{R_g}(\rho_i^2 - R_g^i)^2, \text{ where, } \rho_i^2 \equiv \frac{1}{25} \sum_{j=1}^{25} (\mathbf{r}_{(j+25i)} - \mathbf{R}_i^{N_b=25})^2.$$

In this time, MD simulation is performed without the excluded volume effect during dozens τ_{mon} , that stands for $\sqrt{m\sigma^2/k_B T}$.

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Building and Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested according to the README file found under the following link:

- https://github.com/espressopp/espressopp/tree/master/examples/hierarchical_strategy_for_one-component/

² <http://onlinelibrary.wiley.com/doi/10.1002/mats.201500013/full>

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espressopp/espressopp/pull/213>

References

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Name Fine-graining procedure for one component melts

Language Python (2.7) and C++

Licence GPL <<https://opensource.org/licenses/gpl-license>>

Documentation Tool Sphinx and Doxygen

Application Documentation <http://espressopp.github.io/>

Relevant Training Material <https://github.com/espressopp/espressopp/tree/master/examples>

Fine-graining: A Component of the Hierarchical Equilibration Strategy for Polymer Melts

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*
- *References*

The module is an implementation of the existing hierarchical strategy¹ for the equilibration of simple one-component polymer melts in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module provides a part of that method described below.

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. One thus obtains a configuration that is equilibrated on large scales but does not provide information about the structure on smaller (i.e. more fine-grained (FG)) scales.

¹ <http://pubs.acs.org/doi/abs/10.1021/mz5000015>

To obtain the latter, the resolution is step-by-step increased by recursively applying a fine-graining procedure to the previous (more coarse-grained) level. In such a fine-graining step, each CG polymer chain is replaced with a more fine-grained chain, by dividing a CG blob into several FG blobs.

The present module provides the python script which performs this fine-graining procedure. The implementation detail is in following below.

1. The microscopic configuration of N polymers consisted of M monomers is prepared. The system size L is determined by the number of density $\rho = (N \times M)/L^3 \approx 0.85$. m and σ stands for the mass and the diameter of monomers.

We presuppose that equilibrated CG chain at N_b is already obtained.

2. The softblobs at N_b is divided into 2 softblobs at $N_b/2$ under the constraint conditions defined as

$$\mathbf{R}_i^{N_b} = \frac{\mathbf{R}_{2i-1}^{N_b/2} + \mathbf{R}_{2i}^{N_b/2}}{2} \equiv \mathbf{r}_{\text{com}}^i,$$

$$R_{g,N_b/2}^{(2i-1)} = R_{g,N_b/2}^{2i} = \frac{R_{g,N_b}^i}{\sqrt{2}},$$

where $\mathbf{R}_i^{N_b}$ stands for \mathbf{R}_i at N_b and R_{g,N_b}^i stand for R_g^i at N_b . Namely, the center of mass of 2 softblobs at $N_b/2$ is identical with the position of a softblob at N_b .

3. For equilibrating a local configuration at $N_b/2$, NVT MD simulation is performed.

In the beginning, a MD simulation takes into account bonding potential V_{bond} ,

the potential for fluctuating radius of gyration V_{sphere} and the constrain potentials for center of mass described as

$$V_{\text{com}}(\mathbf{r}_{\text{com}}^i) = k_{\text{com}}(\mathbf{r}_{\text{com}}^i - \mathbf{R}_i^{N_b})^2.$$

Each $16\tau_{\text{blob}}$, MD simulation is including the bending interactions V_{angle}

and non bonding interactions V_{nb} in this order.

Where $\tau_{\text{blob}} = \sqrt{mN_b\sigma^2/k_B T}$.

4. After including all interactions, MD simulation is performed during $16\tau_{\text{blob}}$.
5. In order to obtain the snapshot which has the ideal mean square internal distance (MSID) $\langle R(n)^2 \rangle$, MD simulation is continued to carry out. Where $\text{MSID} \langle R(n)^2 \rangle$ is defined as

$$\langle R(n)^2 \rangle \equiv \frac{1}{M/N_b - n} \sum_{j=0}^{N-1} \sum_{i=1}^{M/N_b - n} (\mathbf{R}_{i+(M/N_b)j} - \mathbf{R}_{i+(M/N_b)j+n})^2.$$

This is calculated in each τ_{blob} .

After obtaining good snapshot at $N_b/2$, fine-graining procedure is finished.

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Building and Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested according to the README file found under the following link:

- https://github.com/esspressopp/esspressopp/tree/master/examples/hierarchical_strategy_for_one-component/

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/esspressopp/esspressopp/pull/213>

References

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Name Coarse-graining procedure for one component melts

Language Python (2.7) and C++

Licence GPL <<https://opensource.org/licenses/gpl-license>>

Documentation Tool Sphinx and Doxygen

Application Documentation <http://esspressopp.github.io/>

Relevant Training Material <https://github.com/esspressopp/esspressopp/tree/master/examples>

Coarse-Graining: A Component of the Hierarchical Equilibration Strategy for Polymer Melts

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*
- *References*

The module is an implementation of the existing hierarchical strategy¹ for the equilibration of simple one-component polymer melts in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module provides a part of that method described below.

¹ <http://pubs.acs.org/doi/abs/10.1021/mz5000015>

To decrease the relaxation time, microscopic monomers are coarse-grained (CG) by mapping each subchain with N_b monomers onto a soft blob. The CG system is then characterized by a much lower molecular weight and thus is equilibrated quickly. The present module provides a python script which performs this coarse-graining procedure. The implementation detail is in following below.

1. The microscopic configuration of N polymers consisted of M monomers is prepared. The system size L is determined by the number of density $\rho = (N \times M)/L^3 \approx 0.85$. m and σ stands for the mass and the diameter of monomers.
2. The configuration of N CG chain at $N_b = 100$ is generated from the microscopic configuration.

The position of i -th softblobs \mathbf{R}_i is determined by $\mathbf{R}_i = \frac{1}{N_b} \sum_{j=1}^{N_b} \mathbf{r}_{(j+(i-1)N_b)}$, where \mathbf{r}_i stands for the position of i -th monomers.

The radius of gyration i th softblobs R_g^i is also determined by $R_g^i = \frac{1}{N_b} \sum_{j=1}^{N_b} (\mathbf{r}_{(j+(i-1)N_b)} - \mathbf{R}_i)^2$.

3. The CG configuration is equilibrated by NVT MD simulation with mass of softblobs $M = N_b m$ during the equilibration time τ_r defined as $\tau_r \sim \left(\frac{M}{N_b}\right)^2 \tau_{\text{blob}}$, where $\tau_{\text{blob}} = \sqrt{m N_b \sigma^2 / k_B T}$.

Hence, CPU time τ_{100} for softblobs with $N_b = 100$ is estimated as $\tau_{100} \sim N \times \left(\frac{M}{N_b}\right)^3 \tau_{\text{blob}}$.

4. After equilibrating a configuration, we continue to carry out MD simulation for adopting the snapshot which show ideal mean square internal distance (MSID) $< R(n)^2 >$ represented as $\frac{1}{M/N_b - n} \sum_{j=0}^{N-1} \sum_{i=1}^{M/N_b - n} (\mathbf{R}_{i+(M/N_b)j} - \mathbf{R}_{i+(M/N_b)j+n})^2$.

Ideal MSID means the MSID of CG chains generated from fully equilibrated microscopic configurations.

A snapshot is captured in each τ_{blob} .

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Building and Testing

Explanation of installation:

- <https://github.com/espressopp/espressopp>

After installing this module, it can be tested according to the README file found under the following link:

- https://github.com/espressopp/espressopp/tree/master/examples/hierarchical_strategy_for_one-component/

Source Code

This module has been merged into ESPResSo++:

- <https://github.com/espressopp/espressopp/pull/213>

References

These modules have resulted in the final overarching module that captures the goal of the pilot project:

Hierarchical Strategy for Simple One-Component Polymer Melts

Software Technical Information

The information in this section describes ESPResSo++ as a whole. Information specific to the additions in this module are in subsequent sections.

Languages: Python (2.7) and C++

Documentation Tools: Sphinx and Doxygen

Application Documentation: <http://espressopp.github.io/>

Relevant Training Material: <https://github.com/espressopp/espressopp/tree/master/examples>

Licence GNU General Public License

Author Hideki Kobayashi

- *Purpose of Module*
- *Background Information*
- *Testing*
- *Source Code*
- *References*

The module is an implementation of the existing hierarchical strategy¹ for the equilibration of simple one-component polymer melts in ESPResSO++.

Purpose of Module

To study the properties of polymer melts by numerical simulations, equilibrated configurations must be prepared. However, the relaxation time for high molecular weight polymer melts is huge and increases, according to reptation theory, with the third power of the molecular weight. Hence, an effective method for decreasing the equilibration time is required. The hierarchical strategy pioneered in Ref.¹ is a particularly suitable way to do this. The present module provides an integration of that method into the package ESPResSO++.

To decrease the relaxation time, microscopic monomers are coarse-grained by mapping each subchain with N_b monomers onto a soft blob. A polymer chain, originally consisting of N monomers, is replaced by a coarse-grained (CG) chain consisting of N/N_b soft blobs linked by a harmonic bond potential, $V_{bond} = 3k_B T d^2 / 2b_{CG}^2$, and an angular bond-bending potential $V_{bend} = k_B T k_{bend} (1 + \cos(\theta)) / 2$. Here d is the distance and θ is the angle between consecutive bonds. The interactions between non-bonded soft blobs are taken into account by a repulsive pair potential $V_{nb} = k_B T \epsilon U_G(r_{ij})$. Here r_{ij} is the center-to-center distance between the two blobs, $U_G(r_{ij})$ is a Gaussian function with variance $\bar{\sigma}^2 = \sigma_i^2 + \sigma_j^2$ and σ_i is the gyration radius of blob number i . The gyration radius σ is in turn fluctuating. This fluctuation is controlled by the potential $V_{sphere} = k_B T (a_1 N_b^3 \sigma^{-6} + a_2 N_b^{-1} \sigma^2 + a_3 \sigma^{-3})$.

¹ : <http://pubs.acs.org/doi/abs/10.1021/mz5000015>

After equilibrating a configuration at very coarse resolution, each CG polymer chain is replaced with a more fine-grained (FG) chain. In this procedure, a CG blob is divided into several FG blobs. The center of mass (COM) of the FG blobs coincides with the position of the CG blob's center, and is being kept fixed during the relaxation of the local conformation of the FG monomers within the CG blob.

To develop this module, the following classes have been implemented or modified (and may have been described in more detail elsewhere):

- A `VSpherePair` class for calculating $V_{nb} = k_B T \epsilon U_G(r_{ij})$
- A `LangevinThermostatOnRadius` class for simulating the fluctuations of the radii of the blobs
- A `VSphereSelf` class for calculating $V_{sphere} = k_B T (a_1 N_b^3 \sigma^{-6} + a_2 N_b^{-1} \sigma^2 + a_3 \sigma^{-3})$
- A `FixedLocalTupleList` class for storing the N-tuple of particles consisting of both real and virtual particles
- A `ConstrainCOM` class for conserving the COM of N FG blobs with the CG blob

Background Information

The implementation of this module is based on ESPResSO++. You can learn about ESPResSO++ from the following links:

- ESPResSO++ documentation: <http://espressopp.github.io/ESPResSo++.pdf>
- ESPResSO++ source code: <https://github.com/espressopp/espressopp>

Testing

Explanation of installation of ESPResSO++ can be found at:

- <https://github.com/espressopp/espressopp>

After installing this module, an example can be run from *hierarchical_strategy_for_one-component* subdirectory of the *examples* folder using the *run_example* script to be found there.

- https://github.com/hidekb/espressopp/tree/hierarchical-strategy/examples/hierarchical-strategy/simple_one-component

Source Code

This module was merged into ESPResSo++ in the Pull Request:

- <https://github.com/espressopp/espressopp/pull/213>

References

4.3.3 ParaDiS

The following modules connected to the ParaDiS code have been produced so far:

Software Technical Information

Name ParaDiS_Precipitate_GC

Language C++

Licence This is patch based on the ParaDiS version 2.5.1. The additions are GPL.

Documentation Tool Sphinx

Application Documentation <http://paradis.stanford.edu/>

Relevant Training Material https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs/tree/master/test_run

ParaDiS with precipitates

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Discrete dislocation dynamics (DDD) simulations usually treat with “pure” crystals and dislocations in them. In reality, there is a need to look at more complicated scenarios of impurities interacting with the dislocations and their motion. Effects on a single atom / vacancy level may be incorporated by renormalizing the dislocation mobility but in many cases the dislocation dynamics is changed by the presence of clusters or precipitates, that act as local pinning centers. The consequences of the impurities are multiple: the yield stress is changed, and in general the plastic deformation process is greatly affected. Simulating these by DDD allows to look at a large number of issues from materials design to controlling the yield stress and may be done in a multiscale manner by computing the dislocation-precipitate interactions from microscopic simulations or by coarse-graining the DDD results for the stress-strain curves on the mesoscopic scale to more macroscopic Finite Element Method (the material model therein). This module provides an extension of the ParaDiS DDD code (LLNL, <http://paradis.stanford.edu/>) where dislocation/precipitate interactions are included.

Purpose of Module

The method is based on extending a recent version of ParaDiS to handle the presence of pinning centers. These work as localized Gaussian potentials that interact with the near-by dislocations (see A. Lehtinen et al. Phys. Rev. E 93, 013309 (2016)). The “disorder field” is given as an input where the locations of the precipitates are given in 3D, and the interactions are parametrized by the impurity strength (which may vary from precipitate to another) and the range of the Gaussian potential (which also may vary). The dislocation dynamics is handled as in ParaDiS in general with an additional force terms that accounts for each dislocation segment for the nearby impurities (a cut-off is applied in the force).

The Module thus allows to study various precipitate fields (density, geometry, strength, interaction range) as desired. In a typical ParaDiS simulation one does a simulation of the response of a dislocation system to a strain/stress protocol. The starting point is a dislocation system, which has been obtained from relaxing a random or patterned configuration under zero external stress until the evolution becomes negligible. In the presence of impurities the customary approach is to do two relaxation steps: first follow the relaxation of dislocation configuration, then add the disorder field to that and re-relax.

Background Information

The module version is built on the ParaDiS version 2.5.1 which can be obtained from <http://paradis.stanford.edu/> and following the steps outlined there for obtaining the code.

Building and Testing

The version offered is built exactly like the normal ParaDiS; the makefiles etc. are for the local CSC system and should be modified for the local environment. To test the ParaDiS build, an example case of a constant strain rate simulation of BCC iron with precipitates is included. The input of the test simulation is in file ParaDiS_test.ctrl, where the output directories and the used number of computational domains need to be defined. The initial dislocation structure is contained in the ParaDiS_test.data as usual and the structure of the file is identical to the files used by default ParaDiS. In addition, the simulation has ~8500 precipitates which are included in the ParaDiS_test.pdata file. This .pdata file has first some domain variables defined similar to .data file, and then the precipitates. These are presented one precipitate per line, and the data columns are as follows: [precipitate tag, position x, y and z, impurity strength, interaction radius, boolean], where the boolean states if the precipitate is active.

The used printing options defined in .ctrl file can be modified. Here, examples of the output property data and restart files are included in run_output folder and the file called ParaDiS_test.out contains the standard output of the test when the simulation system is run for ~1.5e-9 seconds. The restart files are written similarly as in unmodified ParaDiS, except that now the precipitates are also included in corresponding rsXXXX.pdata files. In addition to the property files produced by original ParaDiS, the modified ParaDiS writes also files allepsdot and avalanche. Allepsdot contains columns [simulations time, strain rate tensor element 11, stress tensor element 11, ...], and avalanche columns [time, average velocity, plastic strain, applied stress, total dislocation length, integrated strain rate] where the average velocity is calculated as a segment weighted average velocity of dislocations.

The test case is illustrated with three files: ParaDiS_test.out, and two plots, which are: aver_velocity_time.pdf (the resulting average dislocation velocity during the run) and stress_plastic_strain.pdf (yield strain versus applied stress during the run).

Source Code

Due to licensing reasons, only the difference between ParaDiS version 2.5.1 files and modified files are submitted, and these files can be found in https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs.git.

Software Technical Information

Name ParaDiS_Precipitate_HPC

Language C++

Licence This is patch based on the ParaDiS version 2.5.1. The additions are GPL.

Documentation Tool Sphinx

Application Documentation <http://paradis.stanford.edu/>

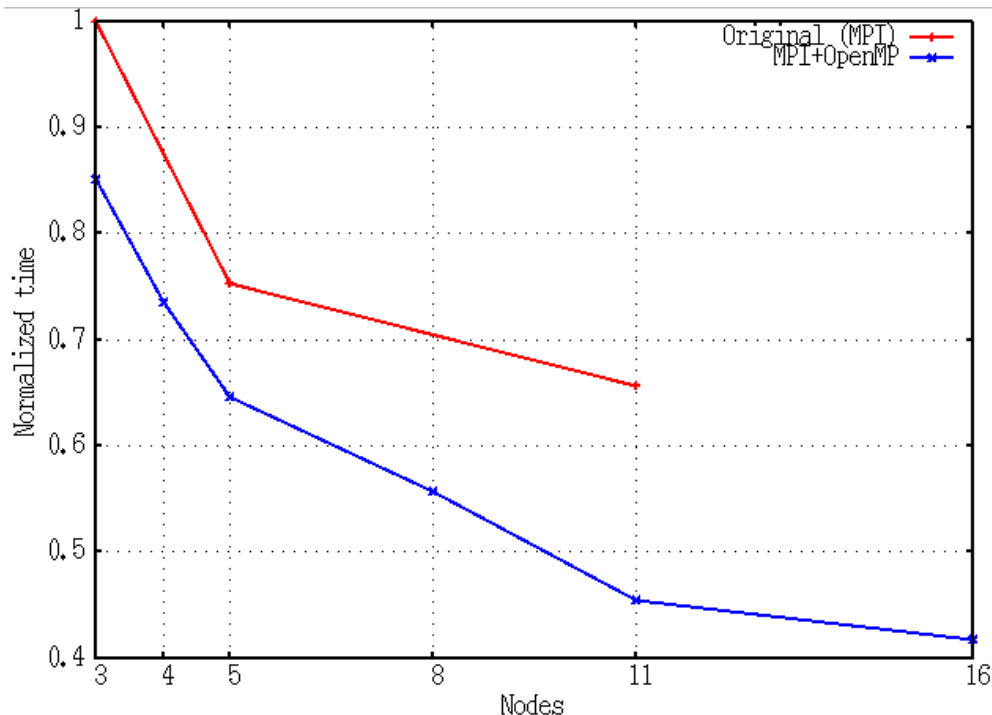
Relevant Training Material https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs/tree/master/test_run

ParaDiS with precipitates optimized to HPC environment

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Discrete dislocation dynamics (DDD) simulations usually treat with “pure” crystals and dislocations in them. In reality, there is a need to look at more complicated scenarios of impurities interacting with the dislocations and their motion. Effects on a single atom / vacancy level may be incorporated by renormalizing the dislocation mobility but in many cases the dislocation dynamics is changed by the presence of clusters or precipitates, that act as local pinning centers. The consequences of the impurities are multiple: the yield stress is changed, and in general the plastic deformation process is greatly affected. Simulating these by DDD allows to look at a large number of issues from materials design to controlling the yield stress and may be done in a multiscale manner by computing the dislocation-precipitate interactions from microscopic simulations or by coarse-graining the DDD results for the stress-strain curves on the mesoscopic scale to more macroscopic Finite Element Method (the material model therein).

This module provides an extension of the ParaDIS DDD code (LLNL, <http://paradis.stanford.edu/>) where dislocation/precipitate interactions are included. The extension is for an HPC environment, in which the original code has been optimized for the Cray XC40 cluster at CSC in Finland in mind. Vectorizing better some subroutines and using threads better by a combination of MPI and Open MP allows for large-scale jobs speed-ups of a factor of 1.5 and allows to use more computational nodes than what is reasonable with the original version, so that the production time is speeded up by a factor of two. This is visualized in the figure below, where the normalized time is plotted versus the number of nodes for both the original and the HPC optimized codes.



Purpose of Module

The method is based on extending a recent version of ParaDIS to handle the presence of pinning centers. These work as localized Gaussian potentials that interact with the near-by dislocations (see A. Lehtinen et al. Phys. Rev. E 93, 013309 (2016)). The “disorder field” is given as an input where the locations of the precipitates are given in 3D, and

the interactions are parametrized by the impurity strength (which may vary from precipitate to another) and the range of the Gaussian potential (which also may vary). The dislocation dynamics is handled as in ParaDiS in general with an additional force terms that accounts for each dislocation segment for the nearby impurities (a cut-off is applied in the force).

The Module thus allows to study various precipitate fields (density, geometry, strength, interaction range) as desired. In a typical ParaDiS simulation one does a simulation of the response of a dislocation system to a strain/stress protocol. The starting point is a dislocation system, which has been obtained from relaxing a random or patterned configuration under zero external stress until the evolution becomes negligible. In the presence of impurities the customary approach is to do two relaxation steps: first follow the relaxation of dislocation configuration, then add the disorder field to that and re-relax. In the current version apart from HPC-related parallelization-relevant steps the subroutines SegSegForce (segment-to-segment force calculation) and FMSigma2Core2 (force multipole expansion) are well vectorized, and the code now also uses better multiple threads in their context.

Background Information

The module version is built on the ParaDiS version 2.5.1 which can be obtained from <http://paradis.stanford.edu/> and following the steps outlined there for obtaining the code.

Building and Testing

The version offered is built exactly like the normal ParaDiS; the makefiles etc. are for the local CSC system and should be modified for the local environment. To test the ParaDiS build, an example case of a constant strain rate simulation of BCC iron with precipitates is included. The input of the test simulation is in file ParaDiS_test.ctrl, where the output directories and the used number of computational domains need to be defined. The initial dislocation structure is contained in the ParaDiS_test.data as usual and the structure of the file is identical to the files used by default ParaDiS. In addition, the simulation has ~8500 precipitates which are included in the ParaDiS_test.pdata file. This .pdata file has first some domain variables defined similar to .data file, and then the precipitates. These are presented one precipitate per line, and the data columns are as follows: [precipitate tag, position x, y and z, impurity strength, interaction radius, boolean], where the boolean states if the precipitate is active.

The used printing options defined in .ctrl file can be modified. Here, examples of the output property data and restart files are included in run_output folder and the file called ParaDiS_test.out contains the standard output of the test when the simulation system is run for ~1.5e-9 seconds. The restart files are written similarly as in unmodified ParaDiS, except that now the precipitates are also included in corresponding rsXXXX.pdata files. In addition to the property files produced by original ParaDiS, the modified ParaDiS writes also files allepsdot and avalanche. Allepsdot contains columns [simulations time, strain rate tensor element 11, stress tensor element 11, ...], and avalanche columns [time, average velocity, plastic strain, applied stress, total dislocation length, integrated strain rate] where the average velocity is calculated as a segment weighted average velocity of dislocations.

The test case is illustrated with three files: ParaDiS_test.out, and two plots, which are: aver_velocity_time.pdf (the resulting average dislocation velocity during the run) and stress_plastic_strain.pdf (yield strain versus applied stress during the run).

Source Code

Due to licensing reasons, only the difference between ParaDiS version 2.5.1 files and modified files are submitted, and these files can be found in https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs.git.

Software Technical Information

Name ParaDiS_Precipitate_GC optimized for AMD Zen2

Language C++

Licence Extension is based on ParaDiS version 2.5.1. The additions in the extension are GPL.

Documentation Tool Sphinx

Application Documentation <http://paradis.stanford.edu/>

Relevant Training Material https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs/tree/master/test_run

Software Module Developed by Phuong Nguyen (phuong.nguyen@csc.fi)

ParaDiS with precipitates optimized for AMD Zen2

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Discrete dislocation dynamics (DDD) simulations usually treat with “pure” crystals and dislocations in them. An extension of the ParaDiS DDD code (LLNL, <http://paradis.stanford.edu/>) that includes dislocation/precipitate interactions has been developed (E-CAM module: [ParaDiS with precipitates](#)).

This module provides a guide for optimal porting of the [ParaDiS with precipitates](#) to the AMD Rome CPUs, in preparation for the [Mahti supercomputer](#) service at CSC, Finland. Mahti is an Atos BullSequana XH2000 system consisting of 1404 nodes each with two 64-core AMD Zen2 CPUs (AMD EPYC 7H42, 2.6GHz). Since Mahti is not ready for general access at this moment, the module was prepared based on a single testing node which has 2 AMD EPYC 7742 @2.25GHz (128 cores in total).

By choosing a suitable compiler and compiler optimization flags, the application works more efficiently on the target platform. On the testing node, Intel compilers with either AVX or AVX2 vector sets gives the best performance for *ParaDiS with precipitates*. Alternatively, GCC compilers with AVX2 vector support is competitive with the Intel compilers.

Purpose of Module

This module helps to run simulations of the *ParaDiS with precipitates* more efficiently. By using a suitable set of optimization flags for compilers, especially the one determining vectorization type, the best library routines can be chosen.

Background Information

The module is based on the ParaDiS (<http://paradis.stanford.edu/>) extension [ParaDiS with precipitates](#).

Building and Testing

Build instructions for [ParaDiS with precipitates](#) are provided with the extension.

Different compilers and compiler options were tested to find the most optimal ones for the Zen2 architecture. Figure 1 (below) shows a comparison of normalized running times between different vectorization extensions and compilers. On the testing platform, Intel compilers with either AVX or AVX2 helps the application to achieve good performance. Alternatively, GCC compilers with AVX2 can be used to obtain the same performance as the Intel ones.

Table 1 presents a comparison of different optimization flags for the Intel and GCC compilers. For the Intel compilers, the optimal performance is reached with the compiler options: `-O3 -mavx2` or `-O3 -mavx`. For the GCC compilers, `-O2 -march=znver2 -pipe -fomit-frame-pointer -ftree-vectorize` compiler options help the application to achieve good performance.

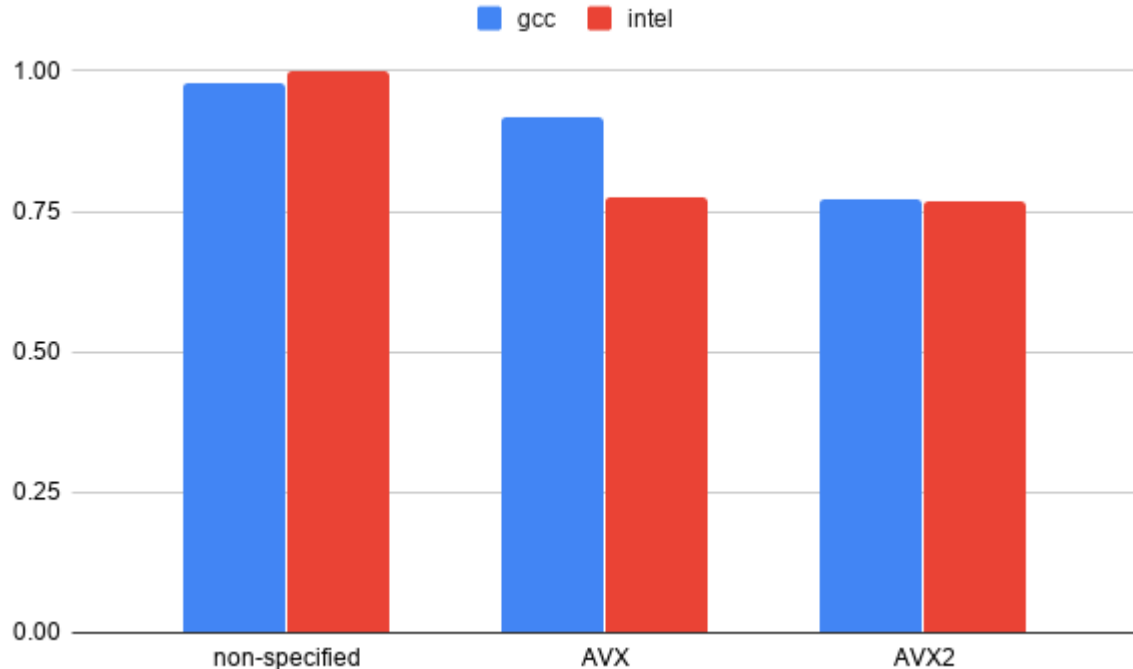


Fig. 4.1: Figure 1: Comparison of normalized times between different compilers and vectorization extensions (smaller is better)

Table 1: Comparison between different optimization flag options

| Compilers | Flags | Time (s) |
|-----------|---|----------|
| Intel | <code>-O2 -axCORE-AVX2</code> | 328 |
| | <code>-O2 -axHASWELL</code> | 360 |
| | <code>-O2 -mavx2</code> | 309 |
| | <code>-O3 -mavx2</code> | 295 |
| | <code>-O3 -mavx</code> | 298 |
| | <code>-Ofast -mavx2</code> | 301 |
| | <code>-O3 -mavx2 -funroll-all-loops</code> | 317 |
| | <code>-O3 -mavx2 -funroll-all-loops -ftree-vectorize</code> | 317 |
| GCC | <code>-O2 -march=znver1 -pipe -fomit-frame-pointer -ftree-vectorize</code> | 352 |
| | <code>-O2 -march=znver2 -pipe -fomit-frame-pointer -ftree-vectorize</code> | 296 |
| | <code>-O3 -march=znver2</code> | 382 |
| | <code>-O2 -march=haswell -pipe -fomit-frame-pointer -ftree-vectorize</code> | 352 |

* The input case in these tests are different to the ones at [ParaDiS](#) with precipitates optimized for Puhti .

In the *ParaDiS with precipitates* optimized to HPC environment, it's written that using multiple threads through a hybrid OpenMP and MPI model speeds up the calculation up to a factor of 1.5, especially for large-scale simulations. However, this combination did not give an advantage of performance on the Zen2 testing machine. Thus, using a single thread for each MPI process is recommended.

Source Code

Source code modifications for the extension *ParaDiS with precipitates* are available here: https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs.git.

Software Technical Information

Name ParaDiS_Precipitate_GC optimized for Puhti

Language C++

Licence Extension is based on ParaDIS version 2.5.1. The additions in the extension are GPL.

Documentation Tool Sphinx

Application Documentation <http://paradis.stanford.edu/>

Relevant Training Material https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs/tree/master/test_run

Software Module Developed by Phuong Nguyen (phuong.nguyen@csc.fi)

ParaDiS with precipitates optimized for Puhti

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

Discrete dislocation dynamics (DDD) simulations usually treat with “pure” crystals and dislocations in them. An extension of the ParaDIS DDD code (LLNL, <http://paradis.stanford.edu/>) that includes dislocation/precipitate interactions has been developed (E-CAM module: *ParaDiS with precipitates*).

This module provides a guide for optimal porting of the *ParaDiS with precipitates* to the *Puhti supercomputer* at CSC, Finland. Puhti is an Atos BullSequana X400 system consisting of 682 nodes each with two 20-core Intel Cascade Lake CPUs (Intel Xeon Gold 6230, 2.1GHz). By choosing a suitable compiler and compiler optimization flags, the application works more efficiently on the target platform. On Puhti, Intel compilers with AVX-512 vector sets gives the best performance for *ParaDiS with precipitates*.

Purpose of Module

This module helps to run simulations of the *ParaDiS with precipitates* more efficiently. By using a suitable set of optimization flags for compilers, especially the one determining vectorization type, the best library routines can be chosen.

Background Information

The module is based on the ParaDiS (<http://paradis.stanford.edu/>) extension `ParaDiS with precipitates`.

Building and Testing

Build instructions for `ParaDiS with precipitates` are provided with the extension.

Different compilers and compiler options were tested to find the most optimal ones for the Puhti supercomputer. Figure 1 (below) shows a comparison of normalized running times between different vectorization extensions and compilers. On the Puhti platform, Intel compilers with AVX-512 helps the application to achieve the best performance.

Table 1 presents a comparison of different optimization flags for the Intel compiler. The optimal performance is reached with the compiler options: `-O2 -axCASCADELAKE`.

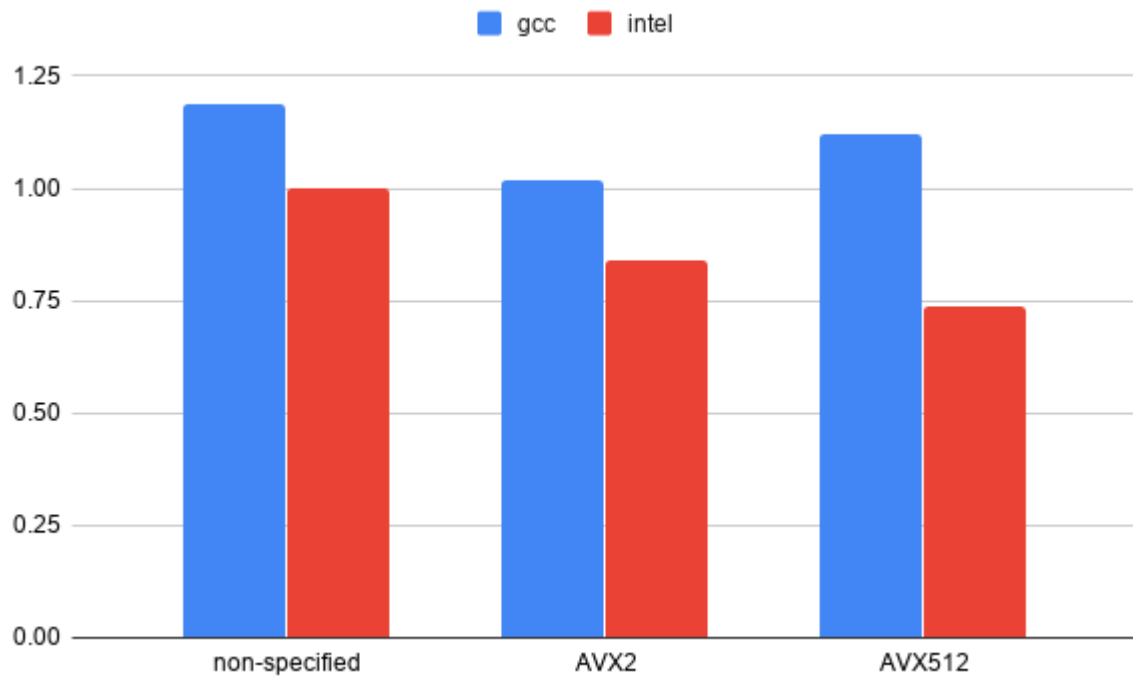


Fig. 4.2: Figure 1: Comparison of normalized times between different compilers and vectorization extensions (smaller is better)

Table 1: Comparison between different optimization flag options

| Flags | Time (s) |
|---------------------------------------|----------|
| -O2 -axCORE-AVX2 | 273 |
| -O2 -axCORE_AVX2 -mtune=cascadelake | 338 |
| -O2 -axCORE-AVX2 -mtune=broadwell | 298 |
| -O2 -xCORE-AVX2 -axCASCADELAKE | 254 |
| -O2 -axCOMMON-AVX512 | 321 |
| -O2 -axCORE-AVX512 | 249 |
| -O2 -axCASCADELAKE | 240 |
| -O2 -axCASCADELAKE -mtune=cascadelake | 280 |
| -O3 -axCASCADELAKE | 270 |

In the *ParaDiS with precipitates optimized to HPC environment*, it's written that using multi threads through the hybrid OpenMP and MPI model speeds up the calculation up a factor of 1.5, especially for the large-scale simulations. However, this combination did not give an advantage of performance on the Puhti. Thus, using single thread for each MPI process is recommended.

Source Code

Source code modifications for the extension *ParaDiS with precipitates* are available here: https://version.aalto.fi/gitlab/csm_open/paradis_version_diffs.git.

4.3.4 Cubble

The following modules connected to the Cubble code, a mesoscale CUDA/C++ simulator of foams, have been produced so far:

Software Technical Information

Name Cubble_coarsening_static

Language C++/CUDA

Licence This software will be released under the GPL licence.

Documentation Tool Sphinx

Application Documentation https://version.aalto.fi/gitlab/lankinj5/cuda_bubble

Relevant Training Material https://version.aalto.fi/gitlab/lankinj5/cuda_bubble/wikis/home

Cubble: Static foam coarsening simulator using c++/CUDA

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The mechanics of foams exhibit dynamical behavior familiar from other jammed materials, such as granular matter. These are so called yield stress fluids i.e. their flow requires external stress that exceeds a particular value, called the yield stress. On the other hand, foams differ from other materials by their internal structure development, coarsening, where while the gas concentration of the foam remains constant, the gas diffuses between the bubbles. The larger bubbles grow at the expense of the smaller ones.

This module provides a c++/CUDA implementation of a foam coarsening simulator (<https://journals.aps.org/pre/abstract/10.1103/PhysRevE.98.012607>) designed from the bottom to be run in a GPU environment.

Purpose of Module

The goal was set to be able to run simulations involving up to one million bubbles reaching a scaling state in systems with non-periodic boundary conditions in three dimensions, an undoable task for even the most efficient single core CPU implementation. The Cubble code demonstrates the power of efficiently used GPU code, and provides a model implementation strategy for mesoscale DEM simulators.

Background Information

The code runs as a standalone simulation on a cluster (`triton.aalto.fi`) environment. It is developed mostly on NVidia's Tesla P100 and V100 GPUs.

Building and Testing

The binary is built using a build automation tool Make. The dimensionality of the simulation is controlled from within the `makefile`. Each `make` target is built into a separate directory: `final`, `default` or `debug`. Each of these directories has its own `makefile` and all the targets are built/cleaned in the same way:

```
make
make clean
```

- `final` target is the one that should be run when doing simulations. It's the fastest and most optimized.
- `default` target is built with `-O2` flag so it's quite fast, but some internal debug capabilities are still on and it's significantly slower than the `final` target. Mostly for testing some new capabilities.
- `debug` is built with `-O0` and debug capabilities, only meant for debugging and therefore it is very slow.

In addition to the options above, there are some extra parameters in the `makefile` which can be used to e.g. turn profiling on/off.

The program can be run by typing

```
make run
```

or by manually writing the path to the executable and the io files, e.g.

```
final/bin/cubble input_parameters.json output_parameters.json
```

We include a set of reference parameters in `input_parameters.json`

The program runs until a certain amount of bubbles is left. After this, the program writes one final data file and returns. The parameter that controls the amount of bubbles (called `MinNumBubbles`) should always be larger than the number of bubbles in one cell multiplied by 3^{NumDim} . In other words, if the number of bubbles in a cell is 32 and the dimensionality of the program is 2 (2D simulation), then the minimum number of bubbles should be larger

than $32 * 3^2 = 32 * 3 * 3 = 288$. For 3D this would be 864. 300 and 900 are nice round numbers for `MinNumBubbles`.

The reason for this is that the neighbor search is done in a manner that assumes at least 3 cells in each dimension. If there are less than 3 cells per dimension, some cells are searched through more than once, leading to bubbles having the same bubble as a neighbor multiple times. The implementation should and could be improved to circumvent this, but “in the mean time” just follow the above rule.

Source Code

The source code is freely available for download at *Cubble sources* <<https://github.com/KJLankinen/cubble>>. This is the multiGPU version of the code running simultaneously on several GPUs to allow for simulations beyond 10 million bubbles. The module specifically refers to the commit 3216d46c7e523e7885d12607197390496b379597.

Software Technical Information

Name `Cubble_coarsening_flow`

Language C++/CUDA

Licence This software will be released under the GPL licence.

Documentation Tool Sphinx

Application Documentation https://version.aalto.fi/gitlab/lankinj5/cuda_bubble

Relevant Training Material https://version.aalto.fi/gitlab/lankinj5/cuda_bubble/wikis/home

Cubble: Flowing foam coarsening simulator using c++/CUDA

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The mechanics of foams exhibit dynamical behavior familiar from other jammed materials, such as granular matter. These are so called yield stress fluids i.e. their flow requires external stress that exceeds a particular value, called the yield stress. On the other hand, foams differ from other materials by their internal structure development, coarsening, where while the gas concentration of the foam remains constant, the gas diffuses between the bubbles. The larger bubbles grow at the expense of the smaller ones.

This module provides the capability to run dynamical coarsening on flowing systems to the *Cubble: Static foam coarsening simulator using c++/CUDA* module.

Purpose of Module

This module implements background flow to the Cubble simulator. Implementation includes flow by enforcing bubble velocity to all bubbles inside a defined regime. Implementation includes viscous dissipation at the boundaries.

Background Information

The code runs as a standalone simulation on a cluster (`triton.aalto.fi`) environment. It is developed mostly on NVidia's Tesla P100 and V100 GPUs.

Building and Testing

The binary is built using a build automation tool Make. The dimensionality of the simulation is controlled from within the `makefile`. Each make target is built into a separate directory: `final`, `default` or `debug`. Each of these directories has its own `makefile` and all the targets are built/cleaned in the same way:

```
make
make clean
```

- `final` target is the one that should be run when doing simulations. It's the fastest and most optimized.
- `default` target is built with `-O2` flag so it's quite fast, but some internal debug capabilities are still on and it's significantly slower than the `final` target. Mostly for testing some new capabilities.
- `debug` is built with `-O0` and debug capabilities, only meant for debugging and therefore it is very slow.

In addition to the options above, there are some extra parameters in the `makefile` which can be used to e.g. turn profiling on/off.

The program can be run by typing

```
make run
```

or by manually writing the path to the executable and the io files, e.g.

```
final/bin/cubble input_parameters.json output_parameters.json
```

We include a set of reference parameters in `input_parameters.json`

The program runs until a certain amount of bubbles is left. After this, the program writes one final data file and returns. The parameter that controls the amount of bubbles (called `MinNumBubbles`) should always be larger than the number of bubbles in one cell multiplied by 3^{NumDim} . In other words, if the number of bubbles in a cell is 32 and the dimensionality of the program is 2 (2D simulation), then the minimum number of bubbles should be larger than $32 * 3^2 = 32 * 3 * 3 = 288$. For 3D this would be 864. 300 and 900 are nice round numbers for `MinNumBubbles`.

The reason for this is that the neighbor search is done in a manner that assumes at least 3 cells in each dimension. If there are less than 3 cells per dimension, some cells are searched through more than once, leading to bubbles having the same bubble as a neighbor multiple times. The implementation should and could be improved to circumvent this, but "in the mean time" just follow the above rule.

Source Code

The source code is freely available for download at *Cubble sources* <<https://github.com/KJLankinen/cubble>> and the module refers to the repository version with commit 0830686ac628d884d23666560ddded5361b7f606.

Software Technical Information

Name Cubble_HIP

Language C++/CUDA and HIP

Licence This software will be released under the GPL licence.

Documentation Tool Sphinx

Application Documentation https://version.aalto.fi/gitlab/lankinj5/cuda_bubble

Relevant Training Material https://version.aalto.fi/gitlab/lankinj5/cuda_bubble/wikis/home

Cubble: Hip implementation

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The mechanics of foams exhibit dynamical behavior familiar from other jammed materials, such as granular matter. These are so called yield stress fluids i.e. their flow requires external stress that exceeds a particular value, called the yield stress. On the other hand, foams differ from other materials by their internal structure development, coarsening, where while the gas concentration of the foam remains constant, the gas diffuses between the bubbles. The larger bubbles grow at the expense of the smaller ones.

This module provides a recipe and makefile for converting the c++/CUDA implementation of a foam coarsening simulator (<https://journals.aps.org/pre/abstract/10.1103/PhysRevE.98.012607>) to instead use HIP to allow it to be run on both AMD and Nvidia GPUs.

Purpose of Module

The goal was set to be able to run simulations involving up to one million bubbles reaching a scaling state in systems with non-periodic boundary conditions in three dimensions, an undoable task for even the most efficient single core CPU implementation. The Cubble code demonstrates the power of efficiently used GPU code, and provides a model implementation strategy for mesoscale DEM simulators.

Background Information

The converted code runs on single MI50 and Radeon VII GPUs.

Building and Testing

Converting the Cubble code to use HIP instead of Cuda enables it to run on AMD GPUs, in addition to Nvidia GPUs. The conversion process is easy.

First the source code needs to be converted from Cuda to HIP, this is done by converting everything in the src directory using the `hipify-perl` script. All `.cu` files are changed to `cpp` files and `.cuh` changed to `.h`, we then need to update any include statements to reflect this.

When converting `Util.h` the converter will emit the following warnings


```
warning: old/Util.h:#25 : cubble::cudaCallAndLog((call), #call, __FILE__, __LINE__)
warning: old/Util.h:#27 : cubble::cudaCallAndThrow((call), #call, __FILE__, __LINE__
↪)
warning: old/Util.h:#125 : inline bool cudaCallAndLog(hipError_t result, const char_
↪*callStr,
warning: old/Util.h:#137 : inline void cudaCallAndThrow(hipError_t result, const char_
↪*callStr,
```

This is due to the cubble program using the same naming scheme as cuda for some functions, i.e the name is cuda-Something and the converter is warning that it was uable to convert them, however in this case these are not actual cuda calls and it should not covert these, so it is safe to ignore the warnings.

The cubble code also uses the NVTX library for better profiling, this library does not work on AMD hardware and while comparable functionality exists this will not be automatically converted. The Cubble program will work without NVTX so in this case we just remove the inclusion of `nvToolsExt.h` and not enable profiling when compiling the Cubble program.

On some systems we may need to add `-DENABLE_HIP_PROFILE=0` to the compilation flags to suppress errors about some profiling headers not being found.

After this we should have a version of the code that can be compiled with the included `makefile`. Unfortunately this version will not run with the current version of HIP, tested with version 3.1. The program will fail with not finding symbols in `hipGetSymbolAddress` calls or complaining it does not have device functions for certain calls. The cubble code places the majority of its code in the cubble namespace, unfortunately currently the HIP compiler struggles with device symbols and functions being in a namespace. The easiest solution in this case is to move all the code out of the cubble namespace, this will give you a code that will run on AMD hardware. It is likely this will improve in the future and this step will no longer be needed.

Source Code

The source code used as a base for the conversion is freely available for download in *Cubble sources* <<https://github.com/KJLankinen/cubble>>. In addition the modified `makefile` is included in this repository: `makefile`

4.3.5 GC-AdResS

This modules are connected to the Adaptive Resolution Simulation implementation in GROMACS.

Software Technical Information

Name GC-AdResS: Abrupt scheme

Language Implemented in GROMACS version 5.1.5

Licence MD Simulation:

See GROMACS web page: <http://www.gromacs.org>

Analysis tools and thermodynamic force calculation:

see VOTCA web page: <http://www.votca.org/home>

Documentation Tool

Application Documentation

See GROMACS web page: <http://www.gromacs.org>

See VOCTA web page: <http://www.votca.org/Documentation>

Relevant Training Material

See GROMACS web page: <http://www.gromacs.org>

See VOCTA web page: <http://www.votca.org/tutorials>

Abrupt GC-AdResS: A new and more general implementation

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The original idea of our proposal: to work on a general implementation of AdResS in class. MD packages. The current implementation of GC- AdResS in GROMACS has several performance problems. This module presents a very straight forward way to implement a new partitioning scheme, which solves two problems which affect the performance, the neighborlist search and the generic force kernel. Furthermore, we update the implementation to address this in a way that decouples the method directly from the core of any MD code, which does not hinder the performance and makes the scheme hardware independent. Theory, application and tests see <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>.

Purpose of Module

The main performance loss of AdResS simulations in GROMACS is in the neighboring list search and the generic serial force kernel, linking the atomistic (AT) and coarse grained (CG) forces together via a smooth weighting function. Thus, to get rid of the bottleneck with respect to performance and a hindrance regarding the easy/general implementation into other codes and thus get rid of the not optimized force kernel used in GROMACS we had to change the neighborlist search. This lead to a considerable speed up of the code. Furthermore it decouples the method directly from the core of any MD code, which does not hinder the performance and makes the scheme hardware independent. For the theory, application and tests see <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>.

Background Information

This module presents a very straight forward way to implement a new partitioning scheme. And this solves two problems which affect the performance, the neighborlist search and the generic force kernel.

In GROMACS the neighbor list is put together and organized in the file 'ns.c'. In GROMACS 5.1 there are two functions which basically sort the incoming particles into the different neighbor list. In its current official GROMACS release everything other than CG (with $w_i = w_j = 1$) or AT (with $w_i = w_j = 0$) is sorted into the neighbor lists. Any other particles are sorted into a special neighbor list only for AdResS.

We now changed this neighborlist sorting into: Everything is taken into account other than: (AT and ($w_i = 0$ or $w_j = 0$)) or (CG and ($w_i \geq 0$ and $w_j \geq 0$)). This leads to 5 distinct interactions: (1) AT-AT in the atomistic region, (2) CG-CG in the CG region, (3) AT-AT between particles in the hybrid region, (4) AT-AT between particles of the atomistic region with the hybrid region and (5) CG-CG between particles of the CG region with the hybrid region. This if statement excludes the CG-CG interaction in the hybrid region.

This is a very straight forward way to implement a new partitioning scheme and utilize a constant weighting function. This solves both parts of the performance problem, the neighborlist search and the generic force kernel which can be simply switch off by switching to the standard interaction scheme implemented in GROMACS.

Building and Testing

We tested this new implementation on SPC water with varying system sizes. GROMACS is optimized especially for handling of bio-systems, i.e. GROMACS has the best performance in case of water simulations. We set up a couple of Abrupt GC-AdResS simulations ranging from small 6912 water molecules to 48k water molecules. We used a standard desktop machine (Intel Core i5-4590 CPU @ 3.30GHz x4) and run small 20 ps runs. We can see that our performance is much improved up to a factor of 2.5.

Here is a short manual on how to run the test and set up AdResS simulations in GROMACS:

- 1) mandatory requirement: a working full atomistic simulation (molecular configuration, force fields and optimal MD input parameter)
- 2) You need a very well converged NVT run, which can be used as starting point for the coarse grained (CG) and then later the AdResS simulations.
- 3) You have to generate a coarse grained (CG) potential. We use, for convenience, the inverse Boltzmann iteration provided in the VOTCA package (<http://www.votca.org/home>). The resulting tabulated CG potentials are used in the AdResS simulation. Alternatively you can use WCA potentials or standard Lennard-Jones potentials. The main requirement for the AdResS simulation is that the density in the CG region is the same as in the atomistic (AT) region.

NOTE: The method can be used with any potential, which preserves the correct density. If only a SPC/E CG potential is available it can be used for SPC/e water models as well as for a more advanced water model. It is possible to use a WCA potential, which is basically a Lennard-Jones potential. *And* it is possible to switch the CG ptential completely off. That will transform the CG region to a true thermodynamic reservoir with a non-interacting gas.

- 4) The next step is to create a double resolution configuration and adjust the dependencies (force field, topology, index file, GROMACS input file). Creating the configuration is straight forward (we use <http://www.votca.org/home>).

Example **from VOTCA**:

```
csg_map --top topol.tpr --cg cg_mapping_scheme --hybrid --trj conf.gro --out conf_
→hybrid.gro
```

Of course, if you want to use this configuration in a MD simulation you have to adjust the force field (see example file: *spc.adress.itp*). You have to define a virtual side:

```
[ virtual_sites3 ]
; Site from funct a d
; atom dependencies func      a      b
  4      1 2 3      1      0.05595E+00 0.05595E+00
```

The next step is to adjust the status of the CG particle in the topology file (in our example: *topol.top*) from *A* for *atom* to *V* as *virtual particle*. And of course insert the new force field.

```
#include "spc.adress.itp"
```

Then you have to generate an index file with the different energy groups. In this example, we have 2 groups (EXW and WCG, the name of the CG particle):

```
gmx make_ndx -f conf_hybrid.gro
> a WCG

Found 3456 atoms with name WCG

3 WCG                : 3456 atoms

> !3

Copied index group 3 'WCG'
Complemented group: 10368 atoms

4 !WCG               : 10368 atoms
> name 4 EXW
> q
```

The next step is to adjust the GROMACS input file. AdResS needs the Langevin dynamics, so you have to choose:

```
integrator = sd
```

Since the system is double resolution, meaning we have the atomistic details and the virtual particles, we have to define the energygroups:

```
; Selection of energy groups
energygrps = EXW WCG
energygrp_table = WCG WCG
```

GROMACS version 5.1.5 is using verlet as standard cutoff-scheme, so we have to change that to *group*:

```
; nblist update frequency
cutoff-scheme = group
```

Furthermore, in our simulations we use:

```
coulombtype = reaction-field
rcoulomb = 1.0
vdw-type = user
rvdw = 1.0
```

In case of local thermostat simulations (see <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>) we use:

```
coulombtype = reaction-field-zero
rcoulomb = 1.0
vdw-type = user
rvdw = 1.0
```

If you use the stochastic dynamics, we add the following entries to make sure we have only NVT and a thermalization via the Langevin dynamics.

```
; Temperature coupling
Tcoupl = no
Pcoupl = no
```

To switch the simulation to AdResS this is the key part. This starts the AdResS runs.

```
; AdResS parameters adress = yes ;no
```

Here you define the geometry of the atomistic region, either *sphere* (a spherical region anywhere in the simulation box) or *xsplitted* (a cuboid slice of the whole simulation box for the atomistic region, with the transition and coarse grained region on each side).

```
adress_type = sphere ;xsplitted sphere or constant
```

This defines the width of the atomistic region, starting from the given reference coordinate (keyword *adress_reference_coords*, by simply using: `tail conf_hybrid.gro | awk '(NF==3){print $1/2., $2/2., $3/2.}'`). In the older versions of AdResS, with a smooth coupling between AT and CG the width of the hybrid region width (*adress_hy_width*) was also defined. In the Abrupt AdResS setup it is not necessary any more, even if you put a number that region is counted (in the code) as AT.

```
adress_ex_width = 1.5
adress_hy_width = 1.5
adress_ex_forcecap = 2000
adress_interface_correction = thermoforce ;off
adress_site = com
adress_reference_coords = 3.7500 1.860355 1.860355
adress_tf_grp_names = WCG
adress_cg_grp_names = WCG
adress_do_hybridpairs = no
```

Another important aspect is the force capping. Abrupt AdResS works fine for small molecules like water, but for larger or more complex molecules the force capping is very important. We cap every force component (i.e. $f(x), f(y), f(z)$) acting on a particle and not the norm of the force, which reduces the computational time spend. This is described in another module.

adress_interface_correction defines if you use an external force to correct the density or not. In case of the old AdResS (smooth coupling) that correction simply refined the simulation, as the density difference was not significant. For the Abrupt AdResS, and the method development based on it, and more complex molecules (i.e. polymers) the thermodynamic force is essential. If it is not taken into account the risk to form interfaces between AT and CG is high. Also if particles coming too close (basically overlap) the run can crash. The role of the thermodynamic force, the force cap and the basic theory behind it see <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>. For this to work you must have a file e.g. in our example case: *tabletf_WCG.xvg* in the directory, otherwise you have to set:

```
adress_interface_correction = off
```

There is a number of properties you have to check. The first check is always the density and you see if the patch works from the density. If you have no thermodynamic force you have rather pronounced spikes in the density at the interfaces. If you have a converged thermodynamic force the density has to be within +/- 3% off from a comparable full atomistic simulation / experimental data. Then you need further properties to make sure you have an open system. The problem with the simulation is that an “artificial” interface is introduced and checks for the diffusion, the RDF’s... (full list see below) ensure that those regions mix and that you have proper particle transfer.

Source Code

The patch file for Abrupt GC-Adress is:

```
diff -ru /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/adress.c /home/mi/
↪ck69giso/gmx-515-hck/src/gromacs/mdlib/adress.c
--- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/adress.c      2016-07-13_
↪14:56:04.000000000 +0200
+++ /home/mi/ck69giso/gmx-515-hck/src/gromacs/mdlib/adress.c      2018-08-15_
↪12:39:32.000000000 +0200
```

(continues on next page)

(continued from previous page)

```

@@ -101,17 +101,17 @@
    return 0;
}
/* molecule is explicit */
- else if (sqr_dl < adressr*adressr)
+ else //if (sqr_dl < adressr*adressr)
{
    return 1;
}
/* hybrid region */
+ /* hybrid region
else
- {
-     dl = sqrt(sqr_dl);
-     tmp = cos((dl-adressr)*M_PI/2/adressw);
-     return tmp*tmp;
- }
+ {
+//     dl = sqrt(sqr_dl);
+//     tmp = cos((dl-adressr)*M_PI/2/adressw);
+     return 0.5;
+ } */
}

void
diff -ru /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/ns.c /home/mi/ck69giso/
↪gmx-515-hck/src/gromacs/mdlib/ns.c
--- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/ns.c      2016-07-13_
↪14:56:04.000000000 +0200
+++ /home/mi/ck69giso/gmx-515-hck/src/gromacs/mdlib/ns.c      2018-08-15 12:55:06.
↪000000000 +0200
@@ -264,10 +264,12 @@
    for (i = 0; i < fr->nnblists; i++)
    {
        nbl = &(fr->nblists[i]);
-
-     if ((fr->adress_type != eAdressOff) && (i >= fr->nnblists/2))
+/* chk */
+//     if ((fr->adress_type != eAdressOff) && (i >= fr->nnblists/2))
+     if ((fr->adress_type != eAdressOff))
        {
-         type = GMX_NBLIST_INTERACTION_ADDRESS;
+         /* type = GMX_NBLIST_INTERACTION_ADDRESS; */
+         type = GMX_NBLIST_INTERACTION_STANDARD;
        }
        init_nblast(log, &nbl->nlist_sr[eNL_VDWQQ], &nbl->nlist_lr[eNL_VDWQQ],
                    maxsr, maxlr, ivdw, ivdwmod, ielec, ielecmod, igeometry_def,
↪type, bElecAndVdwSwitchDiffers);
@@ -601,11 +603,14 @@
    int          *cginfo;
    int          *type, *typeB;
    real         *charge, *chargeB;
+   real         *wf;
    real         qi, qiB, qq, rlj;
    gmx_bool     bFreeEnergy, bFree, bFreeJ, bNotEx, *bPert;
    gmx_bool     bDoVdW_i, bDoCoul_i, bDoCoul_i_sol;
+   gmx_bool     b_hybrid;

```

(continues on next page)

(continued from previous page)

```

    int          iwater, jwater;
    t_nblast     *nlist;
+   gmh_bool     bEnergyGroupCG;

    /* Copy some pointers */
    cginfo = fr->cginfo;
@@ -614,6 +619,7 @@
    type        = md->typeA;
    typeB       = md->typeB;
    bPert       = md->bPerturbed;
+   wf         = md->wf;

    /* Get atom range */
    i0          = index[icg];
@@ -625,7 +631,7 @@
    iwater = (solvent_opt != esolNO) ? GET_CGINFO_SOLOPT(cginfo[icg]) : esolNO;

    bFreeEnergy = FALSE;
-   if (md->nPerturbed)
+   if (md->nPerturbed )
    {
        /* Check if any of the particles involved are perturbed.
         * If not we can do the cheaper normal put_in_list
@@ -683,6 +689,7 @@
    }

    if (!bFreeEnergy)
+/*   if (!bFreeEnergy || (fr->adress_type != eAdressOff)) */
    {
        if (iwater != esolNO)
        {
@@ -846,8 +853,13 @@
            bDoVdW_i = (bDoVdW && bHaveVdW[type[i_atom]]);
            bDoCoul_i = (bDoCoul && qi != 0);

+           /* chk */
+           bEnergyGroupCG = !egp_explicit(fr, igid);
+           /* chk */
+
            if (bDoVdW_i || bDoCoul_i)
            {
+
                /* Loop over the j charge groups */
                for (j = 0; (j < nj); j++)
                {
@@ -867,7 +879,19 @@
                    /* Finally loop over the atoms in the j-charge group */
                    for (jj = jj0; jj < jj1; jj++)
                    {
+
                        bNotEx = NOTEXCL(bExcl, i, jj);
+
                        /* change 7.11.2017 chk*/
                        if ( fr->adress_type != eAdressOff )
                        {
+
                            if ( ( !bEnergyGroupCG && ( wf[i_atom] <= GMX_REAL_EPS_
↪ || wf[jj] <= GMX_REAL_EPS ) ) ||
+
                            ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS_
↪ || wf[jj] > GMX_REAL_EPS ) ) )

```

(continues on next page)

(continued from previous page)

```

+// abrupt-GC          ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS &&
↪wf[jj] > GMX_REAL_EPS ) ))
+
+          {
+              continue;
+          }
+
+      /* change 7.11.2017 chk*/
+
+          if (bNotEx)
+          {
@@ -984,6 +1008,10 @@
+              jj1 = index[jcg+1];
+              /* Finally loop over the atoms in the j-charge group */
+              bFree = bPert[i_atom];
+
+
+              /* chk
+              bEnergyGroupCG = !egp_explicit(fr, igid); */
+
+              for (jj = jj0; (jj < jj1); jj++)
+              {
+                  bFreeJ = bFree || bPert[jj];
@@ -994,6 +1022,16 @@
+                  {
+                      bNotEx = NOTEXCL(bExcl, i, jj);
+
+
+                      /* chk
+
+                      if ( ( !bEnergyGroupCG && ( wf[i_atom] <= GMX_REAL_EPS ||
↪wf[jj] <= GMX_REAL_EPS ) ) ||
+                      ( ( bEnergyGroupCG ) && ( wf[i_atom] >= GMX_REAL_EPS &&
↪wf[jj] >= GMX_REAL_EPS ) )
+
+                      )
+
+                      {
+                          continue;
+                      } */
+
+                      if (bNotEx)
+                      {
+                          if (bFreeJ)
@@ -1250,6 +1288,19 @@
+                          * b_hybrid=true are placed into the _adress neighbour lists and
+                          * processed by the generic AdResS kernel.
+                          */
+
+                          /* change 7.11.2017 chk*/
+                          /* if ( fr->adress_type != eAdressOff )
+                          { */
+                          if ( ( !bEnergyGroupCG && ( wf[i_atom] <= GMX_REAL_EPS ||
↪wf[jj] <= GMX_REAL_EPS ) ) ||
+                          ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS ||
↪wf[jj] > GMX_REAL_EPS ) )
+
+                          )
+
+                          ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS &&
↪ wf[jj] > GMX_REAL_EPS ) )
+
+                          )
+
+                          {
+
+                          continue;

```

(continues on next page)

(continued from previous page)

```

+          }
+          /* } */
+/*      old version from normal GC-AdResS before october 7
+          if ( (bEnergyGroupCG &&
+              wf[i_atom] >= 1-GMX_REAL_EPS && wf[jj] >= 1-GMX_REAL_EPS ) )
+||
+          ( !bEnergyGroupCG && wf[jj] <= GMX_REAL_EPS ) )
@@ -1259,6 +1310,7 @@
+
+          b_hybrid = !((wf[i_atom] >= 1-GMX_REAL_EPS && wf[jj] >= 1-GMX_
+REAL_EPS) ||
+                      (wf[i_atom] <= GMX_REAL_EPS && wf[jj] <= GMX_REAL_
+REAL_EPS));
+*/
+
+          if (bNotEx)
+          {
@@ -1266,28 +1318,15 @@
+          {
+              if (charge[jj] != 0)
+              {
+                  if (!b_hybrid)
+                  {
+                      add_j_to_nblast(coul, jj, bLR);
+                  }
+                  else
+                  {
+                      add_j_to_nblast(coul_adress, jj, bLR);
+                  }
+                  /* chk: removed the !b_hybrid if loops */
+                  add_j_to_nblast(coul, jj, bLR);
+              }
+          }
+          else if (!bDoCoul_i)
+          {
+              if (bHaveVdW[type[jj]])
+              {
+                  if (!b_hybrid)
+                  {
+                      add_j_to_nblast(vdw, jj, bLR);
+                  }
+                  else
+                  {
+                      add_j_to_nblast(vdw_adress, jj, bLR);
+                  }
+              }
+          }
+          else
@@ -1296,38 +1335,16 @@
+          {
+              if (charge[jj] != 0)
+              {
+                  if (!b_hybrid)
+                  {
+                      add_j_to_nblast(vdwc, jj, bLR);
+                  }
+                  else

```

(continues on next page)

(continued from previous page)

```

-                                     {
-                                         add_j_to_nblast(vdwc_adress, jj, bLR);
-                                     }
-                                     }
-                                     else
-                                     {
-                                         if (!b_hybrid)
-                                         {
-                                             add_j_to_nblast(vdw, jj, bLR);
-                                         }
-                                         else
-                                         {
-                                             add_j_to_nblast(vdw_adress, jj, bLR);
-                                         }
-                                     }
-                                     }
-                                     }
-                                     else if (charge[jj] != 0)
-                                     {
-                                         if (!b_hybrid)
-                                         {
-                                             add_j_to_nblast(coul, jj, bLR);
-                                         }
-                                         else
-                                         {
-                                             add_j_to_nblast(coul_adress, jj, bLR);
-                                         }
-                                     }
-                                     }
-                                     }
@@ -2671,8 +2688,10 @@
    rvec      box_size, grid_x0, grid_x1;
    int       i, j, m, ngid;
    real      min_size, grid_dens;
+   real      b_hybrid;
    int       nsearch;
    gmx_bool   bGrid;
+   gmx_bool   bEnergyGroupCG;
    char      *ptr;
    gmx_bool   *i_egp_flags;
    int       cg_start, cg_end, start, end;
@@ -2774,8 +2793,9 @@
    }
    debug_gmx();

-   if (fr->adress_type == eAdressOff)
-   {
+/* chk */
+//   if (fr->adress_type == eAdressOff)
+//   {
        if (!fr->ns.bCGlist)
        {
            put_in_list = put_in_list_at;
@@ -2784,11 +2804,12 @@
        {
            put_in_list = put_in_list_cg;
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

-     }
-     else
-     {
-         put_in_list = put_in_list_address;
-     }
+//     }
+//     else
+//     {
+//         put_in_list = put_in_list_address;
+//     }
+/* chk */

    /* Do the core! */
    if (bGrid)
diff -ru /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/update.cpp /home/mi/
↪ck69giso/gmx-515-hck/src/gromacs/mdlib/update.cpp
--- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/update.cpp      2016-09-07_
↪14:50:21.000000000 +0200
+++ /home/mi/ck69giso/gmx-515-hck/src/gromacs/mdlib/update.cpp          2018-07-24_
↪16:07:27.000000000 +0200
@@ -67,6 +67,7 @@
#include "gromacs/utility/futil.h"
#include "gromacs/utility/gmxomp.h"
#include "gromacs/utility/smalloc.h"
+#include "adress.h"

/*For debugging, start at v(-dt/2) for velolcity verlet -- uncomment next line */
/*#define STARTFROMDT2*/
@@ -569,6 +570,8 @@
    return upd;
}

+/* new */
+
static void do_update_sd1(gmx_stochd_t *sd,
                        int start, int nrend, double dt,
                        rvec accel[], ivec nFreeze[],
@@ -579,10 +582,11 @@
                        int ngtc, real ref_t[],
                        gmx_bool bDoConstr,
                        gmx_bool bFirstHalfConstr,
-                        gmx_int64_t step, int seed, int* gatindex)
+                        gmx_int64_t step, int seed, int* gatindex, real fc)
{
    gmx_sd_const_t *sdc;
    gmx_sd_sigma_t *sig;

+
    real          kT;
    int           gf = 0, ga = 0, gt = 0;
    real          ism;
@@ -625,10 +629,21 @@
    {
        if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
↪nFreeze[gf][d])
        {
-
            real sd_V, vn;
+//
            real sd_V, vn;

```

(continues on next page)

(continued from previous page)

```

+          real sd_V, vn, fn;
+          fn          = f[n][d];
+
+//          fc = 10000.;
+
+          if (fabs(fn)>fc)
+          {
+              printf("SD (I) force-cap %e\n", fn);
+              fn = fc*fn/fabs(fn);
+          }
+
+          sd_V          = ism*sig[gt].V*rnd[d];
-          vn          = v[n][d] + (invmass[n]*f[n][d] + accel[ga][d])*dt;
+          vn          = v[n][d] + (invmass[n]*fn + accel[ga][d])*dt;
+//          vn          = v[n][d] + (invmass[n]*f[n][d] +
+↪accel[ga][d])*dt;
+          v[n][d]      = vn*sdc[gt].em + sd_V;
+          /* Here we include half of the friction+noise
+           * update of v into the integration of x.
@@ -668,7 +683,20 @@
+          {
+              if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
+↪nFreeze[gf][d])
+              {
-                  v[n][d]      = v[n][d] + (im*f[n][d] + accel[ga][d])*dt;
+
+                  real fn;
+
+//                  fc = 10000.;
+
+                  fn          = f[n][d];
+                  if (fabs(fn)>fc)
+                  {
+                      printf("SD (II) force-cap %e\n", fn);
+                      fn = fc*fn/fabs(fn);
+                  }
+
+                  v[n][d]      = v[n][d] + (im*fn + accel[ga][d])*dt;
+//                  v[n][d]      = v[n][d] + (im*f[n][d] + accel[ga][d])*dt;
+                  xprime[n][d] = x[n][d] + v[n][d]*dt;
+              }
+              else
@@ -1644,6 +1672,8 @@
+                  end_th      = start + ((nrend-start)*(th+1))/nth;
+
+                  /* The second part of the SD integration */
+                  if (inputrec->bAdress)
+                  {
+                      do_update_sd1(upd->sd,
+                                  start_th, end_th, dt,
+                                  inputrec->opts.acc, inputrec->opts.nFreeze,
@@ -1653,7 +1683,23 @@
+                                  inputrec->opts.ngtc, inputrec->opts.ref_t,
+                                  bDoConstr, FALSE,
+                                  step, inputrec->ld_seed,
-                                  DOMAINDECOMP(cr) ? cr->dd->gatindex : NULL);
+                                  DOMAINDECOMP(cr) ? cr->dd->gatindex : NULL,

```

(continues on next page)

(continued from previous page)

```

+         inputrec->adress->ex_forcecap);
+     }
+     else
+     {
+         do_update_sd1(upd->sd,
+             start_th, end_th, dt,
+             inputrec->opts.acc, inputrec->opts.nFreeze,
+             md->invmass, md->ptype,
+             md->cFREEZE, md->cACC, md->cTC,
+             state->x, xprime, state->v, force,
+             inputrec->opts.ngtc, inputrec->opts.ref_t,
+             bDoConstr, FALSE,
+             step, inputrec->ld_seed,
+             DOMAINDECOMP(cr) ? cr->dd->gatindex : NULL,
+             5000.);
+     }
+ }
+ inc_nrn(nrn, eNR_UPDATE, homenr);
+ wallcycle_stop(wcycle, ewcUPDATE);
@@ -2031,6 +2077,21 @@
+     break;
+     case (eiSD1):
+         /* With constraints, the SD1 update is done in 2 parts */
+     if (inputrec->bAdress)
+     {
+         do_update_sd1(upd->sd,
+             start_th, end_th, dt,
+             inputrec->opts.acc, inputrec->opts.nFreeze,
+             md->invmass, md->ptype,
+             md->cFREEZE, md->cACC, md->cTC,
+             state->x, xprime, state->v, force,
+             inputrec->opts.ngtc, inputrec->opts.ref_t,
+             bDoConstr, TRUE,
+             step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
+ gatindex : NULL,
+             inputrec->adress->ex_forcecap);
+     }
+     else
+     {
+         do_update_sd1(upd->sd,
+             start_th, end_th, dt,
+             inputrec->opts.acc, inputrec->opts.nFreeze,
+             state->x, xprime, state->v, force,
+             inputrec->opts.ngtc, inputrec->opts.ref_t,
+             bDoConstr, TRUE,
+             step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
+ gatindex : NULL);
+         step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
+ gatindex : NULL,
+             5000.);
+     }
+     break;
+     case (eiSD2):
+         /* The SD2 update is always done in 2 parts,

```

To apply the patch:

- 1) copy into the main directory (gromacs/)
- 2) patch < abrupt_adress.patch

In this module we also include a test scenario for GROMACS version 5.1.5 with a possible CG potential and all necessary input files. To run it simply run `gmx grompp -f grompp.mdp -c conf.gro -p topol.top -n index.ndx -maxwarn 5; gmx mdrun` using the patched version of GROMACS version 5.1.5 (see above).

When `gmx mdrun` finished normally (with the above mentioned setup), we have several mandatory checks to see if the simulation was successful or not.

- 0) Easiest check: load the conf.gro and the trajectory file in vmd and check if you see particle diffusion or depleted areas.
- 1) we check the density along the X-direction (*xsplitt*: e.g. `gmx density -f traj_compt.xtc -d X`) or along the radius (*sphere*: e.g. via VOTCA: `csg_density -axis r -rmax <value> -ref [x_ref,y_ref,z_ref] -trj traj_comp.xtc -top topol.tpr -out test.dens.comp`), the density has to be less then 3% different from experimental data or the density from a full atomistic MD simulation. The density of the example is 1000 kg m⁻³.
- 2) static properties: crucial RDF's (e.g. for water the oxygen-oxygen RDF)
- 3) p(N): It describes the average number of particles in the AT region throughout the simulation.
- 4) the density diffusion for each region (via a very helpful expansion for <http://www.ks.uiuc.edu/Research/vmd/>, the density profile tool see https://github.com/tonigi/vmd_density_profile).
- 5) If we only thermalize the transition region, the AT region is NVE-like, which means it is even possible to determine the dynamics of the system.

The files for the water example can be found here: `spc-example.tar.gz`

Software Technical Information

Name RDF's via Visualize Molecular Dynamics.

Language TCL scripting language.

Licence See <http://www.ks.uiuc.edu/Research/vmd/allversions/disclaimer.html>

Documentation Tool none

Application Documentation <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Relevant Training Material <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Radial Distribution Functions for GC-AdResS

- *Purpose of Module*
- *Background Information*
- *Example Collection*

Purpose of Module

One purpose of our project is to promote GC-AdResS as method which provides new insights and is not much more complex and difficult to use. In GC-AdResS simulations we introduce artificial interfaces, from atomistic to hybrid

and hybrid to coarse grained. To make sure that we indeed have an open system we have to check several properties, from structural to dynamic properties. Radial distribution functions (RDF's) are the easiest way to check the structural properties of the simulation. This module is dedicated to describe a straight forward and easy way to generate them for the atomistic regions in the GC-AdResS simulations. In the current implementation in GROMACS we have two geometric setups. One is radial and the other is a slab like structure. We use VMD as the tool of choice to calculate the RDF's.

Background Information

The most widespread tool for analysing molecular dynamics simulations is [VMD](#)). The program is based on TCL and Tk scripting language. Documentation and tutorials can be found here: [VMD Docs](#)

The reference coordinates (center of the AdResS region, as defined in the GROMACS mdp input file), configuration (standard input GROMACS: conf.gro) and trajectory (standard output GROMACS: traj_comp.xtc) are necessary to run AdResS. And they have to be used for two essential analysis parts for AdResS. The structural part, the radial distribution functions of the simulated system. And the second part with the help of the [Density Profile tool](#) (repository contains a tutorial on how to use) show the diffusion of the molecules during the simulation. The RDF plugin is described here: <https://www.ks.uiuc.edu/Research/vmd/plugins/gofrgui/>. All results can be stored as plain ASCII files, thus can be analysed via any plotting program.

CASE 1: RDF

- Since we apply the routine in a subspace, the normalization factor is wrong. The correcting factor can be obtained by calculating the RDF in the full atomistic case. Then the RDF in the same subspace (as in the AdResS) in the full atomistic case. The quotient can be used to re-normalize the AdResS RDF.
- Important: for the AdResS RDF the *update selection* option tick has to be used, and the PBC switched off.
- good case: the RDF's are exactly the same
- bad case: the RDF's are different somehow

CASE 2: Density Diffusion

- It is possible to define the different regions in the simulation box. Thus it is possible to look at the region specific density diffusion.
- To do that: one has to specify several time frames and calculate the average profiles, each one gives a ASCII file which (in the end) can be plotted together.
- The frame of reference is set by the slider in the main menu. (The update selection option has to be switched off.) The particles in that frame are tagged and then via the different time frames one follows their path.
- Good case: smooth regular and symmetric diffusion
- bad case: spikes at the interface and asymmetric diffusion might hint at an artificial interface between the different regions.

Example Collection

We basically work with the atom selection and use the pre-existing Radial Pair Distribution Function tool in VMD. One has to load the configuration file (standard is conf.gro) and the trajectory file (standard: traj_comp.xtc) via either `vmd conf.gro traj_comp.xtc` or the vmd GUI.

Example case (for the radial systems), this selection was used with the reference point being defined in the GROMACS input file.

```

x_ref = x-value of the center of the chosen AdResS region
y_ref = y-value of the center of the chosen AdResS region
z_ref = z-value of the center of the chosen AdResS region

radius: radius of the atomistic region
name "insert your choice of atom here" and (((x-x_ref)^2 + (y-y_ref)^2 + (z-z_ref)^2)
↪< radius*radius )

```

For the slab structures:

```

at_start: start of the atomistic regions along the x axis
at_end: end of the atomistic regions along the x axis
hy_start: start of the hybrid regions along the x axis
hy_end: end of the hybrid regions along the x axis

name "insert your choice of atom here" and (x>at_start and x<at_end)
name "insert your choice of atom here" and ((x>hy_start and x<at_start) or (x>at_end_
↪and x<hy_end)

```

Software Technical Information

Name GC-AdResS -Abrupt scheme- Force Capping

Language Implemented in GROMACS version 5.1.5

Licence See GROMACS web page: <http://www.gromacs.org/>

Documentation Tool

Application Documentation See GROMACS web page: <http://www.gromacs.org/>

Relevant Training Material See GROMACS web page: <http://www.gromacs.org/>

Abrupt-AdResS: Forcecap

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The original idea of our proposal was to work on a general implementation of grand canonical adaptive resolution simulations (GC-AdResS) in classical MD packages. The current implementation of GC- AdResS in GROMACS (up to version 5.1.5) has performance problems. The Abrupt GC-AdResS implementation is avoiding those and make AdResS more interesting for other MD developers (especially since we could remove the force interpolation and weighting functions from the force kernel).

This module works in combination with abrupt_AdResS and is at the same time important for a successful simulation. It shows how to avoid particle overlap at the interface of the atomistic and coarse grained regions.

Purpose of Module

The implementation of Abrupt GC-AdResS is in itself only working for the smallest and simplest of molecules without problems. For larger and more complex molecules the simulation crashes. This module shows a way to avoid this.

Background Information

As studies of ionic liquids and polymer melts have shown for large and complicated molecules even the standard GC-AdResS is not working without additional force capping. The reason for that is when a molecule from the coarse grained region enters the hybrid region the atomistic representations, which are present due to the technically necessary double resolution, interact. It is possible for atoms to be too close together, which results in a too high force and thus in too high velocities of those particles. Since in Abrupt AdResS we can avoid the generic force kernel from GROMACS, the force capping (which was previously implemented at the end of the force calculation) had to be shifted and replaced. We finally looked at the integrator (in our case the stochastic dynamics integrator), which is the place where each force has to be read and handled.

This implementation of force capping is a rudimentary approach. The basic principle is when two particles are too close together, and thus the force are far higher than the average forces in the simulation, the force on the particles are re-scaled to a given value. That tactic makes sure that the insertion of particles in the atomistic region is introduced at reasonable velocities and temperature. As a side effect, the area of disturbance due to the introduction of a particle is limited.

Building and Testing

We have used this new addition to the code on two systems: water and ionic liquids. The results have been published in Ref. <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>. All the information about studied systems and the performance can be found there.

The patch provided can be applied alone without the Abrupt AdResS patch, in the main directory of GROMACS. The important part of the patch is that it adds an upper force limit in the stochastic dynamics integrator. If that upper limit is triggered the force is re-scaled to the given force cap. The rest is basically to make sure that the integrator is called correctly. There is a *print* command which is triggered once the force on a particle is higher than a given force cap value. The force capping simulation can in some cases cause lincs warnings. Since we take care of faulty configuration that way, we can disabling those warnings (*export GMX_MAXBACKUP=-1 ; export GMX_MAXCONSTRWARN=-1*). Otherwise it generates too many files and can crash as well.

Source Code

A note of caution: the chosen force cap trigger has to be a high enough value, otherwise normal interactions (interactions with forces around the average forces in a simulation) will trigger the force capping. That would change the dynamics and the structure of a system. Also it would decrease the performance of the code. If chosen too high it might run into impossible and unstable configuration, which will result in a program crash.

Recipe for hard coded force capping:

```
diff -ru /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/update.cpp /home/mi/
→ck69giso/gmx-515-hck/src/gromacs/mdlib/update.cpp
--- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/update.cpp      2016-09-07
→14:50:21.000000000 +0200
+++ /home/mi/ck69giso/gmx-515-hck/src/gromacs/mdlib/update.cpp      2018-07-24
→16:07:27.000000000 +0200
@@ -67,6 +67,7 @@
```

(continues on next page)

(continued from previous page)

```

#include "gromacs/utility/futil.h"
#include "gromacs/utility/gmxomp.h"
#include "gromacs/utility/smalloc.h"
#include "adress.h"

/*For debugging, start at v(-dt/2) for velolcity verlet -- uncomment next line */
/*#define STARTFROMDT2*/
@@ -569,6 +570,8 @@
    return upd;
}

+/* new */
+
static void do_update_sd1(gmx_stochd_t *sd,
                        int start, int nrend, double dt,
                        rvec accel[], ivec nFreeze[],
@@ -579,10 +582,11 @@
                        int ngtc, real ref_t[],
                        gmx_bool bDoConstr,
                        gmx_bool bFirstHalfConstr,
                        gmx_int64_t step, int seed, int* gatindex)
+
+
                        gmx_int64_t step, int seed, int* gatindex, real fc)
{
    gmx_sd_const_t *sdc;
    gmx_sd_sigma_t *sig;
+
    real      kT;
    int       gf = 0, ga = 0, gt = 0;
    real      ism;
@@ -625,10 +629,21 @@
    {
        if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
nFreeze[gf][d])
        {
            real sd_V, vn;
+//
            real sd_V, vn;
            real sd_V, vn, fn;
            fn = f[n][d];
+
+
            fc = 10000.;
+
+
            if (fabs(fn)>fc)
            {
                printf("SD (I) force-cap %e\n", fn);
                fn = fc*fn/fabs(fn);
            }

            sd_V = ism*sig[gt].V*rnd[d];
            vn = v[n][d] + (invmass[n]*f[n][d] + accel[ga][d])*dt;
+
            vn = v[n][d] + (invmass[n]*fn + accel[ga][d])*dt;
+//
            vn = v[n][d] + (invmass[n]*f[n][d] +
naccel[ga][d])*dt;
            v[n][d] = vn*sdc[gt].em + sd_V;
            /* Here we include half of the friction+noise
             * update of v into the integration of x.
@@ -668,7 +683,20 @@
        {

```

(continues on next page)

(continues on next page)

(continued from previous page)

```

        case (eiSD1):
            /* With constraints, the SD1 update is done in 2 parts */
+         if (inputrec->bAdress)
+         {
+             do_update_sd1(upd->sd,
+                 start_th, end_th, dt,
+                 inputrec->opts.acc, inputrec->opts.nFreeze,
+                 md->invmass, md->ptype,
+                 md->cFREEZE, md->cACC, md->cTC,
+                 state->x, xprime, state->v, force,
+                 inputrec->opts.ngtc, inputrec->opts.ref_t,
+                 bDoConstr, TRUE,
+                 step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
+                 gatindex : NULL,
+                 inputrec->adress->ex_forcecap);
+         }
+         else
+         {
+             do_update_sd1(upd->sd,
+                 start_th, end_th, dt,
+                 inputrec->opts.acc, inputrec->opts.nFreeze,
+                 state->x, xprime, state->v, force,
+                 inputrec->opts.ngtc, inputrec->opts.ref_t,
+                 bDoConstr, TRUE,
+                 step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
+                 gatindex : NULL);
+             step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
+             gatindex : NULL,
+             5000.);
+         }
+         break;
        case (eiSD2):
            /* The SD2 update is always done in 2 parts,

```

with:

fc = Chosen upper force limit for the ionic liquids simulations

fn = force acting on a particle

The patch provided can be applied in the main directory of GROMACS via:

```
patch < forcecap.patch
```

Software Technical Information

Name Thermodynamic Force Calculator for Abrupt AdResS

Language bash Python 2.7

Licence MD Simulation: See GROMACS web page: <http://www.gromacs.org/>

Analysis tools: see VOTCA web page: <http://www.votca.org/home>

Documentation Tool

Application Documentation See GROMACS web page: <http://www.gromacs.org/> See VOCTA web page: <http://www.votca.org/Documentation>

Relevant Training Material See GROMACS web page: <http://www.gromacs.org/> See VOCTA web page: <http://www.votca.org/tutorials>

Thermodynamic Force Calculator for Abrupt AdResS

- *Purpose of Module*
- *Background Information*
- *Building and Running*
- *Source Code*

We introduced with the Abrupt AdResS method a new way of coupling the different simulation regions together. That is the basis for easier implementation into other codes. The implementation of smooth coupling GC- AdResS in GROMACS has several performance problems. However, the new Abrupt AdResS presents a very straight forward way to implement a new partitioning scheme, which solves two problems which affect the performance, the neighbor list search and the generic force kernel. Furthermore, we update the implementation to address this in a way that decouples the method directly from the core of any MD code, which does not hinder the performance and makes the scheme hardware independent. Theory, application and tests see <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>.

The drawback of this method is that a new (as in more direct) way to calculate the thermodynamic force is needed. While the theory is still the same, the interpolation has to be adapted.

Purpose of Module

The new Abrupt coupling scheme introduces a density discrepancy which is very much restricted to the interface of the atomistic region and the coarse grained region. The thermodynamic force calculator in VOTCA (implemented up to version 1.3) is designed for the more smooth coupling over a larger region in space. Thus this code cannot be used for the small area of disturbance in this new scheme.

Here we present a thermodynamic force calculator for the abrupt coupling scheme. It is a mix between bash and Python and can be applied even to the older smooth coupling scheme.

Background Information

Abrupt AdResS presents a very straight forward way to implement a new partitioning scheme. The drawback is that the particles mix in a very narrow region in space. The distance of the molecules at that interface can be too close, which has to be compensated via a force capping. However the flux of particles at that interface is fast, which leads to a rather localized discrepancy in the density.

The thermodynamic force calculator in VOTCA (up to version 1.3) is designed for a smooth and not very much disturbed region in space. Thus a new code to calculate the thermodynamic force was needed. The thermodynamic force is calculated by calculating the gradient of the density in a specific region in space. Thus any code taking this into account can be used. For the detailed discussion of the role and the basic principles behind this force see <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>.

The code provided here is designed for easy adjustable and can be used on different computer architectures (knowledge of bash is of an advantage).

Building and Running

These three scripts present one way to calculate the thermodynamics force.

- TF calculation script:
- Density interpolation script:
- Script to call TF calculation:

The central script is *smooth_dens.sh*. This is a Python 2.7 script which interpolates the density and generates the gradient of the density and provides the force as an ascii table.

TF_calc_water_xplit_sphere.sh is controlling the MD run, and which region to interpolate, and builds the tables needed. The commands and the options used are described in <http://www.gromacs.org/> or if the spherical geometry for AdResS is used also here: <http://www.votca.org/Documentation>.

run_tf_water_xplsit_sphere.sh provides some possible input scenarios for *TF_calc_water_xplit_sphere.sh*.

To run *run_tf_water_xplsit_sphere.sh* one has to first enter the correct region, which should be interpolated in *TF_calc_water_xplit_sphere.sh*.

```
rmin = starting point along a distance with rref as origin
rmax = end point of the interpolation region
rbox = maximal box size (xsplit: x direction; sphere the maximal radius)
rref = is the point defined in the GROMACS input file
lc = defines the binning along the x direction or the radius
prefac = is basically a weighting on the thermodynamic force (small: more iteration,
↳but more careful approach of the target density)
```

Note of caution: in *run_tf_water_xplsit_sphere.sh* and *TF_calc_water_xplit_sphere.sh* the GROMACS and VOTCA version used have to be specifically sourced. Then select which option in *run_tf_water_xplsit_sphere.sh* you want to use and comment the other out and execute:

```
for a new run without a thermodynamic force to start with:

bash run_tf_water_xplsit_sphere.sh 1 20 1

for a start from an existing thermodynamic force:

bash run_tf_water_xplsit_sphere.sh 21 20 2
```

Source Code

TF calculation script: Density interpolation script: Script to call TF
calculation:

Software Technical Information

Name GC-AdResS: Local thermostat adaption of the Abrupt AdResS scheme

Language Implemented in GROMACS version 5.1.5

Licence MD Simulation: See GROMACS web page: <http://www.gromacs.org/>

Analysis tools and thermodynamic force calculation: see VOTCA web page: <http://www.votca.org/home>

Documentation Tool

Application Documentation See GROMACS web page: <http://www.gromacs.org/> See VOCTA web page: <http://www.votca.org/Documentation>

Relevant Training Material See GROMACS web page: <http://www.gromacs.org/> See VOCTA web page: <http://www.votca.org/tutorials>

Local thermostat adaption of the Abrupt GC-AdResS scheme

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The original idea of our proposal was to work on a general implementations of AdResS in classical MD packages. With Abrupt AdresS, we implemented a new partitioning scheme to bypass the performance problems in the smooth coupling GC-AdResS implementation. We switched to the general interaction kernel and simplified the neighbor list search.

This module takes this new ansatz and combines it with a local thermostat approach (see Ref. <http://iopscience.iop.org/article/10.1088/1367-2630/17/8/083042>). The advantages of this are: we can thermalize the transition and the coarse grained region, the atomistic region is thermalized indirectly. Furthermore, we still have a method which is decoupled from the core of any MD code. For theory, application and tests see [Link\(J.Chem.Phys.\):](#) or [Link\(arXiv\):](#).

Purpose of Module

If one looks at the AdResS simulations it is possible to describe it in a nutshell as partitioning the simulation box into different regions. The Abrupt coupling scheme has one atomistic and one coarse grained region, coupled via a transition region, where an additional force is acting on the molecules. In previous work (see [this reference](#)) the idea of a local thermostat was introduced.

This module describes how to couple that ansatz with our new Abrupt AdResS. The first test, as well as an overview over the theory, can be found at <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>.

Background Information

This module combines two different codes. The first part is the Abrupt coupling scheme already presented in a module. In any AdResS simulations the simulation box is partitioned into different regions. Previous AdResS implementations have three regions (atomistic, hybrid and coarse grained). They are coupled in the hybrid region by slowly switching between atomistic forces and coarse grained forces. That is done by introducing the weighting function required for

the smooth coupling into the force kernel, which slows down the performance of the code. The Abrupt coupling scheme has one atomistic and one coarse grained region, coupled via a transition region, which is only due to the fact that an additional force is acting on the molecules. Since the weighting function is not needed any more, the forces are calculated via the standard GROMACS kernels, which increases the performance.

In previous work by Agarwal et al. (Ref. <http://iopscience.iop.org/article/10.1088/1367-2630/17/8/083042>) the idea of a local thermostat was introduced. Since in AdResS the simulation box is partitioned into different regions the next logical step is to adapt the thermalization of the box and apply the thermostat only in the hybrid and coarse grained region, since the hybrid region is an artificial region and the coarse grained region represents a reservoir.

This module describes how to couple these two approaches. The first test, as well as an overview of the theory, can be found here [Link\(J.Chem.Phys.\):](#) or [Link\(arXiv\):](#).

Building and Testing

We tested this new implementation on SPC water and 1,3-dimethyl imidazolium chloride. The results are published in <https://aip.scitation.org/doi/10.1063/1.5031206> or <https://arxiv.org/abs/1806.09870>.

Here is a short manual on how to run the test and set up the local thermostat simulation within this Abrupt AdResS simulations in GROMACS:

- 1) mandatory requirement: a working full atomistic simulation (molecular configuration, force fields and optimal MD input parameter)
- 2) You need a very well converged NVT run, which can be used as starting point for the coarse grained (CG) and then later the AdResS simulations.
- 3) You have to generate a coarse grained (CG) potential. We use, for convenience, the inverse Boltzmann iteration provided in the VOTCA package (<http://www.votca.org/home>). The resulting tabulated CG potentials are used in the AdResS simulation. Alternatively you can use WCA potentials or standard Lennard-Jones potentials. The main requirement for the AdResS simulation is that the density in the CG region is the same as in the atomistic (AT) region.

NOTE: The method can be used with any potential, which preserves the correct density. If only a SPC/E CG potential is available it can be used for SPC/e water models as well as for a more advanced water model. It is possible to use a WCA potential, which is basically a Lennard-Jones potential. And it is possible to switch the CG potential completely off. That will transform the CG region to a true thermodynamic reservoir with a non-interacting gas.

- 4) The next step is to create a double resolution configuration and adjust the dependencies (force field, topology, index file, GROMACS input file). Creating the configuration is straightforward (we use VOTCA [VOTCA](#)):

```
Example from VOTCA:
csg_map --top topol.tpr --cg cg_mapping_scheme --hybrid --trj conf.gro --out conf_
↪ hybrid.gro
```

Of course, if you want to use this configuration in a MD simulation you have to adjust the force field (see example file: *spc.adress.itp* in the [abrupt_adress example repository](#)). You have to define a virtual site:

```
[ virtual_sites3 ]
; Site from funct a d
; atom dependencies func      a      b
  4      1 2 3      1      0.05595E+00 0.05595E+00
```

The next step is to adjust the status of the CG particle in the topology file (in our example: *topol.top*) from *A* for *atom* to *V* as *virtual particle*. And of course insert the new force field.


```
#include "spc.adress.itp"
```

Then you have to generate an index file with the different energy groups. In this example, we have 2 groups (EXW and WCG, the name of the CG particle):

```
gmx make_ndx -f conf_hybrid.gro
> a WCG
> !3
> name 4 EXW
> q
```

The next step is to adjust the GROMACS input file. AdResS needs the Langevin dynamics, so you have to choose:

```
integrator = sd
```

Since the system is double resolution, meaning we have the atomistic details and the virtual particles, we have to define the energy groups:

```
; Selection of energy groups
energygrps = EXW WCG
energygrp_table = WCG WCG
```

Note: the table for the WCG and WCG particles can be a pre-defined coarse grained potential but can be also set to zero.

Alternatively, the non-bonded interactions for WCG in the force field can be set to zero.^a Then the input would be like:

```
; Selection of energy groups
energygrps = EXW WCG
```

GROMACS version 5.1.5 is using verlet as standard cutoff-scheme, so we have to change that to *group*:

```
; nblist update frequency
cutoff-scheme = group
```

In case of local thermostat simulations (see [Link \(for J.Chem.Phys.\):](#) or [Link \(for arXiv\):](#)) we use:

```
coulombtype = reaction-field-zero
rcoulomb = 1.0
vdw-type = user
rvdw = 1.0
```

If you use the stochastic dynamics, we add the following entries to make sure we have only NVT and a thermalization via the Langevin dynamics.

```
; Temperature coupling
Tcoupl = no
Pcoupl = no
```

To switch the simulation to AdResS this is the key part. This starts the AdResS runs.

```
; AdResS parameters
adress = yes ;no
```

Here you define the geometry of the atomistic region, either *sphere* (a spherical region anywhere in the simulation box) or *xsplit* (a cuboid slice of the whole simulation box for the atomistic region, with the transition and coarse grained region on each side).

```
adress_type = sphere ;xsplit sphere or constant
```

This defines the width of the atomistic region, starting from the given reference coordinate (keyword *adress_reference_coords*, by simply using: `tail conf_hybrid.gro | awk '(NF==3){print $1/2., $2/2., $3/2.}'`). In the older versions of AdResS, with a smooth coupling between AT and CG the width of the hybrid region width (*adress_hy_width*) was also defined. In the Abrupt_AdResS setup it is not necessary any more, even if you put a number that region is counted (in the code) as AT.

```
adress_ex_width = 1.5
adress_hy_width = 1.5
adress_ex_forcecap = 2000
adress_interface_correction = thermoforce ;off
adress_site = com
adress_reference_coords = 3.7500 1.860355 1.860355
adress_tf_grp_names = WCG
adress_cg_grp_names = WCG
adress_do_hybridpairs = no
```

Another important aspect is the force capping. Abrupt AdResS works fine for small molecules like water, but for larger or more complex molecules the force capping is very important. We cap every force component (i.e. $f(x), f(y), f(z)$) acting on a particle and not the norm of the force, which reduces the computational time spend. This is described in another module.

adress_interface_correction defines if you use an external force to correct the density or not. In case of the old AdResS (smooth coupling) that correction simply refined the simulation, as the density difference was not significant. For the Abrupt AdResS, and the method development based on it, and more complex molecules (i.e. polymers) the thermodynamic force is essential. If it is not taken into account the risk to form interfaces between AT and CG is high. Also if particles coming too close (basically overlap) the run can crash. The role of the thermodynamic force, the force cap and the basic theory behind it see [Link \(for J.Chem.Phys.\):](#) or [Link \(for arXiv\):](#). For this to work you must have a file e.g. in our example case: *tabletf_WCG.xvg* in the directory, otherwise you have to set:

```
adress_interface_correction = off
```

The local thermostat simulations are significantly different from the Abrupt coupling AdResS simulations. The atomistic region is indirectly thermalized by the hybrid/coarse grained, which leads to an NVE-like environment. To make sure the simulations run smoothly, a tabulated potential for shifted Lennard-Jones potentials is needed. Furthermore, GROMACS has to be compiled with double resolutions. It is easy to see when the simulation didn't work, as the atomistic region is evacuated by all molecules and the resulting density has an error of around 50% and higher.

When the simulation worked, the same checks as for Abrupt AdResS are required. The first check is about the density: if you have no thermodynamic force you will have rather pronounced spikes in the density at the interfaces. If you have a converged thermodynamic force the density has to be within +/- 3% (optimal) and +/- 5% (still valid) off from a comparable full atomistic simulation / experimental data. However, an "artificial" interface is introduced and checks for the diffusion, the RDFs, etc. (full list see below), ensure that the regions mix together and that you have proper particle transfer.

This is an example of test scenario for GROMACS version 5.1.5 with a possible CG potential and all necessary input files, see https://gitlab.e-cam2020.eu:10443/abrupt_adress/abrupt_adress. To run it simply run `gmx grompp -f grompp.mdp -c conf.gro -p topol.top -n index.ndx -maxwarn 5; gmx mdrun` using the patched version of GROMACS version 5.1.5 (see above).

When `gmx mdrun` finishes normally (with the above mentioned setup), we have several mandatory checks to see if the simulation was successful or not.

- 0) Easiest check: load the `conf.gro` and the trajectory file in `vmd` and check if you see particle diffusion or depleted areas.

- 1) we check the density along the X-direction (*xsplrit*: e.g. `gmx density -f traj_compt.xtc -d X`) or along the radius (*sphere*: e.g. via VOTCA: `csg_density -axis r -rmax <value> -ref [x_ref,y_ref,z_ref] -trj traj_comp.xtc -topol.tpr -out test.dens.comp`), the density has to be less then 3% different from experimental data or the density from a full atomistic MD simulation. The density of the example is 1000 kg m⁻³.
- 2) static properties: crucial RDFs (e.g. for water the oxygen-oxygen RDF)
- 3) *p(N)*: It describes the average number of particles in the AT region throughout the simulation.
- 4) the density diffusion for each region (via a very helpful expansion for [VMD](#), the density profile tool see [Link](#)).
- 5) If we only thermalize the transition region, the AT region is NVE-like, which means it is even possible to determine the dynamics of the system.

Source Code

To apply the patch: 1) copy into the main directory (gromacs/) 2) `patch < localT_abrupt_adress.patch`

The patch for Abrupt_AdResS can be found here: ([Patch file for module: Abrupt AdResS](#))

Patch file for module: Abrupt AdResS

The patch for the abrupt AdResS code is:

```

1 diff -Naur /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/legacyheaders/update.h /
  ↪home/mi/ck69giso/gmx-515-lt/src/gromacs/legacyheaders/update.h
2 --- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/legacyheaders/update.h      2016-
  ↪07-13 14:56:04.000000000 +0200
3 +++ /home/mi/ck69giso/gmx-515-lt/src/gromacs/legacyheaders/update.h      2018-12-03_
  ↪12:07:23.228392303 +0100
4 @@ -102,7 +102,9 @@
5
6         t_commrec          *cr, /* these shouldn't be here -- need to think_
  ↪about it */
7
8         t_nrn             *nrnb,
9         gmx_constr_t      constr,
10        - t_idef            *idef);
11        + t_idef            *idef,
12        + t_forcerec        *fr);
13
14        /* Return TRUE if OK, FALSE in case of Shake Error */
15
16        @@ -125,7 +127,8 @@
17
18        gmx_update_t        upd,
19        gmx_constr_t        constr,
20        gmx_bool             bFirstHalf,
21        - gmx_bool           bCalcVir);
22        + gmx_bool           bCalcVir,
23        + t_forcerec        *fr);
24
25        /* Return TRUE if OK, FALSE in case of Shake Error */
26
27 diff -Naur /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/adress.c /home/mi/
  ↪ck69giso/gmx-515-lt/src/gromacs/mdlib/adress.c
28 --- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/adress.c      2016-07-13_
  ↪14:56:04.000000000 +0200
29 +++ /home/mi/ck69giso/gmx-515-lt/src/gromacs/mdlib/adress.c      2018-08-15_
  ↪12:39:32.000000000 +0200

```

(continues on next page)

(continued from previous page)

```

28 @@ -101,17 +101,17 @@
29     return 0;
30 }
31 /* molecule is explicit */
32 - else if (sqr_dl < adressr*adressr)
33 + else //if (sqr_dl < adressr*adressr)
34 {
35     return 1;
36 }
37 - /* hybrid region */
38 + /* hybrid region
39 else
40 {
41     dl = sqrt(sqr_dl);
42     tmp = cos((dl-adressr)*M_PI/2/adressw);
43     return tmp*tmp;
44 }
45 + {
46 +//     dl = sqrt(sqr_dl);
47 +//     tmp = cos((dl-adressr)*M_PI/2/adressw);
48 +     return 0.5;
49 + } */
50 }
51
52 void
53 diff -Naur /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/ns.c /home/mi/
54 ↪ck69giso/gmx-515-lt/src/gromacs/mdlib/ns.c
55 --- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/ns.c      2016-07-13_
56 ↪14:56:04.000000000 +0200
57 +++ /home/mi/ck69giso/gmx-515-lt/src/gromacs/mdlib/ns.c      2018-08-15 12:55:06.
58 ↪000000000 +0200
59 @@ -264,10 +264,12 @@
60     for (i = 0; i < fr->nnblists; i++)
61     {
62         nbl = &(fr->nnblists[i]);
63
64         -
65         - if ((fr->adress_type != eAdressOff) && (i >= fr->nnblists/2))
66 +// if ((fr->adress_type != eAdressOff) && (i >= fr->nnblists/2))
67 + if ((fr->adress_type != eAdressOff))
68         {
69             type = GMX_NBLIST_INTERACTION_ADDRESS;
70             /* type = GMX_NBLIST_INTERACTION_ADDRESS; */
71             type = GMX_NBLIST_INTERACTION_STANDARD;
72         }
73         init_nblast(log, &nbl->nlist_sr[eNL_VDWQQ], &nbl->nlist_lr[eNL_VDWQQ],
74             maxsr, maxlr, ivdw, ivdwmod, ielec, ielecmod, igeometry_def,
75             ↪type, bElecAndVdwSwitchDiffers);
76 @@ -601,11 +603,14 @@
77     int      *cginfo;
78     int      *type, *typeB;
79     real     *charge, *chargeB;
80 + real     *wf;
81     real     qi, qiB, qq, rlj;
82     gmx_bool bFreeEnergy, bFree, bFreeJ, bNotEx, *bPert;
83     gmx_bool bDoVdW_i, bDoCoul_i, bDoCoul_i_sol;
84 + gmx_bool  b_hybrid;

```

(continues on next page)

(continued from previous page)

```

81     int          iwater, jwater;
82     t_nblast     *nlist;
83 +   gmx_bool      bEnergyGroupCG;
84
85     /* Copy some pointers */
86     cginfo = fr->cginfo;
87 @@ -614,6 +619,7 @@
88     type      = md->typeA;
89     typeB     = md->typeB;
90     bPert     = md->bPerturbed;
91 +   wf        = md->wf;
92
93     /* Get atom range */
94     i0        = index[icg];
95 @@ -625,7 +631,7 @@
96     iwater = (solvent_opt != esolNO) ? GET_CGINFO_SOLOPT(cginfo[icg]) : esolNO;
97
98     bFreeEnergy = FALSE;
99 -   if (md->nPerturbed)
100 +   if (md->nPerturbed )
101     {
102         /* Check if any of the particles involved are perturbed.
103          * If not we can do the cheaper normal put_in_list
104 @@ -683,6 +689,7 @@
105     }
106
107     if (!bFreeEnergy)
108 +/*     if (!bFreeEnergy || (fr->adress_type != eAdressOff)) */
109     {
110         if (iwater != esolNO)
111         {
112 @@ -846,8 +853,13 @@
113             bDoVdW_i = (bDoVdW  && bHaveVdW[type[i_atom]]);
114             bDoCoul_i = (bDoCoul && qi != 0);
115
116 +           /* chk */
117 +           bEnergyGroupCG = !egp_explicit(fr, igid);
118 +           /* chk */
119 +
120             if (bDoVdW_i || bDoCoul_i)
121             {
122 +
123                 /* Loop over the j charge groups */
124                 for (j = 0; (j < nj); j++)
125                 {
126 @@ -867,7 +879,19 @@
127                     /* Finally loop over the atoms in the j-charge group */
128                     for (jj = jj0; jj < jj1; jj++)
129                     {
130 +
131                         bNotEx = NOTEXCL(bExcl, i, jj);
132 +           /* change 7.11.2017 chk*/
133 +           if ( fr->adress_type != eAdressOff )
134 +           {
135 +               if ( ( !bEnergyGroupCG && ( wf[i_atom] <= GMX_REAL_EPS_
136 + || wf[jj] <= GMX_REAL_EPS ) ||
137 +               ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS_
138 + || wf[jj] > GMX_REAL_EPS ) ) )

```

(continues on next page)

(continued from previous page)

```

137 +// abrupt-GC          ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS &&
↳wf[jj] > GMX_REAL_EPS ) ))
138 +                      {
139 +                          continue;
140 +                      }
141 +
142 +      /* change 7.11.2017 chk*/
143
144 +                      if (bNotEx)
145 +                      {
146 @@ -984,6 +1008,10 @@
147 +                          jj1 = index[jcg+1];
148 +                          /* Finally loop over the atoms in the j-charge group */
149 +                          bFree = bPert[i_atom];
150 +
151 +                          /* chk
152 +                          bEnergyGroupCG = !egp_explicit(fr, igid); */
153 +
154 +                          for (jj = jj0; (jj < jj1); jj++)
155 +                          {
156 +                              bFreeJ = bFree || bPert[jj];
157 @@ -994,6 +1022,16 @@
158 +                              {
159 +                                  bNotEx = NOTEXCL(bExcl, i, jj);
160 +
161 +                                  /* chk
162 +
163 +                                  if ( ( !bEnergyGroupCG && ( wf[i_atom] <= GMX_REAL_EPS ||
↳wf[jj] <= GMX_REAL_EPS ) ) ||
164 +                                  ( ( bEnergyGroupCG ) && ( wf[i_atom] >= GMX_REAL_EPS &&
↳wf[jj] >= GMX_REAL_EPS ) )
165 +                                  )
166 +                                  {
167 +                                      continue;
168 +                                  } */
169 +
170 +                                  if (bNotEx)
171 +                                  {
172 +                                      if (bFreeJ)
173 @@ -1250,6 +1288,19 @@
174 +                                      * b_hybrid=true are placed into the _adress neighbour lists and
175 +                                      * processed by the generic AdResS kernel.
176 +                                      */
177 +
178 +                          /* change 7.11.2017 chk*/
179 +                          /* if ( fr->adress_type != eAdressOff )
180 +                          { */
181 +                          if ( ( !bEnergyGroupCG && ( wf[i_atom] <= GMX_REAL_EPS ||
↳wf[jj] <= GMX_REAL_EPS ) ) ||
182 +                          ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS ||
↳wf[jj] > GMX_REAL_EPS ) )
183 +                          )
184 +                          ( ( bEnergyGroupCG ) && ( wf[i_atom] > GMX_REAL_EPS &&
↳ wf[jj] > GMX_REAL_EPS ) )
185 +                          )
186 +                          {
187 +                              continue;

```

(continues on next page)

(continued from previous page)

```

188 +          }
189 +          /*      } */
190 +/*      old version from normal GC-AdResS before october 7
191 +          if ( (bEnergyGroupCG &&
192 +              wf[i_atom] >= 1-GMX_REAL_EPS && wf[jj] >= 1-GMX_REAL_EPS ) )
193 +          ( !bEnergyGroupCG && wf[jj] <= GMX_REAL_EPS ) )
194 @@ -1259,6 +1310,7 @@
195
196 +          b_hybrid = !((wf[i_atom] >= 1-GMX_REAL_EPS && wf[jj] >= 1-GMX_
197 +REAL_EPS) ||
198 +                      (wf[i_atom] <= GMX_REAL_EPS && wf[jj] <= GMX_REAL_
199 +REAL_EPS));
200 +*/
201
202 +          if (bNotEx)
203 +          {
204 +              @@ -1266,28 +1318,15 @@
205 +              {
206 +                  if (charge[jj] != 0)
207 +                  {
208 +                      if (!b_hybrid)
209 +                      {
210 +                          add_j_to_nblast(coul, jj, bLR);
211 +                      }
212 +                      else
213 +                      {
214 +                          add_j_to_nblast(coul_adress, jj, bLR);
215 +                      }
216 +                      /* chk: removed the !b_hybrid if loops */
217 +                      add_j_to_nblast(coul, jj, bLR);
218 +                  }
219 +              }
220 +              else if (!bDoCoul_i)
221 +              {
222 +                  if (bHaveVdW[type[jj]])
223 +                  {
224 +                      if (!b_hybrid)
225 +                      {
226 +                          add_j_to_nblast(vdw, jj, bLR);
227 +                      }
228 +                      else
229 +                      {
230 +                          add_j_to_nblast(vdw_adress, jj, bLR);
231 +                      }
232 +                  }
233 +              }
234 +              else
235 +              @@ -1296,38 +1335,16 @@
236 +              {
237 +                  if (charge[jj] != 0)
238 +                  {
239 +                      if (!b_hybrid)
240 +                      {
241 +                          add_j_to_nblast(vdwc, jj, bLR);
242 +                      }
243 +                      else

```

(continues on next page)

(continued from previous page)

```

242         {
243             add_j_to_nblast(vdwc_adress, jj, bLR);
244         }
245     }
246     else
247     {
248         if (!b_hybrid)
249         {
250             add_j_to_nblast(vdw, jj, bLR);
251         }
252         else
253         {
254             add_j_to_nblast(vdw_adress, jj, bLR);
255         }
256     }
257 }
258 }
259 else if (charge[jj] != 0)
260 {
261     if (!b_hybrid)
262     {
263         add_j_to_nblast(coul, jj, bLR);
264     }
265     else
266     {
267         add_j_to_nblast(coul_adress, jj, bLR);
268     }
269 }
270 }
271 }
272 @@ -2671,8 +2688,10 @@
273     rvec      box_size, grid_x0, grid_x1;
274     int        i, j, m, ngid;
275     real       min_size, grid_dens;
276 +   real       b_hybrid;
277     int        nsearch;
278     gmx_bool    bGrid;
279 +   gmx_bool    bEnergyGroupCG;
280     char        *ptr;
281     gmx_bool    *i_egp_flags;
282     int         cg_start, cg_end, start, end;
283 @@ -2774,8 +2793,9 @@
284     }
285     debug_gmx();
286
287     if (fr->adress_type == eAdressOff)
288     {
289 +/* chk */
290 +//     if (fr->adress_type == eAdressOff)
291 +//     {
292         if (!fr->ns.bCGlist)
293         {
294             put_in_list = put_in_list_at;
295 @@ -2784,11 +2804,12 @@
296         {
297             put_in_list = put_in_list_cg;
298         }

```

(continues on next page)

(continued from previous page)

```

299 -     }
300 -     else
301 -     {
302 -         put_in_list = put_in_list_address;
303 -     }
304 +//     }
305 +//     else
306 +//     {
307 +//         put_in_list = put_in_list_address;
308 +//     }
309 +/* chk */
310
311     /* Do the core! */
312     if (bGrid)
313 diff -Naur /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/update.cpp /home/mi/
↪ck69giso/gmx-515-lt/src/gromacs/mdlib/update.cpp
314 --- /storage/mi/ck69giso/gromacs-5.1.5/src/gromacs/mdlib/update.cpp      2016-09-07_
↪14:50:21.000000000 +0200
315 +++ /home/mi/ck69giso/gmx-515-lt/src/gromacs/mdlib/update.cpp      2018-12-03_
↪12:19:21.924644082 +0100
316 @@ -67,6 +67,7 @@
317 #include "gromacs/utility/futil.h"
318 #include "gromacs/utility/gmxomp.h"
319 #include "gromacs/utility/smalloc.h"
320 +#include "adress.h"
321
322 /*For debugging, start at v(-dt/2) for velolcity verlet -- uncomment next line */
323 /*#define STARTFROMDT2*/
324 @@ -569,6 +570,8 @@
325     return upd;
326 }
327
328 +/* new */
329 +
330 static void do_update_sd1(gmx_stochd_t *sd,
331                          int start, int nrend, double dt,
332                          rvec accel[], ivec nFreeze[],
333 @@ -579,14 +582,28 @@
334                          int ngtc, real ref_t[],
335                          gmx_bool bDoConstr,
336                          gmx_bool bFirstHalfConstr,
337 -                          gmx_int64_t step, int seed, int* gatindex)
338 +                          gmx_int64_t step, int seed, int* gatindex, real fc, t_
↪forcerec *fr)
339 {
340     gmx_sd_const_t *sdc;
341     gmx_sd_sigma_t *sig;
342 +
343     real          kT;
344     int           gf = 0, ga = 0, gt = 0;
345     real          ism;
346 -    int          n, d;
347 +    int          i, n, d;
348 +
349 +    int          adresstype;
350 +    real          adressr, sqr_dl, dl;
351 +//    real          adressw;

```

(continues on next page)

(continued from previous page)

```

352 +   rvec           *ref, dx, xnew;
353 +
354 +   adresstype      = fr->adresstype;
355 +   adressr         = fr->adress_ex_width;
356 +   //   adressw      = fr->adress_hy_width;
357 +   ref            = &(fr->adress_refs);
358 +
359 +   //real l2 = adressr + adressw;
360 +   real fcl = fc;
361
362   sdc = sd->sdc;
363   sig = sd->sdsig;
364 @@ -621,14 +638,65 @@
365
366   gmrx_rng_cycle_3gaussian_table(step, ng, seed, RND_SEED_UPDATE, rnd);
367
368 +   xnew[0] = x[n][0], xnew[1] = x[n][1], xnew[2] = x[n][2];
369 +   rvec_sub(*ref, xnew, dx);
370 +   sqr_dl = 0.0;
371 +   switch (adresstype)
372 +   {
373 +   case eAdressXSplit:
374 +   /* plane through center of ref, varies in x direction */
375 +   sqr_dl      = dx[0]*dx[0];
376 +   break;
377 +   case eAdressSphere:
378 +   /* point at center of ref, assuming cubic geometry */
379 +   // sqr_dl = 0.0;
380 +   for (i = 0; i < 3; i++)
381 +   {
382 +   sqr_dl      += dx[i]*dx[i];
383 +   }
384 +   break;
385 +   }
386 +   /* Don't thermostat the explicit region */
387 +   dl = sqrt(sqr_dl);
388 +
389 +   if (dl < adressr)
390 +   {
391 +   for (d = 0; d < DIM; d++)
392 +   {
393 +   if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
394 +   ↪ nFreeze[gf][d])
395 +   {
396 +   real sd_V;
397 +   real vn, fn;
398 +   fn      = f[n][d];
399 +   vn      = v[n][d] + (invmass[n]*fn + accel[ga][d])*dt;
400 +   /* Here we include half of the friction+noise
401 +   * update of v into the integration of x.
402 +   */
403 +   xprime[n][d] = x[n][d] + 0.5*(vn + v[n][d])*dt;
404 +   }
405 +   else
406 +   {
407 +   v[n][d]      = 0.0;
408 +   xprime[n][d] = x[n][d];

```

(continues on next page)

(continued from previous page)

```

408 +         }
409 +     }
410 +     continue;
411 +     }
412 +
413 +     for (d = 0; d < DIM; d++)
414 +     {
415 +         if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
↪ nFreeze[gf][d])
416 +         {
417 +             real sd_V, vn;
418 +             real sd_V, vn, fn;
419 +             fn = f[n][d];
420 +             if (fabs(fn)>fc)
421 +             {
422 +                 printf("SD (I) force-cap %e\n", fn);
423 +                 fn = fc*fn/fabs(fn);
424 +             }
425 +
426 +             sd_V = ism*sig[gt].V*rnd[d];
427 +             vn = v[n][d] + (invmass[n]*f[n][d] + accel[ga][d])*dt;
428 +             vn = v[n][d] + (invmass[n]*fn + accel[ga][d])*dt;
429 +             v[n][d] = vn*sdc[gt].em + sd_V;
430 +             /* Here we include half of the friction+noise
431 +              * update of v into the integration of x.
432 +             @@ -663,12 +731,39 @@
433 +             {
434 +                 ga = cACC[n];
435 +             }
436 +
437 +             xnew[0] = x[n][0], xnew[1] = x[n][1], xnew[2] = x[n][2];
438 +             rvec_sub((*ref), xnew, dx);
439 +             sqr_dl = 0.0;
440 +             switch (adresstype)
441 +             {
442 +                 case eAdressXSplit:
443 +                     /* plane through center of ref, varies in x direction */
444 +                     sqr_dl = dx[0]*dx[0];
445 +                     break;
446 +                 case eAdressSphere:
447 +                     /* point at center of ref, assuming cubic geometry */
448 +                     // sqr_dl = 0.0;
449 +                     for (i = 0; i < 3; i++)
450 +                     {
451 +                         sqr_dl += dx[i]*dx[i];
452 +                     }
453 +                     break;
454 +             }
455 +             /* Don't thermostat the explicit region */
456 +             dl = sqrt(sqr_dl);
457 +
458 +             if (dl < adressr)
459 +             {
460 +                 for (d = 0; d < DIM; d++)
461 +                 {
462 +                     if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
↪ nFreeze[gf][d])

```

(continues on next page)

(continued from previous page)

```

463         {
464             v[n][d] = v[n][d] + (im*f[n][d] + accel[ga][d])*dt;
465         +
466         real fn;
467         +
468         fn = f[n][d];
469         +
470         v[n][d] = v[n][d] + (im*fn + accel[ga][d])*dt;
471         xprime[n][d] = x[n][d] + v[n][d]*dt;
472     }
473     else
474     @@ -677,6 +772,31 @@
475         xprime[n][d] = x[n][d];
476     }
477     +
478     continue;
479     +
480     }
481     +
482     if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !nFreeze[gf][d])
483     {
484         +
485         real fn;
486         fn = f[n][d];
487         if (fabs(fn)>fc)
488         {
489             // printf("SD (II) force-cap %e\n", fn);
490             fn = fc*fn/fabs(fn);
491         }
492         +
493         v[n][d] = v[n][d] + (im*fn + accel[ga][d])*dt;
494         +
495         v[n][d] = v[n][d] + (im*f[n][d] + accel[ga][d])*dt;
496         +
497         xprime[n][d] = x[n][d] + v[n][d]*dt;
498     }
499     else
500     {
501         +
502         v[n][d] = 0.0;
503         xprime[n][d] = x[n][d];
504     }
505     +
506     }
507     }
508     +
509     }
510     else
511     @@ -697,8 +817,45 @@
512     gt = cTC[n];
513     +
514     }
515     +
516     xnew[0] = x[n][0], xnew[1] = x[n][1], xnew[2] = x[n][2];
517     +
518     rvec_sub((*ref), xnew, dx);
519     +
520     sqr_dl = 0.0;
521     +
522     switch (adresstype)
523     {
524         +
525         case eAdressXSplit:
526             +
527             /* plane through center of ref, varies in x direction */
528             +
529             sqr_dl = dx[0]*dx[0];

```

(continues on next page)

(continued from previous page)

```

517 +         break;
518 +     case eAdressSphere:
519 +         /* point at center of ref, assuming cubic geometry */
520 +         // sqr_dl = 0.0;
521 +         for (i = 0; i < 3; i++)
522 +         {
523 +             sqr_dl += dx[i]*dx[i];
524 +         }
525 +         break;
526 +     }
527 +     gmx_rng_cycle_3gaussian_table(step, ng, seed, RND_SEED_UPDATE, rnd);
528
529 +     /* Don't thermostat the explicit region */
530 +
531 +     dl = sqrt(sqr_dl);
532 +     //     real l2 = adressr+adressw;
533 +
534 +     if (dl < adressr)
535 +     {
536 +         for (d = 0; d < DIM; d++)
537 +         {
538 +             if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
539 +                 ↪nFreeze[gf][d])
540 +             {
541 +                 real vn;
542 +
543 +                 vn = v[n][d];
544 +                 xprime[n][d] = xprime[n][d] + 0.5*(v[n][d] - vn)*dt;
545 +             }
546 +             continue;
547 +         }
548 +         for (d = 0; d < DIM; d++)
549 +         {
550 +             if ((ptype[n] != eptVSite) && (ptype[n] != eptShell) && !
551 +                 ↪nFreeze[gf][d])
552 +             @@ -1523,7 +1680,8 @@
553 +                 gmx_update_t      upd,
554 +                 gmx_constr_t      constr,
555 +                 gmx_bool          bFirstHalf,
556 +                 gmx_bool          bCalcVir)
557 +                 gmx_bool          bCalcVir,
558 +                 t_forcerc         *fr)
559 +             {
560 +                 gmx_bool          bLastStep, bLog = FALSE, bEner = FALSE, bDoConstr = FALSE;
561 +                 double            dt;
562 +             @@ -1644,6 +1802,8 @@
563 +                 end_th = start + ((nrend-start)*(th+1))/nth;
564 +
565 +                 /* The second part of the SD integration */
566 +                 if (inputrec->bAdress)
567 +                 {
568 +                     do_update_sdl(upd->sd,
569 +                         start_th, end_th, dt,
570 +                         inputrec->opts.acc, inputrec->opts.nFreeze,
571 +                         @@ -1653,7 +1813,23 @@
572 +                         inputrec->opts.ngtc, inputrec->opts.ref_t,

```

(continues on next page)

(continued from previous page)

```

572         bDoConstr, FALSE,
573         step, inputrec->ld_seed,
574 -         DOMAINDECOMP(cr) ? cr->dd->gatindex : NULL);
575 +         DOMAINDECOMP(cr) ? cr->dd->gatindex : NULL,
576 +         inputrec->adress->ex_forcecap, fr);
577 +     }
578 +     else
579 +     {
580 +         do_update_sd1(upd->sd,
581 +             start_th, end_th, dt,
582 +             inputrec->opts.acc, inputrec->opts.nFreeze,
583 +             md->invmass, md->ptype,
584 +             md->cFREEZE, md->cACC, md->cTC,
585 +             state->x, xprime, state->v, force,
586 +             inputrec->opts.ngtc, inputrec->opts.ref_t,
587 +             bDoConstr, FALSE,
588 +             step, inputrec->ld_seed,
589 +             DOMAINDECOMP(cr) ? cr->dd->gatindex : NULL,
590 +             5000., fr);
591 +     }
592 + }
593     inc_nrn(nrn, eNR_UPDATE, homenr);
594     wallcycle_stop(wcycle, ewcUPDATE);
595 @@ -1912,7 +2088,8 @@
596     t_commrec      *cr, /* these shouldn't be here -- need to think_
597     ↪about it */
598     t_nrn          *nrnb,
599     gmx_constr_t   constr,
600 -     t_idef        *idef)
601 +     t_idef        *idef,
602 +     t_forcerec    *fr)
603 {
604     gmx_bool        bNH, bPR, bDoConstr = FALSE;
605     double          dt, alpha;
606 @@ -2031,6 +2208,21 @@
607     break;
608     case (eiSD1):
609         /* With constraints, the SD1 update is done in 2 parts */
610 +     if (inputrec->bAdress)
611 +     {
612 +         do_update_sd1(upd->sd,
613 +             start_th, end_th, dt,
614 +             inputrec->opts.acc, inputrec->opts.nFreeze,
615 +             md->invmass, md->ptype,
616 +             md->cFREEZE, md->cACC, md->cTC,
617 +             state->x, xprime, state->v, force,
618 +             inputrec->opts.ngtc, inputrec->opts.ref_t,
619 +             bDoConstr, TRUE,
620 +             step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
621     ↪gatindex : NULL,
622 +             inputrec->adress->ex_forcecap, fr);
623 +     }
624 +     else
625 +     {
626         do_update_sd1(upd->sd,
            start_th, end_th, dt,
            inputrec->opts.acc, inputrec->opts.nFreeze,

```

(continues on next page)

(continued from previous page)

```

627 @@ -2039,7 +2231,9 @@
628                                     state->x, xprime, state->v, force,
629                                     inputrec->opts.ngtc, inputrec->opts.ref_t,
630                                     bDoConstr, TRUE,
631 -                                     step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
        ↳ gatindex : NULL);
632 +                                     step, inputrec->ld_seed, DOMAINDECOMP(cr) ? cr->dd->
        ↳ gatindex : NULL,
633 +                                     5000., fr);
634 +     }
635         break;
636         case (eiSD2):
637             /* The SD2 update is always done in 2 parts,
638 diff -Naur /storage/mi/ck69giso/gromacs-5.1.5/src/programs/mdrun/md.cpp /home/mi/
        ↳ ck69giso/gmx-515-lt/src/programs/mdrun/md.cpp
639 --- /storage/mi/ck69giso/gromacs-5.1.5/src/programs/mdrun/md.cpp      2017-12-21_
        ↳ 10:16:18.000000000 +0100
640 +++ /home/mi/ck69giso/gmx-515-lt/src/programs/mdrun/md.cpp      2018-12-03 12:14:02.
        ↳ 874757865 +0100
641 @@ -1124,7 +1124,7 @@
642         update_coords(fplog, step, ir, mdatoms, state, fr->bMolPBC,
643                     f, bUpdateDoLR, fr->f_twin, bCalcVir ? &fr->vir_twin_
        ↳ constr : NULL, fcd,
644                     ekind, M, upd, bInitStep, etrtVELOCITY1,
645 -                     cr, nrnb, constr, &top->idef);
646 +                     cr, nrnb, constr, &top->idef, fr);
647
648         if (!bRerunMD || rerun_fr.bV || bForceUpdate)          /* Why is rerun_fr.
        ↳ bV here? Unclear. */
649         {
650 @@ -1133,7 +1133,7 @@
651                                     state, fr->bMolPBC, graph, f,
652                                     &top->idef, shake_vir,
653                                     cr, nrnb, wcycle, upd, constr,
654 -                                     TRUE, bCalcVir);
655 +                                     TRUE, bCalcVir, fr);
656         wallcycle_start(wcycle, ewcUPDATE);
657         if (bCalcVir && bUpdateDoLR && ir->nstcalclr > 1)
658         {
659 @@ -1376,7 +1376,7 @@
660                                     state, fr->bMolPBC, graph, f,
661                                     &top->idef, tmp_vir,
662                                     cr, nrnb, wcycle, upd, constr,
663 -                                     TRUE, bCalcVir);
664 +                                     TRUE, bCalcVir, fr);
665         }
666     }
667     /* ##### START SECOND UPDATE STEP ##### */
668 @@ -1416,7 +1416,7 @@
669         update_coords(fplog, step, ir, mdatoms, state, fr->bMolPBC, f,
670                     bUpdateDoLR, fr->f_twin, bCalcVir ? &fr->vir_twin_
        ↳ constr : NULL, fcd,
671                     ekind, M, upd, FALSE, etrtVELOCITY2,
672 -                     cr, nrnb, constr, &top->idef);
673 +                     cr, nrnb, constr, &top->idef, fr);
674     }
675

```

(continues on next page)

(continued from previous page)

```

676         /* Above, initialize just copies ekinh into ekin,
677 @@ -1438,14 +1438,14 @@
678
679         update_coords(fplog, step, ir, mdatoms, state, fr->bMolPBC, f,
680                     bUpdateDoLR, fr->f_twin, bCalcVir ? &fr->vir_twin_constr :
681         ↪NULL, fcd,
682 -                ekind, M, upd, bInitStep, etrtPOSITION, cr, nrnb, constr, &
683         ↪top->idef);
684 +                ekind, M, upd, bInitStep, etrtPOSITION, cr, nrnb, constr, &
685         ↪top->idef, fr);
686         wallcycle_stop(wcycle, ewcUPDATE);
687
688         update_constraints(fplog, step, &dvd1_constr, ir, mdatoms, state,
689                         fr->bMolPBC, graph, f,
690                         &top->idef, shake_vir,
691                         cr, nrnb, wcycle, upd, constr,
692 -                FALSE, bCalcVir);
693 +                FALSE, bCalcVir, fr);
694
695         if (bCalcVir && bUpdateDoLR && ir->nstcalclr > 1)
696         {
697 @@ -1472,7 +1472,7 @@
698
699         update_coords(fplog, step, ir, mdatoms, state, fr->bMolPBC, f,
700                     bUpdateDoLR, fr->f_twin, bCalcVir ? &fr->vir_twin_
701         ↪constr : NULL, fcd,
702 -                ekind, M, upd, bInitStep, etrtPOSITION, cr, nrnb,
703         ↪constr, &top->idef);
704 +                ekind, M, upd, bInitStep, etrtPOSITION, cr, nrnb,
705         ↪constr, &top->idef, fr);
706         wallcycle_stop(wcycle, ewcUPDATE);
707
708         /* do we need an extra constraint here? just need to copy out of
709         ↪state->v to upd->xp? */
710 @@ -1484,7 +1484,7 @@
711
712         state, fr->bMolPBC, graph, f,
713         &top->idef, tmp_vir,
714         cr, nrnb, wcycle, upd, NULL,
715 -                FALSE, bCalcVir);
716 +                FALSE, bCalcVir, fr);
717
718     }
719     if (bVV)
720     {

```

Downloadable version of patch file

There is also the [abrupt_adress example repository](#) which contains example files to be used as described above.

Software Technical Information

Name Tools for AdResS.

Language C/C++, Python, Fortran, BASH, AWK

Licence Opensource

Application Documentation See GROMACS web page: <http://www.gromacs.org>. For analysis tools and thermo-

dynamic force calculation see VOTCA web page: <http://www.votca.org/home>. Visualize molecular dynamics with **VMD**.

Documentation Tool none

Relevant Training Material none

Tools for AdResS

- *Purpose of Module*
- *Background Information*
- *Source Code*

Purpose of Module

One purpose of our project is to promote GC-AdResS as a method. It is an advanced method, for people with experience, and once the simulation is done there are several properties and checks to consider to make sure that the simulation was successful.

This module provides little tools to make working with AdResS easier. Content:

- 1) how to mask the configuration (output from a full atomistic simulation run) to generate the double resolution configuration.
- 2) Quick and dirty: get the reference coordinate from the GROMACS input file
- 3) Checks for the density (for both geometries currently implemented in GROMACS version 5.1.5)
- 4) Check the temperature on the fly
- 5) A short fortran code to calculate the distribution of the angles in slab-like AdResS simulation.

Background Information

Source Code

Quick and fast data grab from the configuration file:

```
tail conf.gro | awk '(NF==3){print $1/2.0,$2/2.0,$3/2.0}'
```

How to mask the configuration for setting it up for the AdResS simulation. A straight forward way is using **VOTCA** :

```
csg_map --cg mapping_scheme.xml --hybrid --trj input_file.gro --out output_file.gro --
↳top atomistic_run/topol.tpr
```

Check temperature on the fly from the output md.log:

```
#!/bin/bash
grep -A 1 --no-group-separator Lambda md.log | grep -v Step | awk '{print $1}' >
↳mdlogging1
grep -A 1 --no-group-separator Temp md.log | grep -v Temp | awk '{print $2}' >
↳mdlogging2
```

(continues on next page)

(continued from previous page)

```
paste mdlogging1 mdlogging2
paste mdlogging1 mdlogging2 >temperature
rm mdlogging1 mdlogging2
```

Quick grab of the density in the xsplit (slab like) configuration. One way is using the tool from GROMACS:

```
gmx density -d X -f trajectory_file.xtc -sl 50
```

Quick grab of the density in the sphere configuration. We use VOTCA for it:

```
csg_density -- axis r -- rmax 10. --ref [x_ref,y_ref,z_ref] --trj trajectory_input.
↪xtc --top topol.tpr --out SOL.dens.out
```

Collect the p(N) data and combine them in one file. For that we use VMD, it includes a script called topotools, which is used to handle the trajectory. This script can be run from the command line directly:

```
vmd -dispdev text -e extract_coord.tcl
grep -B1 "Frame" WCG.xyz > a
sed '/Frame/ {$!N;d;}' a > column2
grep -B0 "Frame" WCG.xyz > a
sed -i s/Frame// a
sed -i s/--// a
sed -i s/:// a
sed '/^$/d' a > column1
paste column1 column2|awk '{print $1, $2}' > dat.3nm.pn.WCG.dat
```

And the corresponding extract_coord.tcl:

```
package require topotools 1.2
mol new conf.gro
mol addfile traj_comp.xtc type xtc waitfor all first 0 last -1 step 1
topo writevarxyz WCG.xyz selmod "name WCG and (x>285 and x<315)"
exit
```

All the small scripts are available as files:

analysis tools source code

Software Technical Information

Name Velocity-Velocity autocorrelation function for AdResS.

Language C/C++

Licence Opensource

Documentation Tool none

Application Documentation <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Relevant Training Material <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Velocity-Velocity autocorrelation function for AdResS

- *Purpose of Module*
- *Source Code*

Purpose of Module

One purpose of our project is to promote GC-AdResS as a method. It is an advanced method, for people with experience, and once the simulation is done there are several properties and checks to consider to make sure that the simulation was successful.

This module provides the code to run a velocity velocity autocorrelation function on the current geometries available in the Abrupt AdResS implementation. The paper [Ref.](#) describes the correlation functions and why they can be used in AdResS. This code is based on that theory and has been developed to check the dynamics of the local thermostat GC-AdResS simulations presented in the paper cited above.

Source Code

Files are stored here: <https://gitlab.e-cam2020.eu/krekeler/analyze.energy>. The source code for the velocity autocorrelation function can be found here: https://gitlab.e-cam2020.eu/krekeler/analyze.energy/tree/master/app/cal_vel_acc_adr.cpp

The installation instruction can be found <https://gitlab.e-cam2020.eu:10443/krekeler/analyze.energy#installation-instructions>.

Usage:

```
cal_vel_acc_adr:
options:
-h , "print this message"
-b start time
-e end time (=number of MD steps)
--x0 lower bound of the interval
--x1 upper bound of the interval (--x1 0, use the whole box = atomistic)
--frame length of correlation
--acc breaks
--total number of frames
--tf Output Frequency (=Delta_t)
-m type of simulation to analyze (adress or atom)
-f input .xtc file
-o output file
```

It is important to have the XDR files and setup in the same directory as they have to be specified in the Makefile. The XDR files can be found via the GROMACS web page, see http://www.gromacs.org/Developer_Zone/Programming_Guide/XTC_Library or <ftp://ftp.gromacs.org/pub/contrib/xdrfile-1.1.4.tar.gz>.

Software Technical Information

Name Dipole-Dipole autocorrelation function for AdResS.

Language C/C++

Licence Open source (no specific licence provided)

Documentation Tool none

Application Documentation <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Relevant Training Material <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Dipole-Dipole autocorrelation function for AdResS

- *Purpose of Module*
- *Background Information*
- *Source Code*

Purpose of Module

One purpose of our project is to promote GC-AdResS as a method. It is an advanced method, for people with experience, and once the simulation is done there are several properties and checks to consider to make sure that the simulation was successful.

This module provides the code to run a dipole dipole autocorrelation function on the current geometries available in the Abrupt AdResS implementation. The paper [Ref.](#) describes the correlation functions and why they can be used in AdResS. This code is based on that theory and has been developed to check the dynamics of the local thermostat GC-AdResS simulations presented in the paper cited above.

Background Information

See *Abrupt GC-AdResS: A new and more general implementation*

Source Code

Files are stored under <https://gitlab.e-cam2020.eu/krekeler/analyze.energy>. The source code for the dipole autocorrelation function can be found at https://gitlab.e-cam2020.eu/krekeler/analyze.energy/blob/master/app/cal_dacf.cpp.

The installation instructions are given at <https://gitlab.e-cam2020.eu:10443/krekeler/analyze.energy#installation-instructions>.

Usage:

```
cal_dacf

options:
-h  print this message
-b  start time
-e  end time (=number of MD steps)
--x0 lower bound of the interval
--x1 upper bound of the interval (--x1 0, use the whole box = atomistic)
--frame length of correlation
--q0 charge on oxygen
--q1 charge on hydrogen
--acc breaks
--total number of frames
```

(continues on next page)

(continued from previous page)

```
--tf Output Frequency (=Delta_t)
-m type of simulation to analyze (address or atom)
-f input .xtc file
-o output file
```

It is important to have the XDR files and setup in the same directory as they have to be specified in the Makefile. The XDR files can be found via the GROMACS web page, see http://www.gromacs.org/Developer_Zone/Programming_Guide/XTC_Library or <ftp://ftp.gromacs.org/pub/contrib/xdrfile-1.1.4.tar.gz>.

Software Technical Information

Name Energy (AT)/Energy(interface) ratio: Necessary condition for AdResS simulations.

Language C/C++

Licence none

Documentation Tool none

Application Documentation <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Relevant Training Material <http://www.ks.uiuc.edu/Research/vmd/current/docs.html>

Energy (AT)/Energy(interface) ratio: Necessary condition for AdResS simulations

- *Purpose of Module*
- *Running the code:*
- *Source Code*

Purpose of Module

One purpose of our project is to promote GC-AdResS as a method. It is an advanced method, thus for people with experience, and once the simulation is done there are several properties and checks to consider to make sure that the simulation was successful.

This module provides the code to check one very important quantity, the interaction energy in the atomistic region compared with the interaction energy in the comparable subregion in a full atomistic simulation. The difference between those energies is the interaction energy at the interface, which has to be much smaller than the interaction energy in the atomistic region.

The theory is described in [Ref.](#). This legacy code is based on that theory and has been developed to check the energy of the local thermostat GC-AdResS simulations presented in the paper cited above.

Running the code:

The executable is called `energy`. The options on how to run the analysis:

```
-h : help message
-b : start frame
-e : end frame
-n : number of regions
-x0 : start transition region
-x1 : end transition region (if x1 == 0, use the whole box)
-q1 : charge on oxygen
-q2 : charge on hydrogen
-sig : sigma
-eps : eps
-beads : no. of beads in one ring polymer
-c : cut off radius for neighbor list search
-f : the input .xtc file (default: traj.xtc)
-o : the output file
```

Source Code

Files are stored under <https://gitlab.e-cam2020.eu/krekeler/analyze.energy>. The source code for the energy calculation can be found at <https://gitlab.e-cam2020.eu/krekeler/analyze.energy/blob/master/app/energy.cpp>. The installation instruction are available at <https://gitlab.e-cam2020.eu:10443/krekeler/analyze.energy#installation-instructions>.

It is important to have the XDR files and setup in the same directory as they have to be specified in the Makefile. The XDR files can be found via the GROMACS web page, see http://www.gromacs.org/Developer_Zone/Programming_Guide/XTC_Library or <ftp://ftp.gromacs.org/pub/contrib/xdrfile-1.1.4.tar.gz>.

4.3.6 ALL (A Load-balancing Library)

Most modern parallelized (classical) particle simulation programs are based on a spatial decomposition method as an underlying parallel algorithm: different processors administrate different spatial regions of the simulation domain and keep track of those particles that are located in their respective region. Processors exchange information

- in order to compute interactions between particles located on different processors
- to exchange particles that have moved to a region administrated by a different processor.

This implies that the workload of a given processor is very much determined by its number of particles, or, more precisely, by the number of interactions that are to be evaluated within its spatial region.

Certain systems of high physical and practical interest (e.g. condensing fluids) dynamically develop into a state where the distribution of particles becomes spatially inhomogeneous. Unless special care is being taken, this results in a substantially inhomogeneous distribution of the processors' workload. Since the work usually has to be synchronized between the processors, the runtime is determined by the slowest processor (i.e. the one with highest workload). In the extreme case, this means that a large fraction of the processors is idle during these waiting times. This problem becomes particularly severe if one aims at strong scaling, where the number of processors is increased at constant problem size: Every processor administrates smaller and smaller regions and therefore inhomogeneities will become more and more pronounced. This will eventually saturate the scalability of a given problem, already at a processor number that is still so small that communication overhead remains negligible.

The solution to this problem is the inclusion of dynamic load balancing techniques. These methods redistribute the workload among the processors, by lowering the load of the most busy cores and enhancing the load of the most idle ones. Fortunately, several successful techniques are known already to put this strategy into practice. Nevertheless, dynamic load balancing that is both efficient and widely applicable implies highly non-trivial coding work. Therefore it has not yet been implemented in a number of important codes of the E-CAM community, e.g. DL_Meso, DL_Poly, Espresso, Espresso++, to name a few. Other codes (e.g. LAMMPS) have implemented somewhat simpler schemes, which however might turn out to lack sufficient flexibility to accommodate all important cases. Therefore,

the ALL library was created in the context of an Extended Software Development Workshop (ESDW) within E-CAM (see [ALL ESDW event details](#)), where code developers of CECAM community codes were invited together with E-CAM postdocs, to work on the implementation of load balancing strategies. The goal of this activity was to increase the scalability of these applications to a larger number of cores on HPC systems, for spatially inhomogeneous systems, and thus to reduce the time-to-solution of the applications.

Software Technical Information

Name A Load Balancing Library (ALL)

Language C++, Fortran interfaces available

Licence [BSD 3-Clause](#)

Documentation Tool No tool used in source code, repo documentation written in [Markdown](#)

Application Documentation See [ALL repository](#)

Relevant Training Material None available

Software Module Developed by Rene Halver

Polymer melt test is provided by Dr. Horacio V. Guzman

Module Committed by Dr. Horacio V. Guzman

ALL Tensor-Product method

- *Purpose of Module*
- *Background Information*
- *ALL: Building and Testing*
- *Source Code*

A Load-Balancing Library (ALL) library aims to provide an easy and portable way to include dynamic domain-based load balancing into particle based simulation codes. The library is developed in the Simulation Laboratory Molecular Systems of the Juelich Supercomputing Centre at Forschungszentrum Juelich.

Purpose of Module

This module provides an additional method to the [ALL library](#) , up-to-date descriptions of the methods in the library can be found in the [ALL README file](#).

For the Tensor-Product method, the work on all processes (subdomains) is reduced over the cartesian planes in the systems. This work is then equalized by adjusting the borders of the cartesian planes.

Background Information

See [ALL \(A Load-balancing Library\)](#) for details.

ALL: Building and Testing

ALL is a C++ header only library using template programming, strictly speaking there is no need to install the library, you simply include the header files in your application. In order to provide examples, ALL uses the [CMake](#) build system, specific build and installation requirements can be found in the [ALL README file](#). If you wish to use/test the topological mesh scheme, you will need an MPI-enabled installation of the [VTK](#) package.

To build ALL, begin in the root directory of the package and use

```
export ALL_INSTALLATION=/path/to/my/loadbalancing/install
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$ALL_INSTALLATION -DCM_ALL_VTK_OUTPUT=ON -DCM_ALL_
↪VORONOI=ON
make -j
make install
cd ..
```

This will create an installation of ALL in the path pointed to by `ALL_INSTALLATION`. `ALL_test` (in the `bin` folder) is the binary that performs the tests. If you omit the option `-DCM_ALL_VTK_OUTPUT=ON` you will not require the VTK dependency (but cannot use the unstructured mesh method).

In the `example/jube/input` subdirectory there are 3 test data sets available, namely:

1. Simple Wye-shape biosystem;
2. Heterogeneous polymer melt and
3. A rotated version of the Wye-shaped biosystem.

These data sets are in raw ascii format and need to be translated into a format that can be consumed by `ALL_test`. A utility `ASCII2MPIBIN` is provided to do the conversion, with the command line options:

```
ASCII2MPIBIN <in_file (ASCII)> <out_file (binary)> <n_x> <n_y> <n_z>
```

where `n_x`, `n_y`, `n_z` are the number of (MPI) processes (in the X, Y and Z directions) that will be used.

`ALL_test` takes a number of options,

```
ALL_test <Method> <Number of iterations> <gamma> <weighted> <input file> <system_
↪size: x, y, z> <domain layout: x, y, z>
```

Method (integer) is the load-balancing scheme to use of which there are 5 options:

```
0 : Tensor
1 : Staggered
2 : Unstructured
3 : Voronoi
4 : Histogram
```

, `gamma` (double) is a relaxation which controls the convergence of the load-balancing methods, `weighted` (boolean) indicates whether points should be assigned a weight. The system size and domain layout are provided in the output of the call to `ASCII2MPIBIN`.

An example execution using the polymer melt data set on 125 processors looks like

```
ASCII2MPIBIN globalBlockCoordsPolymer.txt input.bin 5 5 5
mpirun -n 125 ALL_test 0 50 8.0 0 input.bin 80 80 450 5 5 5
```


Source Code

The implementation of the Tensor-Product method in ALL can be found in [ALL_Tensor.hpp](#).

The source code to the ALL library is available as a git repository at <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing>. To obtain a copy of the repository you can use

```
git clone https://gitlab.version.fz-juelich.de/SLMS/loadbalancing.git
```

However, please note that the source code is currently under embargo until an associated paper is published, if you would like to be obtain a copy of the code, please contact Prof. Godehard Sutmann at g.sutmann@fz-juelich.de.

Software Technical Information

Name A Load Balancing Library (ALL)

Language C++, Fortran interfaces available

Licence [BSD 3-Clause](#)

Documentation Tool No tool used in source code, repo documentation written in [Markdown](#)

Application Documentation See [ALL repository](#)

Relevant Training Material None available

Software Module Developed by Rene Halver

ALL Staggered Grid Method

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

A Load-Balancing Library (ALL) library aims to provide an easy and portable way to include dynamic domain-based load balancing into particle based simulation codes. The library is developed in the Simulation Laboratory Molecular Systems of the Juelich Supercomputing Centre at Forschungszentrum Juelich.

Purpose of Module

This module provides an additional method to the [ALL library](#), up-to-date descriptions of the methods in the library can be found in the [ALL README file](#).

In the *staggered-grid* scheme, a 3-step hierarchical approach is applied, where:

- work over the cartesian planes is reduced, before the borders of these planes are adjusted;
- in each of the cartesian planes the work is reduced for each cartesian column. These columns are then adjusted to each other to homogenize the work in each column;
- the work between neighboring domains in each column is adjusted.

Each adjustment is done locally with the neighboring planes, columns or domains by adjusting the adjacent boundaries.

Background Information

See *ALL (A Load-balancing Library)* for details.

Building and Testing

See *ALL: Building and Testing* for details.

Source Code

The implementation of the method in ALL can be found in *ALL_Staggered.hpp*.

The source code to the ALL library is available as a git repository at <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing> . To obtain a copy of the repository you can use

```
git clone https://gitlab.version.fz-juelich.de/SLMS/loadbalancing.git
```

However, please note that the source code is currently under embargo until an associated paper is published, if you would like to be obtain a copy of the code, please contact Prof. Godehard Sutmann at g.sutmann@fz-juelich.de.

Software Technical Information

Name A Load Balancing Library (ALL)

Language C++, Fortran interfaces available

Licence [BSD 3-Clause](#)

Documentation Tool No tool used in source code, repo documentation written in [Markdown](#)

Application Documentation See [ALL repository](#)

Relevant Training Material None available

Software Module Developed by Rene Halver

ALL Unstructured Mesh Method

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

A Load-Balancing Library (ALL) library aims to provide an easy and portable way to include dynamic domain-based load balancing into particle based simulation codes. The library is developed in the Simulation Laboratory Molecular Systems of the Juelich Supercomputing Centre at Forschungszentrum Juelich.

Purpose of Module

This module provides an additional method to the [ALL](#) library, up-to-date descriptions of the methods in the library can be found in the [ALL README](#) file.

In contrast to *ALL Tensor-Product method* and *ALL Staggered Grid Method*, the *unstructured mesh* method adjusts domains not by moving boundaries but vertices, i.e. corner points, of domains. For each vertex a force, based on the differences in work of the neighboring domains, is computed and the vertex is shifted in a way to equalize the work between these neighboring domains.

Background Information

See *ALL (A Load-balancing Library)* for details.

Building and Testing

See *ALL: Building and Testing* for details.

Source Code

The implementation of the method in ALL can be found in [ALL_Unstructured.hpp](#).

The source code to the ALL library is available as a git repository at <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing>. To obtain a copy of the repository you can use

```
git clone https://gitlab.version.fz-juelich.de/SLMS/loadbalancing.git
```

However, please note that the source code is currently under embargo until an associated paper is published, if you would like to be obtain a copy of the code, please contact Prof. Godehard Sutmann at g.sutmann@fz-juelich.de.

Software Technical Information

Name A Load Balancing Library (ALL)

Language C++, Fortran interfaces available

Licence [BSD 3-Clause](#)

Documentation Tool No tool used in source code, repo documentation written in [Markdown](#)

Application Documentation See [ALL repository](#)

Relevant Training Material None available

Software Module Developed by Rene Halver

ALL Voronoi Mesh Method

- *Purpose of Module*

- *Background Information*
- *Building and Testing*
- *Source Code*

A Load-Balancing Library (ALL) library aims to provide an easy and portable way to include dynamic domain-based load balancing into particle based simulation codes. The library is developed in the Simulation Laboratory Molecular Systems of the Juelich Supercomputing Centre at Forschungszentrum Juelich.

Purpose of Module

This module provides an additional method to the [ALL library](#), up-to-date descriptions of the methods in the library can be found in the [ALL README file](#).

Similar to the topological mesh method ([ALL Unstructured Mesh Method](#)), the *Voronoi mesh* method computes a force, based on work differences. In contrast to the topological mesh method, the force acts on a Voronoi point rather than a vertex, i.e. a point defining a Voronoi cell, which describes the domain. Consequently, the number of neighbors is not a conserved quantity, i.e. the topology may change over time. ALL uses the Voro++ library published by the Lawrence Berkeley Laboratory for the generation of the Voronoi mesh.

Background Information

See [ALL \(A Load-balancing Library\)](#) for details.

Building and Testing

See [ALL: Building and Testing](#) for details.

Source Code

The implementation of the method in ALL can be found in [ALL_Voronoi.hpp](#).

The source code to the ALL library is available as a git repository at <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing>. To obtain a copy of the repository you can use

```
git clone https://gitlab.version.fz-juelich.de/SLMS/loadbalancing.git
```

However, please note that the source code is currently under embargo until an associated paper is published, if you would like to be obtain a copy of the code, please contact Prof. Godehard Sutmann at g.sutmann@fz-juelich.de.

Software Technical Information

Name A Load Balancing Library (ALL)

Language C++, Fortran interfaces available

Licence [BSD 3-Clause](#)

Documentation Tool No tool used in source code, repo documentation written in [Markdown](#)

Application Documentation See [ALL repository](#)

Relevant Training Material None available

Software Module Developed by Rene Halver

ALL Histogram-based Staggered Grid Method

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

A Load-Balancing Library (ALL) library aims to provide an easy and portable way to include dynamic domain-based load balancing into particle based simulation codes. The library is developed in the Simulation Laboratory Molecular Systems of the Juelich Supercomputing Centre at Forschungszentrum Juelich.

Purpose of Module

This module provides an additional method to the [ALL library](#), up-to-date descriptions of the methods in the library can be found in the [ALL README file](#).

The *histogram-based staggered-grid* scheme results in the same grid as the staggered-grid scheme (see [ALL Staggered Grid Method](#)), this scheme uses the cumulative work function in each of the three cartesian directions in order to generate this grid. Using histograms and the previously defined distribution of process domains in a cartesian grid, this scheme generates in three steps a staggered-grid result, in which the work is distributed as evenly as the resolution of the underlying histogram allows. In contrast to the other schemes this scheme depends on a global exchange of work between processes.

Background Information

See [ALL \(A Load-balancing Library\)](#) for details.

Building and Testing

See [ALL: Building and Testing](#) for details.

Source Code

The implementation of the method in ALL can be found in [ALL_Histogram.hpp](#).

The source code to the ALL library is available as a git repository at <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing>. To obtain a copy of the repository you can use

```
git clone https://gitlab.version.fz-juelich.de/SLMS/loadbalancing.git
```

However, please note that the source code is currently under embargo until an associated paper is published, if you would like to be obtain a copy of the code, please contact Prof. Godehard Sutmann at g.sutmann@fz-juelich.de.

Software Technical Information

Name A Load Balancing Library (ALL) - C++ interface

Language C++, Fortran interfaces available

Licence BSD 3-Clause

Documentation Tool In source provided by Doxygen, additional using Sphinx

Application Documentation <http://slms.pages.jsc.fz-juelich.de/websites/all-website/sphinx/api/ALL.html>

Relevant Training Material Webinar (YT)

Software Module Developed by Rene Halver, Stephan Schulz

ALL C++ interface

- *Purpose of Module*
- *Background Information*
- *Technical Details*
- *Building*
- *Source Code*

Since C++ becomes more common in the HPC environment, therefore the default interface for ALL is written in that language. The library uses class inheritance to administrate the different load-balancing methods. Every necessary functionality to use the library is provided by the interface. In addition there is a Fortran interface provided (description in module *ALL Fortran interface*).

Purpose of Module

This module is necessary for all users to couple their code with the ALL library.

It is currently used in all projects, which already include the ALL library to their code and are based on C and C++.

Background Information

The interface is part of ALL which can be found at <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing> in the `include` subdirectory. It is called `ALL.hpp`

Technical Details

The C++ interface is a header-based solution, as it uses C++ templating capabilities to support different types of floating point numbers to describe domain borders and work loads. Internally class inheritance is used to administrate the different included load-balancing methods. This should provide an easy way to include future additions of new methods into the library.

Building

General installation instructions for ALL can be found in the [README.md of the ALL repository](#) (which is also distributed with each release).

As the C++ interface is the default interface of the ALL library, there is no need to explicitly enable it. The interface provides functions to create an object which handles the computations of new boundaries based on provided sets of domain borders and work loads. In addition, if the library is compiled with VTK support (requires `CM_ALL_VTK_OUTPUT` in CMake), functionality to create VTK descriptions of the domain structure is provided (for all methods working on orthogonal domains).

Source Code

The source code for this interface consists of: [include/ALL.hpp](#)

Software Technical Information

Name A Load Balancing Library (ALL)

Language C++, Fortran interfaces available

Licence [BSD 3-Clause](#)

Documentation Tool In source provided by Doxygen, additional using Sphinx

Application Documentation http://slms.pages.jsc.fz-juelich.de/websites/all-website/sphinx/api/ALL_module.html

Relevant Training Material [Webinar \(YT\)](#)

Software Module Developed by Stephan Schulz

ALL Fortran interface

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

There is still a lot of Fortran code in use and the low entry barrier to the language makes it an easy choice for beginning scientific programmers. To be able to utilize the features of the library in Fortran code this interface is provided. It provides basically the same functionality as the C++ interface.

Purpose of Module

This module is necessary for any Fortran developers trying to use this library.

It is currently in use by the Fortran Multi Particle Method written for the thesis of Stephan Schulz. This application of the interface is documented in the module [MPM Integration](#).

Background Information

The interface is part of ALL which can be found at <https://gitlab.version.fz-juelich.de/SLMS/loadbalancing> in the `src` subdirectory. It is called `ALL_module.F90`. Additionally, an internal C wrapper is used for the C++ class, since Fortran can only interoperate with C.

Building and Testing

The Fortran module must be explicitly enabled when building the library. This is done by setting the CMake variable `CM_ALL_FORTRAN` to `ON`. The use of the `mpi_f08` module can also be enabled with `CM_ALL_USE_F08`. Then the new MPI derived types can be used directly. The requisite compilers and MPI implementations must be present. Also note, that the ALL module must be compiled by the same Fortran compiler as the application (and MPI implementation if any MPI module is used). More information is available in the libraries documentation in the code's repository in `docs/Install.rst`.

Source Code

The source code for this interface consists of the C wrapper `src/ALL_fortran.cpp` and the Fortran module `ALL/src/ALL_module.F90`.

Software Technical Information

Name A Load Balancing Library (ALL)/GMPM-PoC

Language Fortran/C/C++

Licence [BSD 3-Clause](#)

Documentation Tool In source documentation using Doxygen, additional man pages and plain text

Application Documentation Non public/in repo

Relevant Training Material [Webinar \(YT\)](#)

Software Module Developed by Stephan Schulz

MPM Integration

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The material point method (MPM) is used to simulate continuous matter and is especially suited for the simulation of large deformations. Once large deformation are present, a dynamic load balancing solution is sensible to efficiently simulate large systems. Even if the initial work distribution is good, it is very often the case, that this distribution is much less so during the simulation run itself. The load balancing library ALL provides an easy plug and play solution to this problem and this module describes the details in how the library is integrated. Thanks to the good load balancing provided by the library larger systems can be simulated with less computational cost.

Purpose of Module

This module shows the straight forwardness of including the load balancing library into already existing code. Depending on the simulation code additional precautions must be taken and those needed for the MPM simulation code are presented here. The prerequisites for the simulation code are also shown. Looking at these will help determine whether a simulation code is particularly suited for integrating ALL or if some further work is needed when integrating.

This module also shows a real world application of the Fortran interface provided with [ALL](#) (documented in [ALL Fortran interface](#)).

The MPM simulation code with integrated ALL is used by Stephan Schulz in his thesis.

Background Information

The load balancing library ALL is integrated into the material point method simulation code GMPM-PoC, which is written by Stephan Schulz during his thesis. The simulation code will be released to the public in the future.

Certain aspects of the simulation code require additional treatment of the library, or additional features of the library. First, the open boundaries of the simulation code require continuous updates of the outer domain boundaries of the boundary domains. The system extent is adapted to the particle's bounding box each time step. This also means, the geometry generated in the last balance step by the library cannot be used directly. It is therefore saved by the simulation code, adapted to the new system extent and then given to the library as the basis for the new geometry.

The communication is based on grid halos and only accommodates nearest neighbor communication. This causes the minimum domain size to be the width of exactly this halo. The library supports this feature and only the aforementioned outer domain bounds must be checked for compliance with the minimum size. The other domain boundaries are automatically sufficiently large due to the library.

And lastly, the particle communication at the end of each time step also only accounts for nearest neighbor communication. This means, that a domain's boundary must not change so much, that it needs to retrieve particles from a process that is not its nearest neighbor. Due to the way the library moves boundaries in the staggered grid and tensor approaches, this is also already guaranteed to be true. There is always an overlap between the old domain's volume and the new domain's.

However, the library also has a few requirements of the simulation code. Due to changing domains, particles must be able to be communicated across processes, which is implemented for all particle codes with moving particles. Depending on the load balancing method this communication may be more involved. In the case of the tensor method the topology does not change and every process has the same 26 neighbors during the entire simulation. If, however, the staggered grid approach is used, the communication must also handle changing number of neighbors and determine where they are and what regions they belong to. For example it is common, that one half of a boundary must be communicated to one process and the other to a different one. So the fixed relationship between boundaries and neighbors is broken up. The GMPM-PoC code was already designed with such a communication scheme in mind and provided the necessary flexibility to simply enable the staggered grid method after fixing a few communication bugs.

Building and Testing

To build the code just run `make LB=ALL` and everything should be build automatically including dependencies. Make sure the correct compiler are found in the path and if you want to use Intel compilers you need to set `COMPILER=INTEL` as well. The normal caveats and required modules for some HPC systems are the described in the main code's README.

Source Code

The main changes are the replacement of the original domain decomposition function which used to equi partition the system extent. Now, ALL is called to update the domain geometry.

```

1  ! The following additional functions were used:
2  !
3  ! Additional information:
4  ! - LB_METHOD_PRE is set by the preprocessor to ALL_STAGGERED or ALL_TENSOR.
5  ! - The domain_bounds_old will only be used during initialisation for the
6  !   initial domain configuration.
7  ! - ``this_image()`` returns the current image index, i.e. current MPI rank+1.
8  ! - The work is estimated using ``lb_estimate_work`` which takes the current
9  !   domain size and number of particles as arguments.
10
11  function domain_decomposition_jall(bounds, dh, num_images3, domain_bounds_old,
↪work, output) result(domain_bounds)
12      use ISO_C_BINDING
13      type(boundingBox), intent(in) :: bounds !< simulation bounds
14      real(kind = real_kind), intent(in) :: dh !< grid width
15      integer, dimension(3), intent(in) :: num_images3 !< the 1 indexed number of
↪images in 3D
16      type(boundingBox_aligned), intent(in) :: domain_bounds_old !< current domain
↪bounds
17      real(real_kind), intent(in) :: work !< work of this domain
18      logical, intent(in) :: output !< output domain bounds to `vtk_outline`
↪directory
19      type(boundingBox_aligned) :: domain_bounds
20      type(boundingBox), save :: domain_old
21      type(ALL_t), save :: jall ! ALL object which is initialized once
22      real(kind = real_kind), dimension(3,2) :: verts
23      integer, dimension(3), save :: this_image3 ! the 1 indexed image number in 3D
24      logical, save :: first_run = .true.
25      integer, save :: step = 1
26      logical, dimension(2,3), save :: domain_at_sim_bound ! true if domain is at
↪the lower/upper simulation boundary
27      real(kind = real_kind), dimension(3), save :: min_size
28      integer(c_int), parameter :: LB_METHOD = LB_METHOD_PRE
29      character(len=ALL_ERROR_LENGTH) :: error_msg
30      if(first_run) then
31          ! calculate this_image3
32          block
33              integer :: x,y,z, cnt
34              cnt = 1
35              do z=1,num_images3(3)
36                  do y=1,num_images3(2)
37                      do x=1,num_images3(1)
38                          if(this_image()==cnt) this_image3 = (/ x,y,z /)
39                          cnt = cnt + 1
40                      enddo
41                  enddo
42              enddo
43          end block
44          call jall%init(LB_METHOD,3,4.0d0)
45          call jall%set_proc_grid_params(this_image3-1, num_images3)
46          call jall%set_proc_tag(this_image())
47          call jall%set_communicator(MPI_COMM_WORLD)

```

(continues on next page)

(continued from previous page)

```

48     min_size(:) = (abs(Rcont_min)+abs(Rcont_max))*dh
49     call jall%set_min_domain_size(min_size)
50     domain_old%bounds_unaligned = domain_bounds_old%bounds_aligned
51     domain_at_sim_bound(1,:) = this_image3==1 ! first image in a direction is
↳ automatically at sim bound
52     domain_at_sim_bound(2,:) = this_image3==num_images3 ! last image likewise
↳ at sim bound
53     call jall%setup()
54     endif
55     call jall%set_work(real(work,real_kind))
56     !! The `domain_old` bounds are not the actual domain bounds, which
57     !! are aligned to grid widths, but what we got from the previous
58     !! iteration of load balancing. However, the simulation boundaries are
59     !! unchanged by the load balancing.
60     block
61         type(boundingBox_aligned) :: aligned_bnds
62         real(kind = real_kind), dimension(3) :: lowest_upper_bound, highest_
↳ lower_bound
63         !> Align the simulation boundaries to the grid and add an additional
64         !! grid width on the top. These may be used instead of our current
65         !! bounds, so they should align properly on the upper bound, if we
66         !! are a simulation boundary. If the simulation bounds have not
67         !! changed they should still coincide with the domain bounds.
68         aligned_bnds%bounds_aligned = floor(bounds%bounds_unaligned/dh)*dh
69         aligned_bnds%bounds_aligned(2,:) = aligned_bnds%bounds_aligned(2,:) + dh
70         !> To make sure, the shrinking domain is still always large enough
71         !! and in particular is not shrunk into the neighbouring domain.
72         !! This can happen if the bounding box is not present in the current
73         !! domain, so the outer bound is moved across the inner bound. This
74         !! must be avoided at all cost. Additionally, we also need to ensure
75         !! the minimum domain width. Also, the outer bound of all boundary
76         !! domains, must be the same. To achieve this, the outermost inner
77         !! bound is calculated in each direction. This then allows us to
78         !! compute the innermost position any outer bound may have to still
79         !! be the required distance from every next inner bound.
80         ! For the lowest domains:
81         lowest_upper_bound = comm_co_min_f(domain_old%bounds_unaligned(2,:))
82         aligned_bnds%bounds_aligned(1,:) = min(lowest_upper_bound-min_size,
↳ aligned_bnds%bounds_aligned(1,:))
83         ! For the highest domains:
84         highest_lower_bound = comm_co_max_f(domain_old%bounds_unaligned(1,:))
85         aligned_bnds%bounds_aligned(2,:) = max(highest_lower_bound+min_size,
↳ aligned_bnds%bounds_aligned(2,:))
86         ! And now set the boundary domains outer bounds to the new, fixed bounds
87         where(domain_at_sim_bound)
88             domain_old%bounds_unaligned = aligned_bnds%bounds_aligned
89         end where
90     end block
91     !> Make sure that the old domain bounds are sensible. we are only
92     !! updating them, based in the previous value. This also means
93     !! the first call must already contain a sensible approximation
94     !! (the equidistant (geometric) distribution suffices for that).
95     verts = transpose(domain_old%bounds_unaligned)
96     call jall%set_vertices(verts)
97     call jall%balance()
98     call jall%get_result_vertices(verts)
99     domain_bounds%bounds_aligned = transpose(verts)

```

(continues on next page)

(continued from previous page)

```

100     domain_old%bounds_unaligned = domain_bounds%bounds_aligned
101     domain_bounds%bounds_aligned = nint(domain_bounds%bounds_aligned/dh)*dh
102     if(output) then
103         call ALL_reset_error()
104         call jall%print_vtk_outlines(step)
105         if(ALL_error() /= 0) then
106             error_msg = ALL_error_description()
107             print*, "Error in ALL detected:"
108             print*, error_msg
109         endif
110     endif
111     first_run = .false.
112     step = step + 1
113     call assert_domain_width(domain_bounds, dh)
114 end function
115
116 !> Estimate local work
117 function lb_estimate_work(n_part, domain_bounds_old) result(work)
118     integer, intent(in) :: n_part !< number of particles of this domain
119     type(boundingBox_aligned), intent(in) :: domain_bounds_old !< domain bounds
120     real(real_kind) :: work
121     real(real_kind), parameter :: beta = 0.128 ! empirically determined
122     work = n_part + beta*product( domain_bounds_old%bounds_aligned(2,:)-domain_
123     ↪ bounds_old%bounds_aligned(1,:) )/grid%dh**3
124 end function

```

To include the library and its VTK dependency into the existing make build system, the following snippets were used. This builds a ‘minimal’ VTK and links ALL against this. During the linking of the main simulation code VTK is linked using \$ (VTK_LIB) where the order is very important. The calling of make in this Makefile is deprecated and should be replaced by appropriate calls to cmake --build and cmake --install.

```

1  MJOBS ?= $(shell getconf _NPROCESSORS_ONLN)
2  JUELICH_ALL_INCLUDE := external/juelich_all_build/include/modules
3  JUELICH_ALL_LIB := external/juelich_all_build/lib/libALL_fortran.a external/juelich_
4  ↪ all_build/lib/libALL.a
5
6  VTK_LIB := $(subst lib/,external/vtk_build/lib/, lib/libvtkFiltersProgrammable-7.1.a_
7  ↪ lib/libvtkIOParallelXML-7.1.a lib/libvtkIOXML-7.1.a lib/libvtkIOXMLParser-7.1.a lib/
8  ↪ lib/vtkexpat-7.1.a lib/libvtkParallelMPI-7.1.a lib/libvtkParallelCore-7.1.a lib/
9  ↪ lib/vtkIOLegacy-7.1.a lib/libvtkIOCore-7.1.a lib/libvtkCommonExecutionModel-7.1.a_
10 ↪ lib/libvtkCommonDataModel-7.1.a lib/libvtkCommonTransforms-7.1.a lib/
11 ↪ lib/vtkCommonMisc-7.1.a lib/libvtkCommonMath-7.1.a lib/libvtkCommonSystem-7.1.a lib/
12 ↪ lib/vtkCommonCore-7.1.a lib/libvtksys-7.1.a -ldl -lpthread lib/libvtkzlib-7.1.a)
13
14 # ...
15
16 # VTK
17 VTKCONFIG_FILE := external/vtk_build/lib/cmake/vtk-7.1/VTKConfig.cmake
18 $(VTKCONFIG_FILE):
19     mkdir -p external/vtk_build
20     cd external/vtk_build && CC=$(CC) CXX=$(CXX) cmake ../vtk -DCMAKE_INSTALL_
21     ↪ PREFIX=`pwd` $(EXT_LTO_CMFLAGS) -DBUILD_SHARED_LIBS=OFF -DBUILD_TESTING=OFF -DCMAKE_
22     ↪ BUILD_TYPE=Release -DVTK_Group_MPI=OFF -DVTK_Group_Rendering=OFF -DVTK_Group_
23     ↪ StandAlone=OFF -DVTK_RENDERING_BACKEND=None -DVTK_USE_CXX11_FEATURES=ON -DModule_
24     ↪ vtkCommonDataModel=ON -DModule_vtkFiltersProgrammable=ON -DModule_
25     ↪ vtkIOParallelXML=ON -DModule_vtkParallelMPI=ON
26     $(MAKE) -j $(MJOBS) -C external/vtk_build

```

(continues on next page)

(continued from previous page)

```

14      $(MAKE) -C external/vtk_build install
15
16  # juelich_all
17  $(JUELICH_ALL_LIB): $(VTKCONFIG_FILE)
18      mkdir -p external/juelich_all_build
19      cd external/juelich_all_build && CC=$(CC) CXX=$(CXX) cmake ../juelich_all
20      ↪ $(EXT_LTO_CMFLAGS) -DCMAKE_Fortran_FLAGS="-Wall -Wextra -fbacktrace $(EXT_LTO)" -
21      ↪ DCMMAKE_INSTALL_PREFIX=`pwd` -DCM_ALL_FORTRAN=ON -DCM_ALL_USE_F08=$(ALL_USE_F08) -
22      ↪ DCMMAKE_BUILD_TYPE=Release -DCM_ALL_DEBUG=OFF -DCM_ALL_VTK_OUTPUT=ON -DVTK_DIR=`pwd` /
23      ↪ ../vtk_build/lib/cmake/vtk-7.1
24      $(MAKE) -C external/juelich_all_build
25      $(MAKE) -C external/juelich_all_build install

```

Software Technical Information**Name** A Load Balancing Library (ALL)/GMPM-PoC**Language** Fortran/C/C++**Licence** BSD 3-Clause**Documentation Tool** In source documentation using Doxygen, additional man pages and plain text**Application Documentation** [HemeLB website](#)**Relevant Training Material** [General ALL web-based seminar](#)**Software Module Developed by** Rene Halver**Cooperation with HemeLB**

- *Purpose of Module*
- *Background Information / Bibliography*
- *Cooperation content*
- *Cooperation results*

“HemeLB is a high performance lattice-Boltzmann solver optimized for simulating blood flow through sparse geometries, such as those found in the human vasculature.”¹ The code is used within the CompBioMed HPC Centre of Excellence H2020 project², and is already highly optimized for HPC usage. As a consequence of the initial workshop about the ALL library hosted by JSC in the context of E-CAM, a cooperation was set up in order analyse and test whether the use of ALL could improve the existing scalability of the code.

Purpose of Module

This module describes the cooperation between the ALL library and the HemeLB code, from the CompBioMed HPC Centre of Excellence. It provides details about the work performed and the results of the cooperation.

¹ <http://hemelb.org/#about>, 03.02.2021

² <https://www.compbiomed.eu/>

Background Information / Bibliography

More information about HemeLB can be found at¹, while more information about CompBioMed can be found at².

Cooperation content

As ALL was designed to work with particle codes, it was interesting to apply the library to an lattice-Boltzmann solver, which usually is not particle-based. Therefore the different grid points of the solution grid were designated as particles and since each of the grid-points already was assigned a workload, the sum of grid-point workloads could be used as domain work load. Since the creation and change of domain boundaries during the simulation was deemed not feasible, an [example code within ALL](#) was used to create initial, balanced domain decompositions for various systems and the results compared to the already existing load-balancing solution within HemeLB. Since test runs were performed on a large scale, e.g. SuperMUC, it was necessary to also provide MPI-I/O based output for better parallel I/O efficiency.

Cooperation results

As a result it can be stated that the domain compositions provided by ALL show a better theoretical load distribution. Tests to check if this translates into better code performance are inconclusive as yet, due to hardware related issues on the testing platforms. These are currently under further investigation, and more definitive results about the performance of the ALL-provided domain decompositions can be expected in the near future.

The results were part of a publication about HemeLB, which was published in 2020 [[Coveney](#)].

4.3.7 Ludwig: A lattice Boltzmann code for complex fluids

The modules listed here account for the modifications of the code Ludwig, carried out within the E-CAM project.

Software Technical Information

Name Ludwig: A lattice Boltzmann code for complex fluids

Language C

Licence <https://github.com/ludwig-cf/ludwig/blob/master/LICENSE>

Documentation Tool LaTeX-generated pdf

Application Documentation <https://github.com/ludwig-cf/ludwig/tree/master/docs/tutorial>

Externally imposed chemical potential gradient for binary fluid mixture

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source code*

We present a module that implements an externally imposed chemical potential gradient to the Lattice Boltzmann code *Ludwig*. The gradient is further used in simulation of binary fluid mixture and enables the studies of the related flows in various porous materials.

Purpose of Module

The gradient of chemical potential in a binary fluid mixture gives rise to a flow, whenever the value of the order parameter differs from 0. In *Ludwig*, we first implement the gradient as a vectorial physical property, which can, similarly to all other physical properties, be used anywhere in the code. Further on, we use it in the context of binary fluid mixture in the subroutines that account for the time evolution of the order parameter (Cahn-Hilliard equation) and the force, that arises due to the non-zero chemical potential gradient and order parameter.

This module is essential for studying binary fluid flows in porous materials as well as flows, arising due to the wetting effect of the walls in nanochannels.

Background Information

This module implements the externally imposed chemical potential gradient (for binary fluid mixture) in the *Ludwig* code. The latter, together with its documentation and tutorial is available on the following link: <https://github.com/ludwig-cf/ludwig>.

Building and Testing

The module is built and run in the same way as any other simulation in *Ludwig*. A detailed description of the latter is available at: <https://github.com/ludwig-cf/ludwig/tree/master/docs/tutorial>.

Specifically, an example of the input file for a binary fluid simulation with externally imposed chemical potential gradient is available at: <https://github.com/ludwig-cf/ludwig/blob/develop/tests/regression/d3q19-short/serial-muex-st1.inp>. The externally imposed chemical potential gradient is specified in the input file, by the following lines:

```
fd_force_divergence 0
grad_mu 0.00001_0.00002_0.00003
```

Source code

The related pull requests in *Ludwig*'s github repository can be found at <https://github.com/ludwig-cf/ludwig/pull/80> and <https://github.com/ludwig-cf/ludwig/pull/88>.

The changes have been incorporated into the *Ludwig*'s new versions, for the first time within the release/version 0.11.0 (see <https://github.com/ludwig-cf/ludwig/blob/master/CHANGES.md>).

Software Technical Information

Name Ludwig: A lattice Boltzmann code for complex fluids

Language C

Licence <https://github.com/ludwig-cf/ludwig/blob/master/LICENSE>

Documentation Tool LaTeX-generated pdf

Application Documentation <https://github.com/ludwig-cf/ludwig/tree/master/docs/tutorial>

Implementation of simple cubic, body-centered cubic, and face-centered cubic crystalline capillaries

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source code*

We present a module that implements simple cubic (SCC), body-centered cubic (BCC), and face-centered cubic (FCC) crystalline geometries as a utility to create capillaries in the Lattice Boltzmann code Ludwig. The crystalline geometries created are used as porous materials in Lattice Boltzmann simulations.

Purpose of Module

The crystalline structures represent various types of porous materials. We can attribute to them wetting properties, via which the solid parts of the geometries interact with the fluid. This module is essential for the studies of various types of flows through such materials. In particular, we focus on flows of binary fluid mixtures, driven by an externally imposed chemical potential gradient, through porous media.

Background Information

This module implements the SCC, BCC, and FCC crystalline capillaries in the Ludwig code. The latter, together with its documentation and tutorial is available on the following link: <https://github.com/ludwig-cf/ludwig>.

Building and Testing

The module is run by specifying the name of the output file, the output type, the crystalline type, the size of the crystalline cell, and the size of the whole system in the file `capillary.c` (<https://github.com/ludwig-cf/ludwig/blob/master/util/capillary.c>). The file is then compiled and run by:

```
make capillary
./capillary
```

This generates the output file (e.g. “capillary.001-001”). The thus generated capillary file, together with the desired output data, is then specified in the input file of the simulation. An example of this specification is:

```
porous_media_format BINARY
porous_media_file   capillary
porous_media_type   status_with_c_h
```

The latter line `porous_media_type` should match the output type, defined in the `capillary.c` file, prior to its compilation.

Source code

The module has been provided as a pull request on Ludwig’s github repository <https://github.com/ludwig-cf/ludwig/pull/98>.

What is a module?

In the context of **E-CAM**, the definition of a software module is any piece of software that could be of use to the **E-CAM** community and that encapsulates some additional functionality, enhanced performance or improved usability for people performing computational simulations in the domain areas of interest to us.

This definition is deliberately broader than the traditional concept of a module as defined in the semantics of most high-level programming languages and is intended to capture inter alia workflow scripts, analysis tools and test suites as well as traditional subroutines and functions. Because such **E-CAM** modules will form a heterogeneous collection we prefer to refer to this as an **E-CAM** software repository rather than a library (since the word library carries a particular meaning in the programming world). The modules do however share with the traditional computer science definition the concept of hiding the internal workings of a module behind simple and well-defined interfaces. It is probable that in many cases the modules will result from the abstraction and refactoring of useful ideas from existing codes rather than being written entirely de novo.

Perhaps more important than exactly what a module is, is how it is written and used. A final **E-CAM** module adheres to current best-practice programming style conventions, is well documented and comes with either regression or unit tests (and any necessary associated data). **E-CAM** modules should be written in such a way that they can potentially take advantage of anticipated hardware developments in the near future (and this is one of the training objectives of **E-CAM**).

5.1 Scientific Software Development Best Practices

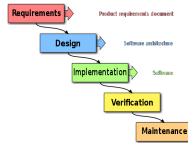
We have attempted to gather a set of best practice guidelines for scientific software development in order to assist people to develop high quality modules. These guidelines are not specific to **E-CAM** and gather together best practices from a number of different sources that can help increase the quality and reusability of the software developed by scientists.

General Information

- *Scientific Software Best Practices*
 - *General Programming Guidelines*
 - *Programming for an HPC Environment*

- *How to contribute?*
- search

5.1.1 Scientific Software Best Practices



This text is not meant to be a set of rules, but a set of guidelines that have been formed on the basis of hard-earned experience. We welcome contributions from anyone, anywhere on any of the topics discussed here. The home repository for this documentation is within the [E-CAM Software Library](#).

It began as a starting point for guidelines for contributions to the Extended Software Development Workshops (ES-DWs) of E-CAM but can be considered as a collection of best practice advice for scientific software development. The scope of these workshops was always expected to vary significantly depending on the research area, as well as over the project lifetime. For this reason the document itself is quite broad and (in many places) introductory in nature.

General Programming Guidelines

Firstly let us consider some guidelines that are applicable regardless of the type of software project you are working on.

General Programming Guidelines

Language-independent best practices

The Unix Philosophy

When we develop software in a community we must consider that our work is not just for ourselves but is expected to be used, adapted and (even) improved by others in that community. Much of the Unix philosophy [[Raymond](#)] is worth considering when developing in such a context:

- Rule of Modularity: Write simple parts connected by clean interfaces.
- Rule of Clarity: Clarity is better than cleverness.
- Rule of Composition: Design programs to be connected to other programs.
- Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
- Rule of Simplicity: Design for simplicity; add complexity only where you must.
- Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.
- Rule of Transparency: Design for visibility to make inspection and debugging easier.
- Rule of Robustness: Robustness is the child of transparency and simplicity.
- Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.
- Rule of Least Surprise: In interface design, always do the least surprising thing.
- Rule of Silence: When a program has nothing surprising to say, it should say nothing.

- Rule of Repair: When you must fail, fail noisily and as soon as possible.
- Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.
- Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.
- Rule of Optimization: Prototype before polishing. Get it working before you optimize it.
- Rule of Diversity: Distrust all claims for “one true way”.
- Rule of Extensibility: Design for the future, because it will be here sooner than you think.

If you’re new to Unix, these principles are worth some meditation (and I would recommend reading the fuller descriptions of the [programming “rules” derived from the Unix philosophy](#))

When we develop software in a community we must consider that our work is not just for ourselves but is expected to be used, adapted and (even) improved by others in that community. Much of *The Unix Philosophy* is worth considering when developing in such a context.

The Unix Philosophy principles apply to the creation of the software, but there are also some universally recommended best practices when it comes to the software development work-flow itself [[BestPractices](#)]:

- Use a version management tool,
- Make a build in one step,
- Test suites to make sure what you are working on does what you think it should,
- Test-first programming: writing the test for a new line of code before writing that new line of code.

For our specific use case we will support [Git](#) as our version management tool (with repositories hosted on the [E-CAM GitLab service](#)), [CMake](#) and [Autotools](#) (complemented by [EasyBuild](#) for HPC environments) as our supported build environments and unit/regression testing and continuous integration through the [E-CAM GitLab service](#).

Development Guidelines

Going a little deeper into the specifics, let’s consider some advice with respect to a desirable software development workflow.

Designing your software

After some discussion with other [Centres of Excellence](#), we are wary of over-engineering advice with respect to software design. While tools exist that are designed to assist this process, such tools typically require repeated use to achieve mastery and this overhead excessive in many cases. For this reason, we have chosen to align our software specification method with our *acceptance criteria* for software contributions to a project (source code, testing, documentation, build instructions).

The subsections below are based on the information from [Wikipedia](#) ([[SoftwareSpecification](#)] and [[TDD](#)]).

Software Requirements Specification

A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

The software requirements specification document lists sufficient and necessary requirements that are required for developing the project. To derive the requirements, all developers need to have clear and thorough understanding of

the application to be developed. This is achieved through continuous communications between the project team (who are typically also the *customer* when we are talking about an open source scientific code).

An example of an SRS would be:

- Purpose
 - Definitions
 - System overview
 - References
- Overall description
 - Product perspective
 - * System Interfaces
 - * User interfaces
 - * Hardware interfaces
 - * Software interfaces
 - * Communication Interfaces
 - * Memory Constraints
 - Design constraints
 - * Operations
 - * Site Adaptation Requirements
 - Product functions
 - User characteristics
 - Constraints, assumptions and dependencies
- Specific requirements
 - External interface requirements
 - Functional requirements
 - Performance requirements
 - Software System attributes
 - * Reliability
 - * Availability
 - * Security
 - * Maintainability
 - * Portability.

Test-driven Development

The acceptance criteria of E-CAM are explicitly test-focused and we therefore advocate for test-driven development as a software specification method, where one first decides how a particular development would be tested (and creates the associated test) before writing the software that would pass this test.

An example of a test-driven development cycle would be:

1. Add a test
2. Run all tests and see if the new test fails
3. Write the code
4. Run tests
5. Refactor code

Such an approach is very task-oriented and if a wider perspective is required (for example if one is beginning a software project or implementing a redesign) we advise the creation of a *Software Requirements Specification* to supplement this and provide an overarching structure.

In addition, when dealing with numerical methods the creation of adequate tests can be difficult since bit-wise reproducibility of results is frequently not possible due to floating point precision and/or the use of random numbers.

Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Many people’s version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they’re clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you’re in and accidentally write to the wrong file or copy over files you don’t mean to.

[Git](#) is a widely used source code management system for software development. As with most other distributed version control systems, and unlike most client–server systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server.

We will use [Git](#) together with the [E-CAM GitLab service](#) within E-CAM. [Git](#) will be our version control utility and the [E-CAM GitLab service](#) will help us manage our work-flow by allowing us to create/label/follow/assign issues, review integration of new features, create milestones, etc (but you could also use [GitHub](#) to get the same basic services).

There are many excellent guides to [Git](#), [GitLab](#) and [GitHub](#) so we will not go into any great detail in this document. For detailed information we refer you to [Software Carpentry’s git lessons](#), the [GitLab documentation](#) and the [GitHub user guides](#).

Branching Strategy

We follow the [GitHub Flow](#) branching strategy which is well described at that link.

Versioning of Releases

Software versioning is the process of assigning either unique version names or unique version numbers to unique states of computer software. Adopting a logical system for releasing versions provides information to users that allows them, for example, to predict whether it is *safe* for them to move to a new release.

We follow the “[Semantic Versioning 2.0.0](#)” strategy. Given a version number $x.y.z$ (MAJOR.MINOR.PATCH), increment the:

- MAJOR version x when you make incompatible API changes,
- MINOR version y when you add functionality in a backwards-compatible manner, and
- PATCH version z when you make backwards-compatible bug fixes.

The approach relies on bumping the correct component up at the right time. Therefore, determining which type of version you should be releasing is simple. If you are mostly fixing bugs, then this would be categorized as a patch, in which case you should bump `z`. If you are implementing new features in a backward-compatible way, then you will bump `y` because this is what's called a minor version. On the other hand, if you implement new stuff that is likely to break the existing API, you need to bump `x` because it is a major version.

Additional labels for pre-release and build meta-data are available as extensions to the `MAJOR.MINOR.PATCH` format. We begin at `MAJOR` version 0 and API changes may be allowed without incrementing it until we are ready to release a first stable version. Once we release `MAJOR` version 1 we intend to strictly follow the API policy.

Creating [releases on GitLab](#) is well described in their documentation pages.

Creating Citable Code

[Zenodo](#) is a great resource that allows you to get a DOI for your code repository.

Zenodo has direct GitHub integration (and there is a [handy guide for how to use GitHub's Zenodo integration](#)) but the process can also be done manually. You can upload a zip-ball of your software to Zenodo, provide some metadata and publish it to get a DOI (just as you would if you uploaded a paper or data). It's probably best if you make a zip-ball of a tagged release, so that the DOI captures something *complete*. Additionally you can supplement your record metadata on Zenodo with a "related identifier" (e.g. a URL) and point back to the tag on your live repository. This way anyone who discovers your software in the future will also have means to follow your live repository and find the most recent version of the software.

Coding Guidelines and Code Review

While this document can be used as a starting point for coding best practices, it is really intended for those who want to contribute code to the E-CAM project, and describes the starting point for our coding standards and code review checklist. We try to adopt best practices from existing projects with plenty of relative experience. [\[PyCogent\]](#) (for example) has useful coding guidelines that will act as a starting point for us.

Code, scripts, and documentation should have their spelling checked. All plain-text files should have line widths of 120 characters or less unless that is not supported for the particular file format. It is typical in many projects that a limit of 80 characters is given but we consider this excessively restrictive.

Variable naming

- Choose the name that people will most likely guess. Make it descriptive, but not too long: `curr_record` is better than `c`, or `curr`, or `current_genbank_record_from_database`.
- Good names are hard to find. Don't be afraid to change names except when they are part of interfaces that other people are also using. It may take some time working with the code to come up with reasonable names for everything: if you have unit tests, it's easy to change them, especially with global search and replace.
- Use singular names for individual things, plural names for collections. For example, you'd expect `self.Name` to hold something like a single string, but `self.Names` to hold something that you could loop through like a list or dict. Sometimes the decision can be tricky: is `self.Index` an int holding a position, or a dict holding records keyed by name for easy lookup? If you find yourself wondering these things, the name should probably be changed to avoid the problem: try `self.Position` or `self.LookUp`.
- Don't make the type part of the name. You might want to change the implementation later. Use `Records` rather than `RecordDict` or `RecordList`, etc. Don't use Hungarian Notation either (i.e. where you prefix the name with the type).

- Make the name as precise as possible. If the variable is the name of the input file, call it `infile_name`, not `input` or `file` (which you shouldn't use anyway, since they're keywords), and not `infile` (because that looks like it should be a file object, not just its name).
- Use `result` to store the value that will be returned from a method or function. Use `data` for input in cases where the function or method acts on arbitrary data (e.g. sequence data, or a list of numbers, etc.) unless a more descriptive name is appropriate.
- One-letter variable names should only occur in math functions or as loop iterators with limited scope. Limited scope covers things like `for k in keys: print k`, where `k` survives only a line or two. Loop iterators should refer to the variable that they're looping through: `for k in keys, i in items`, or `for key in keys, item in items`. If the loop is long or there are several 1-letter variables active in the same scope, rename them.
- Limit your use of abbreviations. A few well-known abbreviations are OK, but you don't want to come back to your code in 6 months and have to figure out what `sptxck2` is. It's worth it to spend the extra time typing `species_taxon_check_2`, but that's still a horrible name: what's check number 1? Far better to go with something like `taxon_is_species_rank` that needs no explanation, especially if the variable is only used once or twice.

Naming Conventions

It is important to follow the naming conventions because they make it much easier to guess what a name refers to. In particular, it should be easy to guess what scope a name is defined in, what it refers to, whether it's OK to change its value, and whether its referent is callable. The following rules provide these distinctions.

- `lowercase_with_underscore` for modules and internal variables (including function/method parameters).
- `MixedCase` for classes and public properties, and for factory functions that act like additional constructors for a class.
- `mixedCaseExceptFirstWord` for public methods and functions.
- `_lowercase_with_leading_underscore` for private functions, methods, and properties.
- `__lowercase_with_two_leading_underscores` for private properties and functions that must not be overwritten by a subclass.
- `CAPS_WITH_UNDERSCORES` for named constants.

Underscores can be left out if the words read OK run together. `infile` and `outfile` rather than `in_file` and `out_file`; `infile_name` and `outfile_name` rather than `in_file_name` and `out_file_name` or `infilename` and `outfilename` (getting too long to read effortlessly).

Merge Requests

We want people who contribute back to use a “branch-hack-pull request” cycle, the [GitHub Flow](#). Our website contains greater detail on the exact steps required (at [How to contribute?](#)) but the basic concept is:

- Create your own copy of the repository on the [E-CAM GitLab service](#) (fork)
- clone your repository to your machine
- Create a new branch for your feature. Feature branches should have descriptive names, like `animated-menu-items` or `issue-#1061`.
- Hack

- push your changes back to GitLab
- Create a [Merge Request](#)¹ against the appropriate of the E-CAM library. This gives other developers an opportunity to review the changes before they become a part of the main codebase.

Code review (see below) is a major benefit of merge requests, but merge requests are actually designed to be a generic way to talk about code. You can think of merge requests as a discussion dedicated to a particular branch. This means that they can also be used much earlier in the development process. For example, if a developer needs help with a particular feature, all they have to do is file a merge request. Interested parties will be notified automatically, and they'll be able to see the question right next to the relevant commits.

Code Review and Checklist

Contributors also make good reviewers so we'd like you to be aware of what the review process looks like. We try to follow the best practices as described in "[11 Best Practices for Peer Code Review](#)". The most important to mention are:

- Merge requests should be small, the target is to review fewer than 400 lines of code at a time.
- Authors should document source code before the review.
- Embrace the subconscious implications of peer review. The knowledge that others will be examining their work naturally drives people to produce a better product.

Copy and paste the following into a merge request comment when it is ready for review (in our case that will be on the [E-CAM GitLab service](#)). This lists helps ensure that we try to reach many of our targets in terms of:

```
- [ ] Is it mergeable? (i.e., there should be no merge conflicts)
- [ ] Did it pass the tests? (Are there unit/regression tests? Do they pass?)
- [ ] If it introduces new functionality, is it tested? (Unit tests?)
- [ ] Is it well formatted? (typos, line length, brackets,...)
- [ ] Did it change any interfaces? Only additions are allowed without a major version
      increment. Changing file formats also requires a major version number increment.
- [ ] Is the Copyright year up to date?
```

Note: After you submit the comment you can check and uncheck the individual boxes on the formatted comment in GitLab; no need to put x or y in the middle.

Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

Our weapon of choice is [GitLab CI](#), the main reason being controlling the servers where the tests are run allows us to customise their configuration. Of course you can get similar services on [GitHub](#), [Travis CI](#) being the most popular (and completely for free if you are open source).

Gamification

One of the big advantages of the automated CI is that it helps to gamify the development experience. Having all your unit tests pass or having 100% code coverage for your tests (and getting a little badge to appear notifying you), gives

¹ Merge requests let you tell others about changes you've pushed to a GitLab repository. Once a merge request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary

a (small) feeling of accomplishment and can be a good motivator to write better merge requests.

Integrated Development Environment

An Integrated Development Environment (IDE) is not a *necessary* development tool but it is a useful one. While we understand that people are familiar with a particular editor or work-flow, use of an IDE can help the development process flow more easily. An IDE is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. We are making this recommendation purely because IDEs can save you lot of **time**, particularly when you are contributing to someone else's code. There are many good reasons to use one:

- Integrated source control (this is major since the Git UI is frequently not intuitive)
- Quickly navigating to a type without needing to worry about namespace, project etc.
- Navigating to members by treating them as hyperlinks
- Auto-completion when you can't remember the names of all members by heart
- Automatic code generation
- Refactoring (major advantage)
- Warning-as-you-type (i.e. some errors don't even require a compile cycle)
- Hovering over something to see the documentation (provided by Doxygen)
- Keeping a view of files, errors/warnings/console/unit tests etc. and source code all on the screen at the same time in a useful way
- Ease of running unit tests from the same window
- Integrated debugging
- Navigating to where a compile-time error or run-time exception occurred directly from the error details.

If you are developing for a parallel environment for multiple HPC systems the [Eclipse IDE](#) even has a plugin specifically designed for this, the [Eclipse Parallel Tools Platform](#).

Programming for an HPC Environment

Due to the nature of the HPC environment (novel hardware, latest techniques, remote resources,...). There are many specific things that need to be considered that impact the software development process.

HPC Programming Guidelines

Once we begin to discuss high performance computing (HPC), we necessarily must begin to discuss not only the latest hardware technologies, but also the latest software technologies that make exploiting the capabilities of that hardware easier.

Hardware Developments

There are a number of different organisations and projects that are generating roadmaps about the current and future technologies that are (or may be in the future) available in the HPC space. [Eurolab-4-HPC](#) has summarised many of these in the [Eurolab-4-HPC Long-Term Vision on High-Performance Computing](#). Here we focus on a small subset of the content of these roadmaps (primarily from [Eurolab-4-HPC](#) and [ETP4HPC](#)) that are most likely to impact the target community of E-CAM in the 3-5 year horizon.

Currently Available Hardware

For the last decade, power and thermal management has been of high importance. The entire market focus has moved from achieving better performance through single-thread optimizations, e.g., speculative execution, towards simpler architectures that achieve better performance per watt, provided that vast parallelism exists. The HPC community, particularly at the higher end, focuses on the flops/watt metric since the running cost of high-end HPC systems are so significant. It is the potential power requirements of exa-scale systems that are the limiting factor (given currently available technologies).

The practical outcome of this is the rise of accelerating co-processors and many-core systems. In the following sections we will discuss three such technologies that are likely to form the major computing components of the first generation of exa-scale machines:

Intel Many-core

The 2nd Generation Intel Xeon Phi platform, known as Knights Landing (KNL), has been released on the market in Q2 of 2016. The chip, based on a 14nm lithography, contains up to 72 cores @1.5GHz with a maximum memory bandwidth of 115.2 GB/s. One of the main features is the increased AVX512 ISE (Instruction Set Extensions) which includes SSE (Streaming SIMD Extensions) and AVX (Advanced Vector Extensions). More details are available at [Intel Knights Landing](#). The same component (Intel Xeon Phi 7250-F) is available in the JURECA Booster as part of the DEEP and DEEP-ER projects.

Knights Hill is the codename for the third-generation MIC architecture and it will be manufactured in a 10 nm process. Intel announced the first details at SC14, however since then no further details have been released and the Aurora project from the DoE delayed (see [Some Surprises in the 2018 DoE Budget for Supercomputing](#)).

[Knights Mill](#) is Intel's codename for a Xeon Phi product specialised in deep learning. It is expected to support reduced variable precision which have been used to accelerate machine learning in other products, such as half-precision floating-point variables in Nvidia's Tesla.

Feedback for software developers

Based on the latest hardware developments specified above (and the AVX512 instruction set used by this hardware), we strongly advise the software developer to take in consideration the importance of enhancing performance through vectorization both from numerical algorithm point of view and at the compiler level. Intel provides very good tools to achieve this through compiler flags (which allow you to have a full report about the vectorization efficiency) or more sophisticated software like [Intel Advisor](#).

At node-level the recommended parallelism is by shared memory. In this case [OpenMP](#) is the de facto standard and Intel provides good tools like [VTune](#).

Many training courses and documents are available on line (see [Intel Advisor training](#) and [VTune training](#)).

NVIDIA GPU

The new NVIDIA [Tesla V100](#) accelerator incorporates the new Volta GV100 GPU. Equipped with 21 billion transistors, Volta delivers over 7.5 Teraflops per second of double precision performance, 1.5x increase compared to the its predecessor, the Pascal GP100 GPU. Moreover, architectural improvements include:

- A tensor core is unit that multiplies two 4x4 FP16 matrices, and then adds a third FP16 or FP32 matrix to the result by using fused multiply-add operations, and obtains an FP32 result that could be optionally demoted to an FP16 result. Tensor cores are intended to speed up the training of neural networks.

- Tesla V100 uses a faster and more efficient HBM2 implementation. HBM2 memory is composed of memory stacks located on the same physical package as the GPU, providing substantial power and area savings compared to traditional GDDR5 memory designs, thus permitting more GPUs to be installed in servers. In addition to the higher peak DRAM bandwidth on Tesla V100 compared to Tesla P100, the HBM2 efficiency on V100 GPUs has been significantly improved as well. The combination of both a new generation HBM2 memory from Samsung, and a new generation memory controller in Volta, provides 1.5x delivered memory bandwidth versus Pascal GP100, and greater than 95% memory bandwidth efficiency running many workloads.
- NVlink 2.0, which is a high-bandwidth bus between multiple GPUs, and between the CPU and GPU. Compared to NVLink on Pascal, NVLink 2.0 on V100 increases the signaling rate from 20 to 25 Gigabits/second. Each link now provides 25 Gigabytes/second in each direction. The number of links supported has been increased from four to six pushing the supported GPU NVLink bandwidth to 300 Gigabytes/second. The links can be used exclusively for GPU-to-GPU communication as in the DGX-1 with V100 topology shown in Figure 2, or some combination of GPU-to-GPU and GPU-to-CPU communication as shown in Figure 3 (currently only available in combination with Power8/9 processors).

The tensor core of the Volta was explicitly added for deep learning workloads. The [NVIDIA Deep Learning SDK](#) provides powerful tools and libraries for designing and deploying GPU-accelerated deep learning applications. It includes libraries for deep learning primitives, inference, video analytics, linear algebra, sparse matrices, and multi-GPU communications.

Feedback for software developers

Several approaches have been developed to exploit the full power of GPUs: from parallel computing platform and application programming interface specific for NVidia GPU, like [CUDA 9.0](#), to the latest version of [OpenMP 4.5](#) which contains directives to offload computational work from the CPU to the GPU. While CUDA currently is likely to achieve best performance from the device, OpenMP allows for better portability of the code across different architectures. Finally, the [OpenACC](#) open standard is an intermediate between the two, more similar to OpenMP than CUDA, but allowing better usage of the GPU. Developers are strongly advised to look into these language paradigms.

Moreover, it is fundamental to consider that there are several issues linked to hybrid architectures, like CPU-GPU and GPU-GPU bandwidth communication (the latest greatly improved through NVlink), direct access through [Unified Virtual Addressing](#), the presence of new APIs for programming (such as [Tensor Core](#) multiplications specifically designed for deep learning algorithms).

Finally, it is important to stress the improvements made by NVidia on the implementation of [Unified Memory](#). This allows the system to automatically migrate data allocated in Unified Memory between host and device so that it looks like CPU memory to code running on the CPU, and like GPU memory to code running on the GPU making programmability greatly simplified.

At this stage, GPU programming is quite mainstream and there are many training courses available online, see for example the [NVidia education site](#) for material related to CUDA and OpenACC. Material for OpenMP is more limited, but as an increasing number of compilers begin to support the OpenMP 4.5 standard, we expect the amount of such material to grow (see [this presentation on performance of the Clang OpenMP 4.5 implementation on NVIDIA gpus](#) for a status report as of 2016).

FPGA

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks.

Xilinx Ultrascale FPGAs and ARM processors have been proposed by the EuroEXA project as a new path towards exascale. EuroEXA is an EU Funded 20 Million Euro for an [ARM+FPGA Exascale Project](#) which will lead Europe

towards exascale, together with [ExaNeSt](#), [EcoScale](#) and [ExaNoDe](#) projects, scaling peak performance to 400 PFLOP in a peak system power envelope of 30MW; over four times the performance at four times the energy efficiency of today's HPC platforms.

Feedback for software developers

Despite their high efficiency in performance and power consumption, FPGA are known for being difficult to program. [OpenCL for FPGA](#) is an example of programming language for FPGA which we recommend, particularly considering that these new technologies will be soon available within the E-CAM community through the EuroEXA project.

We will outline the current generation of technologies in this space and also describe the (currently) most-productive programming model for each. We will not discuss other new CPU technologies (such as Power 9, Intel Skylake, or ARMv8) since in comparison to these technologies they would be expected to only provide ~10% or less of the compute power of potential exa-scale systems.

The problem with the current three-pronged advance is that it is not always easy to develop parallel programs for these technologies and, moreover, those parallel programs are not always performance portable between each technology, meaning that each time the architecture changes the code may have to be rewritten. While there are open standards available for each technology, each product currently has different preferred standards which are championed by the individual vendors (and therefore the best performing).

In general, we see a clear trend towards more complex systems, which is expected to continue over the next decade. These developments will significantly increase software complexity, demanding more and more intelligence across the programming environment, including compiler, run-time and tool intelligence driven by appropriate programming models. Manual optimization of the data layout, placement, and caching will become uneconomic and time consuming, and will, in any case, most likely soon exceed the abilities of the best human programmers.

Impact of Deep Learning

Traditional machine learning uses handwritten feature extraction and modality-specific machine learning algorithms to label images or recognize voices. However, this method has several drawbacks in both time-to-solution and accuracy. Today's advanced deep neural networks use algorithms, big data, and the computational power of the GPU (and other technologies) to change this dynamic. Machines are now able to learn at a speed, accuracy, and scale that are driving true artificial intelligence and AI Computing.

Deep learning is used in the research community and in industry to help solve many big data problems such as computer vision, speech recognition, and natural language processing. Practical examples include:

- Vehicle, pedestrian and landmark identification for driver assistance
- Image recognition
- Speech recognition and translation
- Natural language processing
- Life sciences

The influence of deep-learning on the market is significant with the design of commodity products such as the Intel MIC and NVIDIA Tesla being heavily impacted. Silicon is being dedicated to deep learning workloads and the scientific workloads for these products will need to adapt to leverage this silicon.

Future HPC Hardware in Europe

The European HPC Technology Platform, ETP4HPC, is an industry-led think-tank comprising of European HPC technology stakeholders: technology vendors, research centres and end-users. The main objective of ETP4HPC is

to define research priorities and action plans in the area of HPC technology provision (i.e. the provision of supercomputing systems). It has been responsible for the production and maintenance of the [European HPC Technology Strategic Research Agenda \(SRA\)](#), a document that serves as a mechanism to provide contextual guidance to European researchers and businesses as well as to guide EU priorities for research in the Horizon 2020 HPC programme, i.e. it represents a roadmap for the achievement of European exascale capabilities.

We have had numerous discussions of the E-CAM community software needs through our exchanges with ETP4HPC during the course of our contributions to the SRA. The particular contribution from our discussion related to the software needs for exascale computing within the ETP4HPC SRA report is shown in the paragraphs below:

E-CAM has not committed itself to a single set of applications or use cases that can be represented in such a manner, it is instead driven by the needs of the industrial pilot projects within the project (as well as the wider community). Taking into consideration the CECAM community and the industrial collaborations targeted by E-CAM, probably the largest exa-scale challenge is ensuring that the skillsets of the application developers from academia and industry are sufficiently up to date and are aligned with programming best practices. This means that they are at least competent in the latest relevant language specification (Fortran 2015, C++17, ...) and aware of additional tools and libraries that are necessary (or useful) for application development at the exa-scale. For application users, this means that they have sufficient knowledge of architecture, software installation and typical supercomputing environment to build, test and run application software optimised for the target.

While quite specific “key applications” are under continuous support by other CoEs, this is not the current model of E-CAM. E-CAM is more likely to support and develop a software installation framework (such as [EasyBuild](#)) that simplifies building the (increasingly non-trivial) software stack of a particular application in a reliable, reproducible and portable way. Industry has already shown significant interest in this and E-CAM is particularly interested in extending the capabilities of EasyBuild to EsD architectures, performance analysis workflows and to new scientific software packages. Such an effort could easily be viewed as transversal since such developments could be leveraged by any other CoE.

One important focus of the SRA is the development of the “Extreme-Scale Demonstrators” (EsDs) that are vehicles to optimise and synergise the effectiveness of the entire HPC H2020 programme, through the integration of R&D outcomes into fully integrated HPC system prototypes.

The work in developing the EsDs will fill critical gaps in the H2020 programme, including the following activities:

- Bring technologies from FET-HPC projects closer to commercialisation.
- Combined results from targeted R&D efforts into a complete system (European HPC technology ecosystem).
- Provide the missing link between the three HPC pillars: technology providers, user communities (e.g. E-CAM) and infrastructure.

As one of the CoEs, E-CAM should aim to provide insight and input into the requirements of future exascale systems based on lessons learnt from activities within E-CAM (e.g. software development and relevant performance optimisation and scaling work). This would entail further knowledge and understanding within E-CAM on exploiting current multi-petaflop infrastructures, what future exascale architectures may look like, as well as interaction and close collaboration between E-CAM and other projects (i.e. the projects shown in Figure 12); these are also covered in subsequent sections of this paper.

Emerging hardware architectures relevant to exascale computing

The European Commission supports a number of projects in developing and testing innovative architectures for next generation supercomputers, aimed at tackling some of the biggest challenges in achieving exascale computing. They often involve co-design involving HPC technologists, hardware vendors and code developer/end-user communities in order to develop prototype systems. Some of these projects include:

- The [DEEP](#) (Dynamic Exascale Entry Platform) projects (DEEP, DEEP-ER and DEEP-EST)

- The [Mont-Blanc](#) projects (Mont-Blanc 1, 2 and 3)
- The [PRACE PCP](#) (Pre-Commercial Procurement) initiative

Software Developments

It is clear that the hardware developments described above will greatly impact the software development practices of the E-CAM development community. For this reason, we highlight the the language standards, runtime environments, workflows and software tools that can help E-CAM developers to deliver high quality, resilient software for current and next generation machines.

Programming for HPC

C++17

C++17 is the name for the most recent revision of the [ISO/IEC 14882](#) standard for the C++ programming language.

The previous C++ versions show very limited parallel processing capabilities when using multi/many core architectures. This situation will change with the C++17, in which the parallelised version of [Standard Template Library](#) is included. The STL is a software library for C++ programming which has 4 components: Algorithms, Containers, Functors and Iterators. “[Parallel STL advances the evolution of C++, adding vectorization and parallelization capabilities without resorting to nonstandard or proprietary extensions, and leading to code modernization and the development of new applications on modern architectures.](#)”

A multi-threading programming model for C++ is supported since C++11.

Fortran 2015

[Fortran 2015](#) is a minor revision of Fortran 2008 (which was when Fortran became a Partitioned Global Address Space (PGAS) language with the introduction of [coarrays](#)). The revisions mostly target additional parallelisation features and increased interoperability with C.

Most Fortran-based software E-CAM sees in practice is implemented in Fortran 95 and there appears to be little awareness of the parallel features of the latest Fortran standards. E-CAM is considering organising a workshop that addresses this lack of awareness (similar to the “[Software Engineering and Parallel Programming in Modern Fortran](#)” held at the Cranfield University).

It should be noted that [compiler support for the latest Fortran standards is limited](#). This is most likely due to the fact that Fortran is not widely used outside of the scientific research (limiting its commercial scope).

The (potential) role of Python

Given that it is an interpreted language (i.e., it is only compiled at runtime), Python is not usually discussed much in the HPC space since there is limited scope for control over many factors that influence performance. Where we are observing a lot of growth is where applications are being written in languages like C++ under the hood but are intended to be primarily used via their Python interfaces.

This is a valuable, and user friendly, development model that allows users to leverage Python for fast prototyping while maintaining the potential for high performance application codes.

A warning to would be users: [Python 2 will stop being developed in 2020](#) so please make sure that your code is Python3 compliant.

Open Standards

We describe here some of the open standards that are most likely to be leveraged on next generation HPC resources.

MPI

Now more than 25 years old, Message Passing Interface (MPI) is still with us and remains the de facto standard for internode communication (though it is not the only option, alternatives such as [GASNet](#) exist). [MPI-3.1](#) was approved by the MPI Forum on June 4, 2015. It was mainly an errata release for MPI 3.0 which included some important enhancements to MPI:

- Nonblocking collectives
- Sparse and scalable irregular collectives
- Enhancements to one-sided communication (very important for extreme scalability)
- Shared memory extensions (on clusters of SMP nodes)
- Fortran interface

Maintaining awareness of the [scope of past and future updates to the MPI standard](#) is important since it is the latest features that target the latest architectural developments.

OpenMP

OpenMP is also 20 years old and remains the most portable option for on-node workloads. The standard has introduced new features to deal with increasing node-level heterogeneity (device offloading, such as for the GPU, in particular) and varied workloads (task level parallelism).

From GCC 6.1, OpenMP 4.5 is fully supported for C and C++ (with Fortran support coming in the GCC 7 series). The [level of OpenMP support among other compilers](#) varies significantly.

OpenACC

[OpenACC](#) (for open accelerators) is a programming standard for parallel computing developed by Cray, CAPS, Nvidia and PGI. The standard is designed to simplify parallel programming of heterogeneous CPU/GPU systems. Since the paradigm is very similar to the latest OpenMP specs, a future merger into OpenMP is not unlikely. It should be noted that CUDA (with the [nvcc compiler](#)) is still the most commonly used (and highest performing) library for programming NVIDIA GPUs.

OpenCL

Open Computing Language (OpenCL) is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs, see Section 2.4 for the extreme relevance of this) and other processors or hardware accelerators.

OpenCL 2.2 brings the OpenCL C++ kernel language into the core specification for significantly enhanced parallel programming productivity. When releasing OpenCL version 2.2, the Khronos Group announced that OpenCL would be merging into [Vulkan](#) (which targets high-performance realtime 3D graphics applications) in the future, leaving some uncertainty as to how this may affect the HPC space.

Runtime System Approaches

As noted already, programming paradigm standards are moving forward to adapt to the technologies that we see in the market place. The complexity of the hardware infrastructure necessarily brings complexity to the implementation of the programming standards.

There are number of programming models that leverage runtime systems under development. They promise to abstract away hardware during the development process, with the proviso that tuning at runtime may be required. Our experience to date with these systems is limited so we simply provide a list of three such systems here (which is certainly not exhaustive) in no particular order:

- **HPX**, a C++ Standard Library for concurrency and parallelism. The goal of the HPX project is to create a high quality, freely available, open source implementation of **ParalleX** concepts for conventional and future systems by building a modular and standards conforming runtime system for SMP and distributed application environments. (Most recent release: v1.0, April 2017)
- **Kokkos** implements a programming model in C++ for writing performance portable applications targeting all major HPC platforms. For that purpose it provides abstractions for both parallel execution of code and data management. Kokkos is designed to target complex node architectures with N-level memory hierarchies and multiple types of execution resources. It currently can use OpenMP, Pthreads and CUDA as backend programming models. (Most recent release: v2.04.04, 11 Sept 2017)
- **OmpSs** is an effort to integrate features from the StarSs programming model developed at Barcelona Supercomputing Centre (BSC) into a single programming model. In particular, the objective is to extend OpenMP with new directives to support asynchronous parallelism and heterogeneity (devices like GPUs). However, it can also be understood as new directives extending other accelerator based APIs like CUDA or OpenCL. The OmpSs environment is built on top of BSCs Mercurium compiler and Nanos++ runtime system. (Most recent release: v17.06, June 2017)

Feedback for software developers

Awareness of the latest standards and the status of their implementations are critical at all times during application development. The adoption of new features from standards are likely to have large impact on the scalability of application codes precisely because it is very likely that these features exist to target the scalability challenges on modern systems. Nevertheless, you should be aware that there can be very long gaps between the publication of a standard and the implementation in compilers (which is frequently also biased by who is pushing which standard and why: Intel pushes OpenMP because of their Xeon Phi line, NVIDIA who now own PGI pushes OpenACC because of their GPUs, AMD pushed OpenCL for their own GPUs to compete with CUDA). The likelihood of there being a single common (set of) standards that performs well on all architectures is not high in the immediate future. For typical developers that we see in E-CAM, MPI+OpenMP remains the safest bet and is likely to perform well, as long as the latest standards are used.

More disruptive software technologies (such as GASNet) are more likely to gain traction if they are used by popular abstraction layers (which could be PGAS languages, runtime systems or even domain specific languages) “under the hood”. This would make the transition to new paradigms an implementation issue for the abstraction layer. Indeed, given the expected complexity of next generation machines, new programming paradigms that help remove the performance workload from the shoulders of scientific software developers will gain increasing importance.

As you may have noticed in the previous discussion, the computer scientists developing these abstractions are working mostly in C++, and the implementation of new standards in compilers is also seen first for C++. From a practical perspective this has some clear implications: if you want to access the latest software technologies then you had better consider C++ for your application. This may appear harsh given that the Fortran standard has clear capabilities in this space, but it is a current reality that cannot be ignored. Also, given that the vast majority of researchers will eventually transition to industry (because there simply aren’t enough permanent jobs in academia) it is more responsible to ensure they have programming expertise in a language that is heavily used in the commercial space. Finally, the ecosystem surrounding C++ (IDEs, testing frameworks, libraries, . . .) is much richer because of it’s use in industry and computer science.

Taking all of the above into consideration, if you are starting out with an application we would distil the discussion into the following advice: prototype your application using Python leveraging the Python APIs to the libraries you need; write unit tests as you go; and, when you start doing computationally intensive work, use C++ with Python interfaces to allow you to squeeze out maximal performance using the latest software technologies.

Accessing Resources

All of the above is academic unless you have access to resources to develop and test new software. There are many potential ways to access HPC resources, we simply highlight a limited set of the possibilities here.

Accessing HPC Resources in Europe

As far as the Partnership for Advanced Computing in Europe (PRACE) initiative is concerned, the complete list of available resources are shown in the figure below.

| | <i>JOLIOT CURIE - SKL</i> | <i>JOLIOT CURIE - KNL</i> | <i>Hazel Hen</i> | <i>JUWELS</i> | <i>Marconi Broadwell</i> | <i>Marconi KNL</i> | <i>MareNostrum 4</i> | <i>Piz Daint</i> | <i>SuperMUC Phase 2</i> | <i>SuperMUC-NG</i> |
|-------------------------|----------------------------------|---------------------------|--------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|--|--|--|
| System Type | Bull Sequana | Bull Sequana | Cray XC40 | Bull Sequana | Lenovo System NeXTScale | Lenovo System Adam Pass | Lenovo | Hybrid Cray xCS0 | Lenovo NeXTScale | Lenovo ThinkSystem |
| Processor type | Intel Xeon Platinum 8168 2.7 GHz | Intel Knights Landing | Intel Xeon E5-2680v3 (Haswell) | Intel Xeon Skylake Platinum 8168 | Intel Broadwell | Intel Knights Landing | Intel Xeon Platinum 8160 2.1 GHz | Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores) | Haswell Xeon E5-2697 v3 (Haswell) | Intel Skylake EP |
| Total nb of nodes | 1656 | 666 | 7 712 | 2511 | 720 | 3 600 | 3 456 | 5 320 | 3 072 | 3 072 |
| Total nb of cores | 79 488 | 45 288 | 185 088 | 120528 | 25 920 | 244 800 | 165 888 | 63 840 | 86 016 | 86 016 |
| Nb of accelerators/node | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | 1 GPU per node | n.a. | n.a. |
| Type of accelerator | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | NVIDIA® Tesla® P100 16GB | n.a. | n.a. |
| Memory / Node | 192 GB DDR4 | 96 GB DDR4 + 16 GB MCDRAM | 128 GB | 96 GB | 128 GB - DDR4 | 96 GB – DDR4 + 16 GB - MCDRAM | 96 GB (200 nodes with 384GB) | 64 GB | 64 GB | 96 GB |
| Network Type | Infiniband EDR | BULL BXI | Cray Aries | InfiniBand EDR | Intel Omni-Path Architecture 2:1 | Intel Omni-Path Architecture 2:1 | Intel Omni-Path Architecture | Cray Aries | Infiniband FDR14 | Intel Omni-Path Architecture |
| Connectivity | Fat Tree | Fat Tree | Dragonfly | Fat Tree | Fat Tree | Fat Tree | Fat Tree | Dragonfly | Fat tree within island (512 nodes) pruned tree between islands | Fat tree within island (512 nodes) pruned tree between islands |

Access to PRACE resources can be obtained by application to the [PRACE calls](#).

Moreover, the Distributed European Computing Initiative (DECI) is designed for projects requiring access to resources not currently available in the PI's own country but where those projects do not require resources on the very largest (Tier-0) European Supercomputers or very large allocations of CPU. To obtain resources from the DECI program, applications should be made via the [DECI calls](#).

The software stored in **E-CAM** repositories is developed via two main activities: the work of post-docs in the context of *pilot projects* with industrial partners; and the work of the participants at *Extended Software Development Workshops* (ESDWs).

6.1 Pilot Projects

One of primary activity of **E-CAM** is to engage with pilot projects with industrial partners. These projects are conceived together with the partner and typically are to facilitate or improve the scope of computational simulation within the partner. The related code development for the pilot projects are open source (where the licence of the underlying software allows this) and are described in the modules associated with the pilot projects.

Below is a list of the current pilot projects within **E-CAM**:

- *Classical MD Modules*
 - Binding kinetics
 - Food Proteins
- *Electronic Structure Modules*
 - Calculations for Applications in Photovoltaic Devices
 - Quantum Mechanical Parameterisation of Metal Ions in Proteins
 - Wannier90
- *Quantum Dynamics Modules*
 - Quantum Computing
- *Meso- and Multi-scale Modules*
 - Polarizable Mesoscale Models
 - Rheological Properties of New Composite Materials
 - The GC-AdResS scheme

6.2 Extended Software Development Workshops

E-CAM carries out 2 week software development workshops. These workshops train scientists in the development of modular codes for high performance machines. Documentation and testing are key components of the workshops and the associated on-line manuals and test cases are made available through the E-CAM module library. ESDWs are open to postdocs, senior graduate students and early career researchers in industry and academia. E-CAM carries out 4 ESDWs per year on each of the four scientific areas, with a maximum 2 weeks duration. For more information see <http://www.e-cam2020.eu/events/>

Contributing to this documentation

This documentation is created using ReStructured Text and the git repository for the documentation source files can be found at <https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library> which are open to contributions from anyone in the E-CAM community. If you would like to contribute to this effort then please follow the contribution guidelines that are linked to below.

General Information

- *How to contribute?*
 - *Contribution Guidelines*
 - * *GitLab account*
 - * *Fork the repository*
 - * *Clone your fork of the repository*
 - * *Keep your master branch up-to-date*
 - * *Branching*
 - * *Contributing module documentation*
 - * *Checking your contribution locally*
 - * *Contributing back your input*
 - * *Updating your contribution*

- [search](#)

7.1 How to contribute?

This webpage is actually a repository of files that (typically) document application development efforts during the pilot projects and Extended Software Development Workshops (ESDWs) of E-CAM. This documentation is completely open however and we welcome both internal and external contributions. If you would like to contribute to this effort then please follow the steps below to allow us to include your contribution.

In any case you will simply be adding a simple text file that uses [ReST](#) and we have prepared an example to help you get started:

- *E-CAM example module*

You will find the example within the repository of this documentation under the directory *example_module*. You should make a copy of this directory (renaming it) and place it in the appropriate scientific area directory.

7.1.1 Contribution Guidelines

GitLab account

If you do not have a (free) GitLab account yet on the E-CAM GitLab service, you'll need to get one via <https://gitlab.e-cam2020.eu/>.

Note: You should also register an SSH public key with GitLab (if you have not already done so), so you can easily clone, push to and pull from your repositories. This can be done via <https://gitlab.e-cam2020.eu/profile/keys> (once you're logged in on GitLab).

In the following it is assumed that an SSH public key has been registered with GitLab (see note above), the possibility of using the HTTPS protocol to access GitLab is not covered (but is possible).

Fork the repository

Firstly, you'll need to fork the repository on GitLab you want to work with. Go to <https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library>, and click the grey 'Fork' button either beside or under the repository name (or just click this [fork link](#)).

Clone your fork of the repository

Clone your fork of the repository to your favorite workstation.

```
git clone ssh://git@gitlab.e-cam2020.eu:10022/<Your GitLab username>/E-CAM-Library.git
```

Pull the master branch from the main repository:

```
cd E-CAM-Library
git remote add upstream https://git@gitlab.e-cam2020.eu/e-cam/E-CAM-Library.git
git pull upstream master
```

Keep your master branch up-to-date

Make sure you update it every time you create a feature branch (see below):

```
git checkout master
git pull upstream master
```

Branching

Pick a branch name for your work that makes sense, so you can track things easily and make sense if you end up having several branches in flight at once (each PR is a new branch).

Examples:

```
update_gromacs_module
new_esdw_lammps_module
industry_devel_module
```

Create a feature branch for your work (after updating your master), and check it out

```
git checkout master
git branch BRANCH_NAME
git checkout BRANCH_NAME
```

Make sure to always base your features branches on master!

If you are working on several things at the same time, try and keep things isolated in separate branches, to keep it manageable (both for you, and for reviewing your contributions).

Contributing module documentation

Your contribution to this repository will primarily be a module documentation file (this repository is not for source code, the documentation file will link to source code which is usually somewhere else). There are already several examples of these in the repository, but we provide a template for a generic module as a guide:

Software Technical Information

Name Name of the relevant software.

Language Please indicate the primary language(s) used by the module. Please also state if interfaces for other languages are available.

Licence Specify the licence under which the software is released. Provide a link to the full online description of the licence. You'll find descriptions of the most common licences at <https://opensource.org/licenses>. An example here would be: [GPL](#) or (the more permissive) [MIT](#)

Documentation Tool All source code created for this module should be documented so please indicate what tool has been used for documentation. Doxygen covers most languages but for Fortran you might want to use [Ford](#), for Python [ReST](#), etc.

Application Documentation Provide a link to any documentation for the application.

Relevant Training Material Add a link to any relevant training material. If there currently is none then say 'Not currently available.'

Software Module Developed by Add the name of the person who developed the software for this module here

E-CAM example module

- *Purpose of Module*
- *Background Information*
- *Building and Testing*
- *Source Code*

The E-CAM library is purely a set of documentation that describes software development efforts related to the project. A *module* for E-CAM is the documentation of the single development of effort associated to the project. In that sense, a module does not directly contain source code but instead contains links to source code, typically stored elsewhere. Each module references the source code changes to which it directly applies (usually via a URL), and provides detailed information on the relevant *application* for the changes as well as how to build and test the associated software.

The original source of this page (`readme.rst`) contains lots of additional comments to help you create your documentation *module* so please use this as a starting point. We use [Sphinx](#) (which in turn uses [ReST](#)) to create this documentation. You are free to add any level of complexity you wish (within the bounds of what [Sphinx](#) and [ReST](#) can do). More general instructions for making your contribution can be found in “[How to contribute?](#)”.

Remember that for a module to be accepted into the E-CAM repository, your source code changes in the target application must pass a number of acceptance criteria:

- Style (*use meaningful variable names, no global variables,...*)
- Source code documentation (*each function should be documented with each argument explained*)
- Tests (*everything you add should have either unit or regression tests*)
- Performance (*If what you introduce has a significant computational load you should make some performance optimisation effort using an appropriate tool. You should be able to verify that your changes have not introduced unexpected performance penalties, are threadsafe if needed,...*)

Purpose of Module

Give a brief overview of why the module is/was being created, explaining a little of the scientific background and how it fits into the larger picture of what you want to achieve. The overview should be comprehensible to a scientist non-expert in the domain area of the software module.

This section should also include the following (where appropriate):

- Who will use the module? in what area(s) and in what context?
- What kind of problems can be solved by the code?
- Are there any real-world applications for it?
- Has the module been interfaced with other packages?
- Was it used in a thesis, a scientific collaboration, or was it cited in a publication?
- If there are published results obtained using this code, describe them briefly in terms readable for non-expert users. If you have few pictures/graphs illustrating the power or utility of the module, please include them with corresponding explanatory captions.

Note: If the module is an ingredient for a more general workflow (e.g. the module was the necessary foundation for later code; the module is part of a group of modules that will be used to calculate certain property or have certain application, etc.) mention this, and point to the place where you specify the applications of the more general workflow (that could be in another module, in another section of this repository, an application’s website, etc.).

Note: If you are a post-doc who works in E-CAM, an obvious application for the module (or for the group of modules that this one is part of) is your pilot project. In this case, you could point to the pilot project page on the main website (and you must ensure that this module is linked there).

If needed you can include latex mathematics like $\frac{\sum_{t=0}^N f(t,k)}{N}$ which won't show up on GitLab/GitHub but will in final online documentation.

If you want to add a citation, such as [CIT2009], please check the source code to see how this is done. Note that citations may get rearranged, e.g., to the bottom of the “page”.

Background Information

If the modifications are to an existing code base (which is typical) then this would be the place to name that application. List any relevant urls and explain how to get access to that code. There needs to be enough information here so that the person reading knows where to get the source code for the application, what version this information is relevant for, whether this requires any additional patches/plugins, etc.

Overall, this module is supposed to be self-contained, but linking to specific URLs with more detailed information is encouraged. In other words, the reader should not need to do a websearch to understand the context of this module, all the links they need should be already in this module.

Building and Testing

Provide the build information for the module here and explain how tests are run. This needs to be adequately detailed, explaining if necessary any deviations from the normal build procedure of the application (and links to information about the normal build process needs to be provided).

Source Code

Here link the source code *that was created for the module*. If you are using Github or GitLab and the [Gitflow Workflow](#) you can point to your feature branch. Linking to your pull/merge requests is even better. Otherwise you can link to the explicit commits.

- [Link to a merge request containing my source code changes](#)

There may be a situation where you cannot do such linking. In this case, I'll go through an example that uses a patch file to highlight my source code changes, for that reason I would need to explain what code (including exact version information), the source code is for.

You can create a similar patch file by (for example if you are using git for your version control) making your changes for the module in a feature branch and then doing something like the following:

```
[adam@mbp2600 example (master)]$ git checkout -b tmpsquash
Switched to a new branch "tmpsquash"

[adam@mbp2600 example (tmpsquash)]$ git merge --squash newlines
Updating 4d2de39..b6768b2
Fast forward
Squash commit -- not updating HEAD
 test.txt |    2 ++
 1 files changed, 2 insertions(+), 0 deletions(-)
```

(continues on next page)

(continued from previous page)

```
[adam@mbp2600 example (tmpsquash)]$ git commit -a -m "My squashed commits"
[tmpsquash]: created 75b0a89: "My squashed commits"
1 files changed, 2 insertions(+), 0 deletions(-)

[adam@mbp2600 example (tmpsquash)]$ git format-patch master
0001-My-squashed-commits.patch
```

To include a patch file do something like the following (take a look at the source code of this document to see the syntax required to get this):

```
1 Always remember that a good patch file should have a comment inside about what the
  ↳ patch is for....just like this
2 --- hello.c      2014-10-07 18:17:49.000000000 +0530
3 +++ hello_new.c  2014-10-07 18:17:54.000000000 +0530
4 @@ -1,5 +1,6 @@
5  #include <stdio.h>
6
7 -int main() {
8 +int main(int argc, char *argv[]) {
9     printf("Hello World\n");
10 +    return 0;
11 }
```

If the patch is very long you will probably want to add it as a subpage which can be done as follows

Patch file for module

Downloadable version of patch file

To include a patch file do something like the following:

```
1 Always remember that a good patch file should have a comment inside about what the
  ↳ patch is for....just like this
2 --- hello.c      2014-10-07 18:17:49.000000000 +0530
3 +++ hello_new.c  2014-10-07 18:17:54.000000000 +0530
4 @@ -1,5 +1,6 @@
5  #include <stdio.h>
6
7 -int main() {
8 +int main(int argc, char *argv[]) {
9     printf("Hello World\n");
10 +    return 0;
11 }
```

you can reference it with *Patch file for module*

After creating the branch, implement your contributions: new modules, enhancements or updates to existing modules, bug fixes, structure changes, whatever you like.

Make sure to put your work in the appropriate directory. There are 4 scientific areas in E-CAM and your module is likely most relevant in one of those. Each of these directories has a `modules` subdirectory, create a new directory within it to contain all files relevant to your specific module. In the example below, this directory has been called `gromacs_gpu`.

Make sure you commit your work, and try to do it in bite-size chunks, so the commit log remains clear, for example:

```
git add Classical-MD-Modules/modules/gromacs_gpu/readme.rst
git commit -m "add details on improved GPU support within GROMACS"
```

Checking your contribution locally

You can locally build the documentation to check that the changes you make look as you expect them. To do this you will need the Sphinx python package to be installed (see this [installation link](#) for information on how to install this tool on your operating system).

```
make html # in root directory of repository
firefox _build/html/index.html # Use your browser to view the end result
```

If you do not have Latex installed on your system you are likely to get related errors. Other (non-latex) errors are likely to come from your additions.

Contributing back your input

When you've finished the implementation of a particular contribution, here's how to get it into the main repository.

Push your branch to *your* copy of the repository on GitLab

```
git push origin <BRANCH_NAME>
```

Issue a *Merge Request* for your branch into the main repository. To do this go to https://gitlab.e-cam2020.eu/Your_GitLab_Username/E-CAM-Library/merge_requests and select the *New Merge Request* button.

Make sure the branch you just pushed is selected (not master!) issue a merge request for your branch to the master branch of the main repository.

Updating your contribution

It is common for there to be updates required to contributions, you do **not** need to open a new Merge Request to do this.

To update your contribution you update the appropriate files on your contribution branch. Firstly you need to ensure that you are up to date with the remote repository on GitLab. Make sure you are in the directory of the cloned repository and then check which branch you want to check out:

```
git branch # List all available local branches, to include remote branches add the -r_
↳flag
git checkout <BRANCH_NAME> # Check out the branch we want to update
git pull origin <BRANCH_NAME> # Make sure we have any updates we made to our own_
↳branch
git pull upstream master # Also pull in any changes to the main repository
```

Now that everything is in sync, you can edit and update your files, when you are finished you commit your changes and push the changes back to GitLab:

```
git add modules/gromacs_gpu/readme.rst
git commit -m "update documentation on how to trigger the GPU support"
git push origin <BRANCH_NAME>
```

The Merge Request will now be automatically updated with the changed files.

Bibliography

- [Swenson2014] D.W.H. Swenson and P.G. Bolhuis, J. Chem. Phys. **141**, 044101 (2014); <https://doi.org/10.1063/1.4890037>
- [Rogal2008] J. Rogal and P.G. Bolhuis, J. Chem. Phys. **129**, 224107 (2008); <https://doi.org/10.1063/1.3029696>
- [GaSc2018] <https://doi.org/10.1063/1.5019667>
- [ImAn2018] <https://doi.org/10.1063/1.5024611>
- [BePa2007] <https://doi.org/10.1103/PhysRevLett.98.146401>
- [Beh2011] <https://doi.org/10.1063/1.3553717>
- [Sirk2012] An enhanced entangled polymer model for dissipative particle dynamics, J. Chem. Phys. **136**, 134903 (2012); <https://doi.org/10.1063/1.3698476>
- [Kumar2001] Brownian dynamics simulations of flexible polymers with spring–spring repulsions, J. Chem. Phys. **114**, 6937, (2001); <https://doi.org/10.1063/1.1358860>
- [Vitalis2012] A. Vitalis and A. Caflisch. Efficient Construction of Mesostate Networks from Molecular Dynamics Trajectories. J. Chem. Theory Comput. **8** (3), 1108–1120 (2012) DOI
- [Blum2009] Blum, V., et al. (2009). Ab initio molecular simulations with numeric atom-centered orbitals. CPC, **180** (11), 2175–2196. <https://doi.org/10.1016/j.cpc.2009.06.022>
- [Fuchs1999] Fuchs, Martin, and Matthias Scheffler. “Ab initio pseudopotentials for electronic structure calculations of poly-atomic systems using density-functional theory.” CPC **119.1** (1999): 67–98.
- [DFTB] B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshayé, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, and T. Frauenheim, “DFTB+, a software package for efficient approximate density functional theory based atomistic simulations”, The Journal of Chemical Physics **152**, 124101 (2020) <https://doi.org/10.1063/1.5143190>
- [DION2004] Dion et al., Phys. Rev. Lett. **92**, 246401 (2004).
- [LEE2010] Lee et al., Phys. Rev. B **82**, 081101 (2010).
- [KLIMES2009] Klimes et al., J. Phys. Cond. Matt. **22**, 022201 (2009).

- [VYDROV2010] Vydrov, VanVoorhis, J. Chem. Phys. 133, 244103 (2010).
- [ROMAN2009] Román-Pérez, Soler, Phys. Rev. Lett. 103, 096102 (2009).
- [BONELLA2020] Phys. Chem. Chem. Phys., 2020, 22, 10775-10785
- [BONELLA2020b] Phys. Chem. Chem. Phys., 2020, 22, 10775-10785
- [BONELLA2020a] Phys. Chem. Chem. Phys., 2020, 22, 10775-10785
- [Storn1997] Storn, Rainer, and Kenneth Price. "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces." *Journal of global optimization* 11.4 (1997): 341-359.
- [FF2018] Fracchia F., Del Frate G., Mancini G., Rocchia W., Barone V., Force Field Parametrization of Metal Ions from Statistical Learning Techniques. *J. Chem. Theory Comput.*, 2018, 14(1), pp 255-273
- [vitale2019] [arXiv:1909.00433](https://arxiv.org/abs/1909.00433) [physics.comp-ph]
- [Hoja_ea] First-principles modelling of molecular crystals: structures and stabilities, temperature and pressure. <https://doi.org/10.1002/wcms.1294>
- [Errea_ea] Anharmonic free energies and phonon dispersions from the stochastic self-consistent harmonic approximation: Application to platinum and palladium hydrides. <https://doi.org/10.1103/PhysRevB.89.064302>
- [Christiansen] Vibrational structure theory: new vibrational wave function methods for calculation of anharmonic vibrational energies and vibrational contributions to molecular properties. <https://doi.org/10.1039/B618764A>
- [Hoja_ea1] First-principles modelling of molecular crystals: structures and stabilities, temperature and pressure. <https://doi.org/10.1002/wcms.1294>
- [Lloyd-Williams_Monserrat] Lattice dynamics and electron-phonon coupling calculations using nondiagonal supercells. <https://doi.org/10.1103/PhysRevB.92.184301>
- [Errea_ea1] Anharmonic free energies and phonon dispersions from the stochastic self-consistent harmonic approximation: Application to platinum and palladium hydrides. <https://doi.org/10.1103/PhysRevB.89.064302>
- [Christiansen1] Vibrational structure theory: new vibrational wave function methods for calculation of anharmonic vibrational energies and vibrational contributions to molecular properties. <https://doi.org/10.1039/B618764A>
- [Lloyd-Williams_Monserrat1] Lattice dynamics and electron-phonon coupling calculations using nondiagonal supercells. <https://doi.org/10.1103/PhysRevB.92.184301>
- [Pulay] Convergence acceleration of iterative sequences. The case of scf iteration. [https://doi.org/10.1016/0009-2614\(80\)80396-4](https://doi.org/10.1016/0009-2614(80)80396-4)
- [LCT1] B. F. E. Curchod, T. J. Penfold, U. Rothlisberger, I. Tavernelli *Phys. Rev. A* **84** (2012) 042507 DOI: [10.1103/PhysRevA.84.042507](https://doi.org/10.1103/PhysRevA.84.042507)
- [LCT2] B. F. E. Curchod, T. J. Penfold, U. Rothlisberger, I. Tavernelli *Chem. Phys. Chem.* **16** (2015) 2127 DOI: [10.1002/cphc.201500190](https://doi.org/10.1002/cphc.201500190)
- [Mackernan1] D.Mackernan, G.Ciccotti, R.Kapral, *Trotter-Based Simulation of Quantum-Classical Dynamics*, *J. Phys. Chem. B*, **2008**, 112 (2), pp 424-432.
- [CTMQC1] S. K. Min, F. Agostini, E. K. U. Gross, *Phys. Rev. Lett.* **115** (2015) 073001 DOI: [10.1103/PhysRevLett.115.073001](https://doi.org/10.1103/PhysRevLett.115.073001)
- [CTMQC2] F. Agostini, S. K. Min, A. Abedi, E. K. U. Gross, *J. Chem. Theory Comput* **5** (2016) 2127 DOI: [10.1021/acs.jctc.5b01180](https://doi.org/10.1021/acs.jctc.5b01180)
- [CTMQC3] Graeme H. Gossel, F. Agostini, Neepa T. Maitra, (2018) [arXiv: 1805.03534](https://arxiv.org/abs/1805.03534) [physics.chem-ph]

- [CTMQC4] S. K. Min, Federica Agostini, I. Tavernelli, E. K. U. Gross, *J. Phys. Chem. Lett.* **8** (2017) 3048 DOI: [10.1021/acs.jpclett.7b01249](https://doi.org/10.1021/acs.jpclett.7b01249)
- [EF1] A. Abedi, N. T. Maitra, E. K. U. Gross, *Phys. Rev. Lett.* **105** (2010) 123002 DOI: [10.1103/PhysRevLett.105.123002](https://doi.org/10.1103/PhysRevLett.105.123002)
- [EF2] A. Abedi, N. T. Maitra, E. K. U. Gross, *J. Chem. Phys.* **137** (2012) 22A530 DOI: [10.1063/1.4745836](https://doi.org/10.1063/1.4745836)
- [EF3] A. Abedi, F. Agostini, Y. Suzuki, E. K. U. Gross, *Phys. Rev. Lett.* **110** (2013) 263001 DOI: [10.1103/PhysRevLett.110.263001](https://doi.org/10.1103/PhysRevLett.110.263001)
- [EF4] F. Agostini, B. F. E. Curchod, R. Vuilleumier, I. Tavernelli, E. K. U. Gross, *TDDFT and Quantum-Classical Dynamics: a Universal Tool Describing the Dynamics of Matter* in ‘Handbook of Materials Modeling. Volume 1 Methods: Theory and Modeling’, edited by Wanda Andreoni and Sidney Yip, Springer (in production).
- [Tully] J. C. Tully, *J. Chem. Phys.* **93** (1990) 1061 DOI: [10.1063/1.459170](https://doi.org/10.1063/1.459170)
- [CI1] B. F. E. Curchod, F. Agostini, *J. Phys. Chem. Lett.* **8** (2017) 831 DOI: [10.1021/acs.jpclett.7b00043](https://doi.org/10.1021/acs.jpclett.7b00043)
- [Gross_PRL2010] Abedi, A., Maitra, N. T., Gross, E. K. U. *Phys. Rev. Lett.* **105** (2010) 123002 Exact factorization of the time-dependent electron-nuclear wave function.
- [Gross_JCP2012] Abedi, A., Maitra, N. T., Gross, E. K. U. *Phys. Rev. Lett.* **137** (2012) 22A530 Correlated electron-nuclear dynamics: Exact factorization of the molecular wave-function.
- [Gross_MP2013] Agostini, F., Abedi, A., Suzuki, Y., and Gross, E. K. U. *Mol. Phys.* **111** (2013) 3625–3640 Mixed quantum-classical dynamics on the exact time-dependent potential energy surfaces: A fresh look at non-adiabatic processes.
- [Gross_JCP2015] Agostini, F., Abedi, A., Suzuki, Y., Min, S. K., Maitra, N. T., and Gross, E. K. U. *J. Chem. Phys.* **142** (2015) 084303 The exact forces on classical nuclei in non-adiabatic charge transfer.
- [Agostini_JPCL2017] Curchod, B. F. E., and Agostini, F. *J. Phys. Chem. Lett.* **105** (2017) 831–837 On the dynamics through a conical intersection.
- [ElVibRot] Lauvergnat, D. *J. Chem. Phys.* [Elvibrot: Quantum dynamics code](#).
- [Mackernan] D. Mackernan, G. Ciccotti, R. Kapral, *Trotter-Based Simulation of Quantum-Classical Dynamics*, *J. Phys. Chem. B*, **2008**, 112 (2), pp 424–432.
- [EF] F. Agostini, E. K. U. Gross, *Quantum chemistry and dynamics of excited states: Methods and applications*, edited by L. González and R. Lindh, Wiley (2020).
- [CT-MQC] S. K. Min, F. Agostini, E. K. U. Gross, *Phys. Rev. Lett.* **115** (2015) 073001 DOI: [10.1103/PhysRevLett.115.073001](https://doi.org/10.1103/PhysRevLett.115.073001)
- [TSH] J. C. Tully, *J. Chem. Phys.* **93** (1990) 1061 DOI: [10.1063/1.459170](https://doi.org/10.1063/1.459170)
- [EH] J. C. Tully, *Faraday Discuss.* **110** (1998) 407 DOI: [10.1039/A801824C](https://doi.org/10.1039/A801824C)
- [TSH-EDC] G. Granucci, M. Persico, *J. Chem. Phys.* **126** (2007) 134114 DOI: [10.1063/1.2715585](https://doi.org/10.1063/1.2715585)
- [G-CT-MQC] F. Talotta, S. Morisset, N. Rougeau, D. Lauvergnat, F. Agostini, *J. Chem. Theory Comput.* **16** (2020) 4833–4848 DOI: [10.1021/acs.jctc.0c00493](https://doi.org/10.1021/acs.jctc.0c00493)
- [PSB3] E. Marsili, M. Olivucci, D. Lauvergnat and F. Agostini, *J. Chem. Theory Comput.* **16** (2020) 6032–6048 DOI: [10.1021/acs.jctc.0c00679](https://doi.org/10.1021/acs.jctc.0c00679)
- [SOC] F. Talotta, S. Morisset, N. Rougeau, D. Lauvergnat, F. Agostini, *Phys. Rev. Lett.* **124** (2020) 033001 DOI: [10.1103/PhysRevLett.124.033001](https://doi.org/10.1103/PhysRevLett.124.033001)
- [IC] C. Pieoroni, E. Marsili, D. Lauvergnat and F. Agostini, *J. Chem. Phys.* **154** (2021) 034104 DOI: [10.1063/5.0036726](https://doi.org/10.1063/5.0036726)

- [KapralCiccotti1999] R. Kapral, G. Ciccotti, *J. Chem. Phys.* **110** (1999) 8919 DOI: [10.1063/1.478811](https://doi.org/10.1063/1.478811)
- [HsiehKapral2012] C. Hsieh, K. Raymond, *J. Chem. Phys.* **137** (2012) 22A507 DOI: [10.1063/1.4736841](https://doi.org/10.1063/1.4736841)
- [HsiehKapral2013] C. Hsieh, K. Raymond, *J. Chem. Phys.* **138** (2013) 134110 DOI: [10.1063/1.4798221](https://doi.org/10.1063/1.4798221)
- [IshizakiFleming2009] A. Ishizaki, G. R. Fleming, *J. Chem. Phys.* **130** (2009) 234111 DOI: [10.1063/1.3155372](https://doi.org/10.1063/1.3155372)
- [IshizakiFleming2009PNAS] A. Ishizaki, G. R. Fleming, *PNAS* **106** (2009) 17255 DOI: [10.1073/pnas.0908989106](https://doi.org/10.1073/pnas.0908989106)
- [WilkinsDattani2015] D. Wilkins, N. Dattani, *J. Chem. Theory Comput.* (2015) 3411 DOI: [10.1021/ct501066k](https://doi.org/10.1021/ct501066k)
- [PMon1] M. Monteferrante, S. Bonella, G. Ciccotti *Mol. Phys.* **109** (2011) 3015 DOI: [10.1080/00268976.2011.619506](https://doi.org/10.1080/00268976.2011.619506)
- [PMon2] M. Monteferrante, S. Bonella, G. Ciccotti *J. Chem. Phys.* **138** (2013) 054118 DOI: [10.1063/1.4789760](https://doi.org/10.1063/1.4789760)
- [PBeu] J. Beutier, M. Monteferrante, S. Bonella, R. Vuilleumier, G. Ciccotti *Mol. Sim.* **40** (2014) 196 DOI: [10.1080/08927022.2013.843776](https://doi.org/10.1080/08927022.2013.843776)
- [PJin] Z. Jin, B. Braams, J. Bowman *J. Phys. Chem. A* **110** (2006) 1569 DOI: [10.1021/jp053848o](https://doi.org/10.1021/jp053848o)
- [Pen] D. M. Ceperley, M. Dewing *J. Chem. Phys.* **110** (1999) 9812 DOI: <http://dx.doi.org/10.1063/1.478034>
- [Ken] A. D. Kennedy, J. Kuti *Phys. Rev. Lett.* **54** (1985) 2473 DOI: <https://doi.org/10.1103/PhysRevLett.54.2473>
- [Dam] H. Dammak, Y. Chalopin, M. Laroche, M. Hayoun, J.-J. Greffet, Quantum Thermal Bath for Molecular Dynamics Simulation, *Phys. Rev. Lett.* **103** (2009) 190601.
- [Man] E. Mangaud, S. Huppert, T. Plé, P. Depondt, S. Bonella, F. Finocchi, The Fluctuation–Dissipation Theorem as a Diagnosis and Cure for Zero-Point Energy Leakage in Quantum Thermal Bath Simulations, *J. Chem. Th. Comput.* **15** (2019) 2863–2880.
- [Bri] F. Brieuc, Y. Bronstein, H. Dammak, P. Depondt, F. Finocchi, M. Hayoun, Zero-point energy leakage in quantum thermal bath molecular dynamics simulations, *J. Chem. Th. Comput.* **12** (2016) 5688–5697.
- [Hern] J. Hern'andez-Rojas, F. Calvo, E. G. Noya, Applicability of Quantum Thermal Baths to Complex Many-Body Systems with Various Degrees of Anharmonicity, *Journal of Chemical Theory and Computation* **11** (2015) 861–870.
- [Lei] B. Leimkuhler, C. Matthews, Rational Construction of Stochastic Numerical Methods for Molecular Sampling, *Applied Mathematics Research eXpress* (2012).
- [Jin] Z. Jin, B. Braams, J. Bowman *J. Phys. Chem. A* **110** (2006) 1569 DOI: [10.1021/jp053848o](https://doi.org/10.1021/jp053848o)
- [Lef] C. Leforestier, R. H. Bisseling, C. Cerjan, M. D. Feit, R. Friesner, A. Guldborg, A. Hammerich, G. Jolicard, W. Karrlein, H.-D. Meyer, N. Lipkin, O. Roncero, R. Kosloff *J. Comp. Phys.* **94** (1991) 59 DOI: [https://doi.org/10.1016/0021-9991\(91\)90137-A](https://doi.org/10.1016/0021-9991(91)90137-A)
- [1Lef] C. Leforestier, R. H. Bisseling, C. Cerjan, M. D. Feit, R. Friesner, A. Guldborg, A. Hammerich, G. Jolicard, W. Karrlein, H.-D. Meyer, N. Lipkin, O. Roncero, R. Kosloff *J. Comp. Phys.* **94** (1991) 59 DOI: [https://doi.org/10.1016/0021-9991\(91\)90137-A](https://doi.org/10.1016/0021-9991(91)90137-A)
- [1Tal] H. Tal-Ezer, R. Kosloff *J. Chem. Phys.* **81** (1984) 3967 DOI: [10.1063/1.448136](https://doi.org/10.1063/1.448136)
- [Tn1] D. Lauvergnat, A. Nauts, *Phys. Chem. Chem. Phys.* **12** (2010) 8405–8412 DOI: [10.1039/C001944E](https://doi.org/10.1039/C001944E)
- [Sm1] S. A. Smolyak, *Dokl. Akad. Nauk SSSR* **148** (1963) 1042–1045 <http://mi.mathnet.ru/eng/dan27586>
- [Tn] D. Lauvergnat, A. Nauts, *Phys. Chem. Chem. Phys.* **12** (2010) 8405–8412 DOI: [10.1039/C001944E](https://doi.org/10.1039/C001944E)
- [Sm] S. A. Smolyak, *Dokl. Akad. Nauk SSSR* **148** (1963) 1042–1045 <http://mi.mathnet.ru/eng/dan27586>
- [Curc] B. F. E. Curchod, T. J. Penfold, U. Rothlisberger, I. Tavernelli *Phys. Rev. A* **84** (2012) 042507 DOI: [10.1103/PhysRevA.84.042507](https://doi.org/10.1103/PhysRevA.84.042507)

- [Mon1] M. Monteferrante, S. Bonella, G. Ciccotti *Mol. Phys.* **109** (2011) 3015 DOI: [10.1080/00268976.2011.619506](https://doi.org/10.1080/00268976.2011.619506)
- [Mon2] M. Monteferrante, S. Bonella, G. Ciccotti *J. Chem. Phys.* **138** (2013) 054118 DOI: [10.1063/1.4789760](https://doi.org/10.1063/1.4789760)
- [Beu] J. Beutier, M. Monteferrante, S. Bonella, R. Vuilleumier, G. Ciccotti *Mol. Sim.* **40** (2014) 196 DOI: [10.1080/08927022.2013.843776](https://doi.org/10.1080/08927022.2013.843776)
- [ZJin] Z. Jin, B. Braams, J. Bowman *J. Phys. Chem. A* **110** (2006) 1569 DOI: [10.1021/jp053848o](https://doi.org/10.1021/jp053848o)
- [Beck] M. Beck, A. Jäckle, G.A. Worth, and H.-D. Meyer *Phys. Rep.* **324** (2000) 1–106 DOI: [10.1016/S0370-1573\(99\)00047-2](https://doi.org/10.1016/S0370-1573(99)00047-2)
- [Lefo] C. Leforestier, R. H. Bisseling, C. Cerjan, M. D. Feit, R. Friesner, A. Guldberg, A. Hammerich, G. Jolicard, W. Karrlein, H.-D. Meyer, N. Lipkin, O. Roncero, R. Kosloff *J. Comp. Phys.* **94** (1991) 59 DOI: [10.1016/0021-9991\(91\)90137-A](https://doi.org/10.1016/0021-9991(91)90137-A)
- [Mey] H.-D. Meyer, G. A. Worth *Theor. Chem. Acc.* **109** (2003) 251 DOI: [10.1007/s00214-003-0439-1](https://doi.org/10.1007/s00214-003-0439-1)
- [Ric] G. W. Richings, I. Polyak, K. E. Spinlove, G. A. Worth, I. Burghardt, B. Lasorne *Int. Rev. Phys. Chem.* **34** (2015) 269 DOI: [10.1080/0144235X.2015.1051354](https://doi.org/10.1080/0144235X.2015.1051354)
- [Wor1] G. A. Worth, M. A. Robb, B. L. Lasorne *Mol. Phys.* **106** (2008) 2077–2091 DOI: [10.1080/00268970802172503](https://doi.org/10.1080/00268970802172503)
- [Wor2] G. A. Worth, K. Giri, G. W. Richings, M. H. Beck, A. Jackle, H.-D. Meyer Quantics package, version 1.1, (2015)
- [Tnum] D. Lauvergnat, A. Nauts, *Phys. Chem. Chem. Phys.* **12** (2010) 8405–8412 DOI: [10.1039/C001944E](https://doi.org/10.1039/C001944E)
- [Smo] S. A. Smolyak, *Dokl. Akad. Nauk SSSR* **148** (1963) 1042–1045 <http://mi.mathnet.ru/eng/dan27586>
- [Duboue2015] E. Duboué-Dijon, A. Laage, *Characterization of the local structure in liquid water by various order parameters*, *J. Phys. Chem. B*, **119**, 8406 (2015).
- [SPME] J. Chem. Phys. 103, 8577 (1995)
- [Coveney] Coveney, P. V. et al., Towards blood flow in the virtual human: efficient self-coupling of HemeLB, *Interface focus* 11(1), 20190119
- [Raymond] The Art of Unix Programming, Eric Steven Raymond <http://www.faqs.org/docs/artu/index.html>
- [BestPractices] https://en.wikibooks.org/wiki/Computer_Programming/Standards_and_Best_Practices
- [SoftwareSpecification] Software requirements specification https://en.wikipedia.org/wiki/Software_requirements_specification
- [TDD] Test-driven Development https://en.wikipedia.org/wiki/Test-driven_development
- [PyCogent] http://pycogent.org/coding_guidelines.html
- [CIT2009] This is a citation (as often used in journals).