PIConGPU Documentation

Release 0.8.0-dev

The PIConGPU Community

Apr 24, 2024

INSTALLATION

Particle-in-Cell Simulations for the Exascale Era

PIConGPU is a fully relativistic, manycore, 3D3V and 2D3V particle-in-cell (PIC) code. The PIC algorithm is a central tool in plasma physics. It describes the dynamics of a plasma by computing the motion of electrons and ions in the plasma based on the Vlasov-Maxwell system of equations.

Generally, to get started **follow the manual pages in order**. Individual chapters are based on the information in the chapters before. In case you are already fluent in compiling C++ projects and HPC, running PIC simulations or scientific data analysis, feel free to jump the respective sections.

Note: We are migrating our wiki to this manual, but some pages might still be missing. We also have an official homepage .

CHAPTER

ONE

INTRODUCTION

Section author: Axel Huebl

Installing PIConGPU means *installing C++ libraries* that PIConGPU depends on and *setting environment variables* to find those dependencies. The first part is usually the job of a system administrator while the second part needs to be configured on the user side.

Depending on your experience, role, computing environment and expectations for optimal hardware utilization, you have several ways to install and select PIConGPU's dependencies. Choose your favorite *install and environment management method* below, young padawan, and follow the corresponding sections of the next chapters.

1.1 Ways to Install

Choose *one* of the installation methods below to get started.

1.1.1 Load Modules

On HPC systems and clusters, software is usually provided by system administrators via a module system (e.g. [modules], [Lmod]). In case our *software dependencies* are available, we usually create a file in our `$HOME` named *<queueName>_picongpu.profile*. It loads according modules and sets *helper environment variables*.

Important: For many HPC systems we have already prepared and maintain an environment which will run out of the box. See if your system is *in the list* so you can skip the installation completely!

1.1.2 Build from Source

You choose a supported C++ compiler and configure, compile and install all missing dependencies from source. You are responsible to manage the right versions and configurations. Performance will be ideal if architecture is chosen correctly (and/or if built directly on your hardware). You then set environment variables to find those installs.

1.1.3 Conda

We currently do not have an official conda install (yet). Due to pre-build binaries, performance could be not ideal and HPC cluster support (e.g. MPI) might be very limited. Useful for small desktop or single-node runs.

1.2 References

CHAPTER

TWO

INSTRUCTIONS

Section author: Axel Huebl

As explained in the previous section, select and **follow exactly one** of the following install options.

See also:

You will need to understand how to use the terminal.

Note: This section is a short introduction in case you are missing a few software packages, want to try out a cutting edge development version of a software or have no system administrator or software package manager to build and install software for you.

2.1 From Source

Section author: Axel Huebl

Don't be afraid, young physicist, self-compiling C/C++ projects is easy, fun and profitable!

Building a project from source essentially requires three steps:

1. configure the project and find its dependencies
2. compile the project
3. install the project

All of the above steps can be performed without administrative rights ("root" or "superuser") as long as the install is not targeted at a system directory (such as `/usr`) but inside a user-writable directory (such as `$HOME` or a project directory).

2.1.1 Preparation

In order to compile projects from source, we assume you have individual directories created to store *source code*, *build temporary files* and *install* the projects to:

```
# source code
mkdir $HOME/src
# temporary build directory
mkdir $HOME/build
# install target for dependencies
mkdir $HOME/lib
```

Note that on some supercomputing systems, you might need to install the final software outside of your home to make dependencies available during run-time (when the simulation runs). Use a different path for the last directory then.

2.1.2 What is Compiling?

Note: This section is **not** yet the installation of PIConGPU from source. It just introduces in general how one compiles projects.

If you like to skip this introduction, *jump straight to the dependency install section*.

Compling can differ in two principle ways: building *inside* the source directory ("in-source") and in a *temporary directory* ("out-of-source"). Modern projects prefer the latter and use a build system such as [CMake].

An example could look like this

```
# go to an empty, temporary build directory
cd $HOME/build
rm -rf ../build/*

# configurate, build and install into $HOME/lib/project
cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/project $HOME/src/project_to_compile
make
make install
```

Often, you want to pass further options to CMake with `-DOPTION=VALUE` or modify them interactively with `ccmake .` after running the initial cmake command. The second step which compiles the project can in many cases be parallelized by `make -j`. In the final install step, you might need to prefix it with `sudo` in case `CMAKE_INSTALL_PREFIX` is pointing to a system directory.

Some older projects often build *in-source* and use a build system called *autotools*. The syntax is still very similar:

```
# go to the source directory of the project
cd $HOME/src/project_to_compile

# configurate, build and install into $HOME/lib/project
configure --prefix=$HOME/lib/project
make
make install
```

One can usually pass further options with `--with-something=VALUE` or `--enable-thing` to `configure`. See `configure --help` when installing an *autotools* project.

That is all on the theory of building projects from source!

2.1.3 Now Start

You now know all the basics to install from source. Continue with the following section to *build our dependencies*.

2.1.4 References

If anything goes wrong, an overview of the full list of PIConGPU dependencies is provided in *section Dependencies*.

After installing, the last step is the setup of a *profile*.

See also:

You will need to understand how to use the terminal, what are environment variables and please read our *compiling introduction*.

Note: If you are a scientific user at a super computing facility we might have already prepared a software setup for you. See the *following chapter* if you can skip this step fully or in part by loading existing modules on those systems.

CHAPTER THREE

DEPENDENCIES

Section author: Axel Huebl, Klaus Steiniger, Sergei Bastrakov, Rene Widera

3.1 Overview

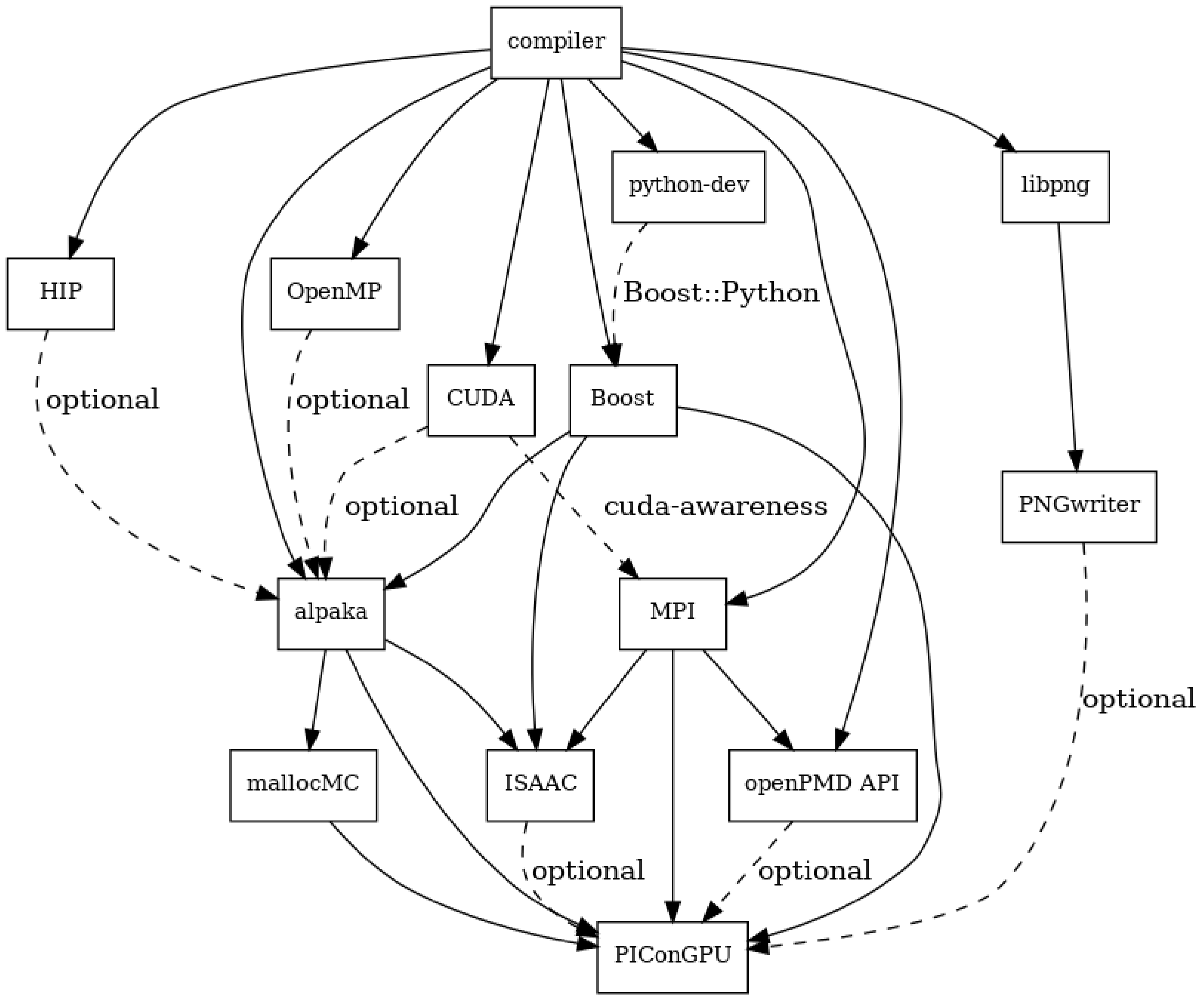


Fig. 3.1: Overview of inter-library dependencies for parallel execution of PIConGPU on a typical HPC system. Due to common binary incompatibilities between compilers, MPI and boost versions, we recommend to organize software with a version-aware package manager such as spack and to deploy a hierarchical module system such as lmod. An Lmod example setup can be found here.

3.2 Requirements

3.2.1 Mandatory

Compiler

- C++17 supporting compiler, e.g. GCC 9+ or Clang 11+
- if you want to build for Nvidia GPUs, check the CUDA supported compilers page
- *note:* be sure to build all libraries/dependencies with the *same* compiler version
- *Debian/Ubuntu:*
  - `sudo apt-get install gcc-9 g++-9 build-essential`
  - `sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 60 --slave /usr/bin/g++ g++ /usr/bin/g++-9`
- *Arch Linux:*
  - `sudo pacman --sync base-devel`
  - if the installed version of **gcc** is too new, compile an older gcc
- *Spack:*
  - `spack install gcc@12.2.0`
  - make it the default in your packages.yaml or *suffix* all following `spack install` commands with a *space* and `%gcc@12.2.0`

CMake

- 3.22.0 or higher
- *Debian/Ubuntu:* `sudo apt-get install cmake file cmake-curses-gui`
- *Arch Linux:* `sudo pacman --sync cmake`
- *Spack:* `spack install cmake`

MPI 2.3+

- **OpenMPI** 1.7+ / **MVAPICH2** 1.8+ or similar
- for running on Nvidia GPUs, perform a GPU aware MPI install *after* installing CUDA
- *Debian/Ubuntu:* `sudo apt-get install libopenmpi-dev`
- *Arch Linux:* `sudo pacman --sync openmpi`
- *Spack:*
  - *GPU support:* `spack install openmpi+cuda`
  - *CPU only:* `spack install openmpi`
- *environment:*
  - `export MPI_ROOT=<MPI_INSTALL>`
  - as long as CUDA awareness (`openmpi+cuda`) is missing: `export OMPI_MCA_mpi_leave_pinned=0`

Boost

- 1.74.0+ (`program_options`, `atomic` and header-only libs)
- *Debian/Ubuntu:* `sudo apt-get install libboost-program-options-dev libboost-atomic-dev`
- *Arch Linux:* `sudo pacman --sync boost`
- *Spack:* `spack install boost +program_options +atomic`
- *from source:*
  - `mkdir -p ~/src ~/lib`
  - `cd ~/src`
  - `curl -Lo boost_1_74_0.tar.gz https://boostorg.jfrog.io/artifactory/main/release/1.74.0/source/boost_1_74_0.tar.gz`
  - `tar -xzf boost_1_74_0.tar.gz`
  - `cd boost_1_74_0`
  - `./bootstrap.sh --with-libraries=atomic,program_options --prefix=$HOME/lib/boost`
  - `./b2 cxxflags="-std=c++17" -j4 && ./b2 install`
- *environment:* (assumes install from source in `$HOME/lib/boost`)
  - `export CMAKE_PREFIX_PATH=$HOME/lib/boost:$CMAKE_PREFIX_PATH`

git

- not required for the code, but for our workflows
- 1.7.9.5 or higher
- *Debian/Ubuntu:* `sudo apt-get install git`
- *Arch Linux:* `sudo pacman --sync git`
- *Spack:* `spack install git`

rsync

- not required for the code, but for our workflows
- *Debian/Ubuntu:* `sudo apt-get install rsync`
- *Arch Linux:* `sudo pacman --sync rsync`
- *Spack:* `spack install rsync`

alpaka 1.1.X

- alpaka is included in the PIConGPU source code

mallocMC 2.6.0crp-dev

- only required for CUDA and HIP backends
- mallocMC is included in the PIConGPU source code

3.2.2 PIConGPU Source Code

- `git clone https://github.com/ComputationalRadiationPhysics/picongpu.git $HOME/src/picongpu`
  - *optional:* update the source code with `cd $HOME/src/picongpu && git fetch && git pull`
  - *optional:* change to a different branch with `git branch` (show) and `git checkout <BranchName>` (switch)
- *environment*:
  - `export PICSRC=$HOME/src/picongpu`
  - `export PIC_EXAMPLES=$PICSRC/share/picongpu/examples`
  - `export PATH=$PATH:$PICSRC`
  - `export PATH=$PATH:$PICSRC/bin`
  - `export PATH=$PATH:$PICSRC/src/tools/bin`
  - `export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH`

3.2.3 Optional Libraries

CUDA

- 11.0.0+
- required if you want to run on Nvidia GPUs
- *Debian/Ubuntu:* `sudo apt-get install nvidia-cuda-toolkit`
- *Arch Linux:* `sudo pacman --sync cuda`
- *Spack:* `spack install cuda`
- at least one **CUDA** capable **GPU**
- *compute capability*: `sm_60` or higher
- full list of CUDA GPUs and their *compute capability*
- More is always better. Especially, if we are talking GPUs :-)
- *environment:*
  - `export CUDA_ROOT=<CUDA_INSTALL>`

If you do not install the following libraries, you will not have the full amount of PIConGPU plugins. We recommend to install at least **pngwriter** and **openPMD**.

libpng

- 1.2.9+ (requires *zlib*)
- *Debian/Ubuntu dependencies:* `sudo apt-get install libpng-dev`
- *Arch Linux dependencies:* `sudo pacman --sync libpng`
- *Spack:* `spack install libpng`
- *from source:*
  - `mkdir -p ~/src ~/lib`
  - `cd ~/src`
  - `curl -Lo libpng-1.6.34.tar.gz ftp://ftp-osl.osuosl.org/pub/libpng/src/libpng16/libpng-1.6.34.tar.gz`
  - `tar -xf libpng-1.6.34.tar.gz`
  - `cd libpng-1.6.34`
  - `CPPFLAGS=-I$HOME/lib/zlib/include LDFLAGS=-L$HOME/lib/zlib/lib ./configure --enable-static --enable-shared --prefix=$HOME/lib/libpng`
  - `make`
  - `make install`
- *environment:* (assumes install from source in `$HOME/lib/libpng`)
  - `export PNG_ROOT=$HOME/lib/libpng`
  - `export CMAKE_PREFIX_PATH=$PNG_ROOT:$CMAKE_PREFIX_PATH`

pngwriter

- 0.7.0+ (requires *libpng*, *zlib*, and optional *freetype*)
- *Spack:* `spack install pngwriter`
- *from source:*
  - `mkdir -p ~/src ~/lib`
  - `git clone -b 0.7.0 https://github.com/pngwriter/pngwriter.git ~/src/pngwriter/`
  - `cd ~/src/pngwriter`
  - `mkdir build && cd build`
  - `cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/pngwriter ..`
  - `make install`
- *environment:* (assumes install from source in `$HOME/lib/pngwriter`)
  - `export CMAKE_PREFIX_PATH=$HOME/lib/pngwriter:$CMAKE_PREFIX_PATH`

openPMD API

- optional, but strongly recommended as most PIConGPU output requires it
- 0.15.0+
- *Spack*: `spack install openpmd-api`
- For usage in PIConGPU, the openPMD API must have been built either with support for ADIOS2 or HDF5 (or both). When building the openPMD API from source (described below), these dependencies must be built and installed first.
  - For ADIOS2, CMake build instructions can be found in the official documentation. Besides compression, the default configuration should generally be sufficient, the `CMAKE_INSTALL_PREFIX` should be set to a fitting location. Compression with `c-blosc` is described below.
  - For HDF5, CMake build instructions can be found in the official documentation. The parameters `-DHDF5_BUILD_CPP_LIB=OFF -DHDF5_ENABLE_PARALLEL=ON` are required, the `CMAKE_INSTALL_PREFIX` should be set to a fitting location.
- *from source:*
  - `mkdir -p ~/src ~/lib`
  - `git clone -b 0.15.0 https://github.com/openPMD/openPMD-api.git ~/src/openPMD-api`
  - `cd ~/src/openPMD-api`
  - `mkdir build && cd build`
  - `cmake .. -DopenPMD_USE_MPI=ON -DCMAKE_INSTALL_PREFIX=~/lib/openPMD-api` Optionally, specify the parameters `-DopenPMD_USE_ADIOS2=ON -DopenPMD_USE_HDF5=ON`. Otherwise, these parameters are set to `ON` automatically if CMake detects the dependencies on your system.
  - `make -j $(nproc) install`
- environment:* (assumes install from source in `$HOME/lib/openPMD-api`)
  - `export CMAKE_PREFIX_PATH="$HOME/lib/openPMD-api:$CMAKE_PREFIX_PATH"`
- If PIConGPU is built with openPMD output enabled, the JSON library nlohmann_json will automatically be used, found in the `thirdParty/` directory. By setting the CMake parameter `PIC_nlohmann_json_PROVIDER=extern`, CMake can be instructed to search for an installation of nlohmann_json externally. Refer to LICENSE.md for further information.

c-blosc for openPMD API with ADIOS2

- not a direct dependency of PIConGPU, but an optional dependency for openPMD API with ADIOS2; installation is described here since it is lacking in documentation elsewhere
- general purpose compressor, used in ADIOS2 for in situ data reduction
- *Debian/Ubuntu:* `sudo apt-get install libblosc-dev`
- *Arch Linux:* `sudo pacman --sync blosc`
- *Spack:* `spack install c-blosc`
- *from source:*
  - `mkdir -p ~/src ~/lib`
  - `git clone -b v1.21.1 https://github.com/Blosc/c-blosc.git ~/src/c-blosc/`
  - `cd ~/src/c-blosc`
  - `mkdir build && cd build`
  - `cmake -DCMAKE_INSTALL_PREFIX=$HOME/lib/c-blosc -DPREFER_EXTERNAL_ZLIB=ON ..`

    - `make install`

- *environment:* (assumes install from source in `$HOME/lib/c-blosc`)
    - `export BLOSC_ROOT=$HOME/lib/c-blosc`
    - `export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH`

ISAAC

- 1.6.0+
- requires *boost* (header only), *IceT*, *Jansson*, *libjpeg* (preferably *libjpeg-turbo*), *libwebsockets* (only for the ISAAC server, but not the plugin itself)
- enables live in situ visualization, see more here Plugin description
- *Spack:* `spack install isaac`
- *from source:* build the *in situ library* and its dependencies as described in ISAAC's INSTALL.md
- *environment:* set environment variable `CMAKE_PREFIX_PATH` for each dependency and the ISAAC in situ library

FFTW3

- required for Shadowgraphy plugin
- *from tarball:*
    - `mkdir -p ~/src ~/lib`
    - `cd ~/src`
    - `wget -O fftw-3.3.10.tar.gz http://fftw.org/fftw-3.3.10.tar.gz`
    - `tar -xf fftw-3.3.10.tar.gz`
    - `cd fftw-3.3.10`
    - `./configure --prefix="$FFTW_ROOT"`
    - `make`
    - `make install`
- *environment:* (assumes install from source in `$HOME/lib/fftw-3.3.10`)
    - ``export FFTW3_ROOT =$HOME/lib/fftw-3.3.10
    - `export LD_LIBRARY_PATH=$FFTW3_ROOT/lib:$LD_LIBRARY_PATH`

See also:

You need to have all *dependencies installed* to complete this chapter.

CHAPTER

FOUR

PICONGPU.PROFILE

Section author: Axel Huebl, Klaus Steiniger, Sergei Bastrakov

We recommend to use a `picongpu.profile` file, located directly in your `$HOME/` directory, to set up the environment within which PIConGPU will run by conviently performing

```
source $HOME/picongpu.profile
```

on the command line after logging in to a system. PIConGPU is shipped with a number of ready-to-use profiles for different systems which are located in `etc/picongpu/<cluster>-<institute>/` within PIConGPU's main folder. Have a look into this directory in order to see for which HPC systems profiles are already available. If you are working on one of these systems, just copy the respective `*_picongpu.profile.example` from within this directory into your `$HOME` and make the necessary changes, such as e-mail address or PIConGPU source code location defined by `$PICSRC`. If you are working on an HPC system for which no profile is available, feel free to create one and contribute it to PIConGPU by opening a pull request.

A selection of available profiles is presented below, after some general notes on using CPUs. Beware, these may not be up-to-date with the latest available software on the respective system, as we do not have continuous access to all of these.

4.1 General Notes on Using CPUs

On CPU systems we strongly recommend using MPI + OpenMP parallelization. It requires building PIConGPU with the OpenMP 2 backend. Additionally it is recommended to add an option for target architecture, for example, `pic-build -b omp2b:znver3` for AMD Zen3 CPUs. When building on a compute node or a same-architecture node, one could use `-b omp2b:native` instead. The default value for option `-b` can be set with environment variable `$PIC_BACKEND` in the profile.

With respect to selecting an optimal MPI + OpenMP configuration please refer to documentation of your system. As a reasonable default strategy, we recommend running an MPI rank per NUMA node, using 1 or 2 OpenMP threads per core depending on simultaneous multithreading being enabled, and binding threads to cores through affinity settings. This approach is used, for example, in the `defq` partition of Hemera as shown below.

The properties of OpenMP parallelization, such as number of threads used, are controlled via OpenMP environment variables. In particular, the number of OpenMP threads to be used (per MPI rank) can be set via `$OMP_NUM_THREADS`. Beware that task launch wrappers used on your system may effectively override this setting. Particularly, a few systems require running PIConGPU with `mpirun --bind-to none` in order to properly use all CPU cores.

For setting thread affinity, we provide a helper wrapper `cpuNumaStarter.sh` that should be applicable to most systems.

4.2 Your Workstation

This is a very basic `picongpu.profile` enabling compilation on CPUs by setting the OpenMP backend, declaring commonly required directories, and providing default parameters for *TBG*.

```bash
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# Self-Build Software #########################################################
# Optional, not required.
# Needs to be compiled by the user.
# Set environment variables required for compiling and linking PIConGPU here.
#export PIC_LIBS=$HOME/lib

# For example install openPMD-api yourself
#   https://picongpu.readthedocs.io/en/latest/install/dependencies.html#openpmd-api
#export OPENPMD_ROOT=$PIC_LIBS/openPMD-api
#export CMAKE_PREFIX_PATH="$OPENPMD_ROOT:$CMAKE_PREFIX_PATH"
#export LD_LIBRARY_PATH="$OPENPMD_ROOT/lib:$OPENPMD_ROOT/lib64:$LD_LIBRARY_PATH"

# For example install pngwriter yourself:
#   https://picongpu.readthedocs.io/en/latest/install/dependencies.html#pngwriter
#export PNGwriter_ROOT=$PIC_LIBS/pngwriter-0.7.0
#export CMAKE_PREFIX_PATH=$PNGwriter_ROOT:$CMAKE_PREFIX_PATH
#export LD_LIBRARY_PATH=$PNGwriter_ROOT/lib:$LD_LIBRARY_PATH

# Environment #################################################################
#
export PIC_BACKEND="omp2b:native" # running on cpu
# For more examples on possible backends, depending on your hardware, see
# https://picongpu.readthedocs.io/en/latest/usage/basics.html#pic-configure
# or the provided profiles of other systems.

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/bash"}

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
export TBG_SUBMIT="bash"
export TBG_TPLFILE="etc/picongpu/bash/mpirun.tpl"

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.3 Crusher (ORNL)

System overview: link

Production directory: usually `$PROJWORK/$proj/` (link). Note that `$HOME` is mounted on compute nodes as read-only.

For this profile to work, you need to download the *PIConGPU source code* and install *PNGwriter and openPMD* manually.

4.3.1 MI250X GPUs using hipcc (recommended)

```
printf "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n" >&2
printf "@ Do not forget to increase the GCD's reserved memory in  @\n" >&2
printf "@ memory.param by setting                                 @\n" >&2
printf "@   constexpr size_t reservedGpuMemorySize =              @\n" >&2
printf "@       uint64_t(2147483648); // 2 GiB                    @\n" >&2
printf "@ Further, set the initial buffer size in your ADIOS2     @\n" >&2
printf "@ configuration of your job's *.cfg file to 28GiB,        @\n" >&2
printf "@ and do not use more than this amount of memory per GCD  @\n" >&2
printf "@ in your setup, or you will see out-of-memory errors.    @\n" >&2
printf "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n" >&2

# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"


# Project Information ######################################## (edit this line)
#   - project for allocation and shared directories
export PROJID=<yourProject>

# Job control
# Allocate more nodes then required to execute the jobs to be able to handle broken␣
↪nodes
# Oversubscribe the nodes allocated by N per thousand required nodes.
export PIC_NODE_OVERSUBSCRIPTION_PT=2

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="vim"

# General modules #############################################################
#
# There are a lot of required modules already loaded when connecting
# such as mpi, libfabric and others.
# The following modules just add to these.

# Compiling with cray compiler wrapper CC
module load PrgEnv-cray/8.3.3
```
(continues on next page)

(continued from previous page)

```
module load craype-accel-amd-gfx90a
module load rocm/5.1.0

export MPICH_GPU_SUPPORT_ENABLED=1
module load cray-mpich/8.1.16

module load cmake/3.22.2
module load boost/1.78.0-cxx17

## set environment variables required for compiling and linking w/ hipcc
##   see (https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html#compiling-
↪with-hipcc)
export CXX=hipcc
export CXXFLAGS="$CXXFLAGS -I${MPICH_DIR}/include"
export LDFLAGS="$LDFLAGS -L${MPICH_DIR}/lib -lmpi -L${CRAY_MPICH_ROOTDIR}/gtl/lib -
↪lmpi_gtl_hsa"

# Other Software ##############################################################
#
module load zlib/1.2.11
module load git/2.35.1
module load c-blosc/1.21.1 adios2/2.7.1 hdf5/1.12.0 openpmd-api/0.15.2
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OLCF_HDF5_ROOT/lib
module load libpng/1.6.37 freetype/2.11.0

# Self-Build Software #########################################################
# Optional, not required.
#
# needs to be compiled by the user
export PIC_LIBS=$HOME/lib/crusher

# optionally install pngwriter yourself:
#   https://picongpu.readthedocs.io/en/0.6.0/install/dependencies.html#pngwriter
# But add option `-DCMAKE_CXX_COMPILER=$(which CC)` to `cmake` invocation
export PNGwriter_ROOT=$PIC_LIBS/pngwriter-0.7.0
export CMAKE_PREFIX_PATH=$PNGwriter_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$PNGwriter_ROOT/lib:$LD_LIBRARY_PATH

# Environment #################################################################
#
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="hip:gfx90a"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/crusher-ornl
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
```

(continues on next page)

(continued from previous page)

```
#   - "caar" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/crusher-ornl/batch.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    srun  --time=1:00:00 --nodes=$numNodes --ntasks=$((numNodes * 8)) --cpus-per-
↪task=8 --ntasks-per-gpu=1 --gpu-bind=closest --mem-per-gpu=64000 -p batch -A
↪$PROJID --pty bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numGPUs=1
    else
        if [ "$1" -gt 8 ] ; then
            echo "The maximal number of devices per node is 8." 1>&2
            return 1
        else
            numGPUs=$1
        fi
    fi
    srun  --time=1:00:00 --nodes=1 --ntasks=$(($numGPUs)) --cpus-per-task=8 --ntasks-
↪per-gpu=1 --gpu-bind=closest --mem-per-gpu=64000 -p batch -A $PROJID --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.3.2 MI250X GPUs using craycc

```
printf "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n" >&2
printf "@ Do not forget to increase the GCD's reserved memory in  @\n" >&2
printf "@ memory.param by setting                                  @\n" >&2
printf "@   constexpr size_t reservedGpuMemorySize =               @\n" >&2
printf "@       uint64_t(2147483648); // 2 GiB                     @\n" >&2
printf "@ Further, set the initial buffer size in your ADIOS2     @\n" >&2
printf "@ configuration of your job's *.cfg file to 28GiB,        @\n" >&2
printf "@ and do not use more than this amount of memory per GCD  @\n" >&2
printf "@ in your setup, or you will see out-of-memory errors.    @\n" >&2
printf "@                                                          @\n" >&2
printf "@ WARNING: Using CRAY CC is not recommended!               @\n" >&2
```

(continues on next page)

(continued from previous page)

```
printf "@          Tests showed hipcc is two times faster.        @\n" >&2
printf "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n" >&2

# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"


# Project Information ######################################## (edit this line)
#   - project for allocation and shared directories
export PROJID=<yourProject>

# Job control
# Allocate more nodes then required to execute the jobs to be able to handle broken␣
↪nodes
# Oversubscribe the nodes allocated by N per thousand required nodes.
export PIC_NODE_OVERSUBSCRIPTION_PT=2

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="vim"

# General modules #############################################################
#
# There are a lot of required modules already loaded when connecting
# such as mpi, libfabric and others.
# The following modules just add to these.

# Compiling with cray compiler wrapper CC
module load PrgEnv-cray/8.3.3
module load craype-accel-amd-gfx90a
module load rocm/5.1.0

export MPICH_GPU_SUPPORT_ENABLED=1
module load cray-mpich/8.1.16
PE_MPICH_GTL_DIR_amd_gfx90a="-L${CRAY_MPICH_ROOTDIR}/gtl/lib"
PE_MPICH_GTL_LIBS_amd_gfx90a="-lmpi_gtl_hsa"
export CXXFLAGS="$CXXFLAGS -I${ROCM_PATH}/include"
export LDFLAGS="$LDFLAGS -L${ROCM_PATH}/lib -lamdhip64"

module load cmake/3.22.2
module load boost/1.78.0-cxx17

# Other Software ##############################################################
#
module load zlib/1.2.11
module load git/2.35.1
module load c-blosc/1.21.1 adios2/2.7.1 hdf5/1.12.0 openpmd-api/0.15.2
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OLCF_HDF5_ROOT/lib
```

(continues on next page)

(continued from previous page)

```
module load libpng/1.6.37 freetype/2.11.0

# Self-Build Software ##########################################################
# Optional, not required.
#
# needs to be compiled by the user
export PIC_LIBS=$HOME/lib/crusher

# optionally install pngwriter yourself:
#   https://picongpu.readthedocs.io/en/0.6.0/install/dependencies.html#pngwriter
# But add option `-DCMAKE_CXX_COMPILER=$(which CC)` to `cmake` invocation
export PNGwriter_ROOT=$PIC_LIBS/pngwriter-0.7.0
export CMAKE_PREFIX_PATH=$PNGwriter_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$PNGwriter_ROOT/lib:$LD_LIBRARY_PATH

# Environment ##################################################################
#
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="hip:gfx90a"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/crusher-ornl
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ########################################################
#   - SLURM (sbatch)
#   - "caar" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/crusher-ornl/batch.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    srun  --time=1:00:00 --nodes=$numNodes --ntasks=$((numNodes * 8)) --cpus-per-
↪task=8 --ntasks-per-gpu=1 --gpu-bind=closest --mem-per-gpu=64000 -p batch -A
↪$PROJID --pty bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numGPUs=1
    else
```

(continues on next page)

(continued from previous page)

```
        if [ "$1" -gt 8 ] ; then
            echo "The maximal number of devices per node is 8." 1>&2
            return 1
        else
            numGPUs=$1
        fi
    fi
    srun  --time=1:00:00 --nodes=1 --ntasks=$(($numGPUs)) --cpus-per-task=8 --ntasks-
↪per-gpu=1 --gpu-bind=closest --mem-per-gpu=64000 -p batch -A $PROJID --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.4 Hemera (HZDR)

System overview: link (internal)

User guide: *None*

Production directory: `/bigdata/hplsim/` with `external/`, `scratch/`, `development/` and `production/`

Profile for HZDR's home cluster hemera. Sets up software environment, i.e. providing libraries to satisfy PIConGPU's dependencies, by loading modules, setting common paths and options, as well as defining the `getDevice()` and `getNode()` aliases. The latter are shorthands to request resources for an interactive session from the batch system. Together with the *-s bash* option of *TBG*, these allow to run PIConGPU interactively on an HPC system.

For this profile to work, you need to download the *PIConGPU source code* manually.

4.4.1 Queue: defq (2x Intel Xeon Gold 6148, 20 Cores + 20 HyperThreads/CPU)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #############################################################
#
module purge
module load git
```

(continues on next page)

(continued from previous page)

```
module load gcc/12.2.0
module load ucx/1.14.0
module load libfabric/1.17.0-co79-csk
module load openmpi/4.1.5-ucx
module load python/3.10.4
module load boost/1.82.0

# Other Software ##############################################################
#
module load zlib/1.2.11
module load hdf5-parallel/1.14.0-omp415
module load c-blosc/1.21.4
module load adios2/2.9.2-csk
module load openpmd/0.15.2-csk
module load cmake/3.26.1

module load libpng/1.6.39
module load pngwriter/0.7.0

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:skylake-avx512"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/hemera-hzdr
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "defq" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/defq.tpl"
# use defq for regular queue and defq_low for low priority queue
export TBG_partition="defq"

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    srun --time=1:00:00 --nodes=$numNodes --ntasks-per-node=2 --cpus-per-task=20  --
↪mem=360000 -p defq --pty bash
}
```

(continues on next page)

(continued from previous page)

```bash

# allocate an interactive shell for one hour
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 2 ] ; then
            echo "The maximal number of devices per node is 2." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    srun --time=1:00:00 --ntasks-per-node=$(($numDevices)) --cpus-per-task=$((20 *
↪$numDevices)) --mem=$((180000 * numDevices)) -p defq --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.4.2 Queue: gpu (4x NVIDIA P100 16GB)

```bash
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #############################################################
#
module purge
module load git
module load gcc/12.2.0
module load cuda/12.1
module load libfabric/1.17.0
module load ucx/1.14.0-gdr
module load openmpi/4.1.5-cuda121-gdr
module load python/3.10.4
module load boost/1.82.0

```

(continues on next page)

(continued from previous page)

```
# Other Software ##############################################################
#
module load zlib/1.2.11
module load hdf5-parallel/1.12.0-omp415-cuda121
module load c-blosc/1.21.4
module load adios2/2.9.2-cuda121
module load openpmd/0.15.2-cuda121
module load cmake/3.26.1
module load fftw/3.3.10-ompi415-cuda121-gdr

module load libpng/1.6.39
module load pngwriter/0.7.0

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:60"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/hemera-hzdr
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "gpu" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/gpu.tpl"
# use gpu for regular queue and gpu_low for low priority queue
export TBG_partition="gpu"

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    srun  --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --cpus-per-task=6 --
↪gres=gpu:4 --mem=378000 -p gpu --pty bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numGPUs=1
```

(continues on next page)

(continued from previous page)

```
    else
        if [ "$1" -gt 4 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numGPUs=$1
        fi
    fi
    srun  --time=1:00:00 --ntasks-per-node=$(($numGPUs)) --cpus-per-task=6 --gres=gpu:
↪$numGPUs --mem=$((94500 * numGPUs)) -p gpu --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.4.3 Queue: fwkt_v100 (4x NVIDIA V100 32GB)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #############################################################
#
module purge
module load git
module load gcc/12.2.0
module load cuda/12.1
module load libfabric/1.17.0
module load ucx/1.14.0-gdr
module load openmpi/4.1.5-cuda121-gdr
module load python/3.10.4
module load boost/1.82.0

# Other Software ##############################################################
#
module load zlib/1.2.11
module load hdf5-parallel/1.12.0-omp415-cuda121
module load c-blosc/1.21.4
module load adios2/2.9.2-cuda121
```

(continues on next page)

(continued from previous page)

```
module load openpmd/0.15.2-cuda121
module load cmake/3.26.1
module load fftw/3.3.10-ompi415-cuda121-gdr

module load libpng/1.6.39
module load pngwriter/0.7.0

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/hemera-hzdr
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "fwkt_v100" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/hemera-hzdr/fwkt_v100.tpl"
# use fwkt_v100 for regular queue and casus_low for low priority queue
export TBG_partition="fwkt_v100"

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    srun  --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --cpus-per-task=6 --
↪gres=gpu:4 --mem=378000 -p $TBG_partition -A $TBG_partition --pty bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numGPUs=1
    else
        if [ "$1" -gt 4 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numGPUs=$1
```

(continues on next page)

(continued from previous page)

```
        fi
    fi
    srun  --time=1:00:00 --ntasks-per-node=$(($numGPUs)) --cpus-per-task=6 --gres=gpu:
↪$numGPUs --mem=$((94500 * numGPUs)) -p $TBG_partition -A $TBG_partition --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.5 Summit (ORNL)

System overview: link

User guide: link

Production directory: usually `$PROJWORK/$proj/` (link). Note that `$HOME` is mounted on compute nodes as read-only.

For this profile to work, you need to download the *PIConGPU source code* and install *PNGwriter* manually.

4.5.1 V100 GPUs (recommended)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on job (-B)egin, Fi(-N)ish
export MY_MAILNOTIFY=""
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj=<yourProject>

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#module load nano
#export EDITOR="nano"

# basic environment ################################################
module unload xl
module load gcc/12.1.0
module load spectrum-mpi/10.4.0.6-20230210

export CC=$(which gcc)
export CXX=$(which g++)
```

(continues on next page)

(continued from previous page)

```
# required tools and libs
module load git/2.42.0
module load cmake/3.27.7
module load cuda/12.2.0
# boost needs to be installed manually (or uncomment if you have access to csc380)
#export CMAKE_PREFIX_PATH=/gpfs/alpine2/csc380/proj-shared/lib_summit/lib/boost:
↪$CMAKE_PREFIX_PATH

# plugins (optional) ##########################################################
module load hdf5/1.14.3
module load c-blosc/1.21.5 zfp/1.0.0-cuda117 sz/2.1.12.5 lz4/1.9.4
module load adios2/2.9.2

module load openpmd-api/0.15.2

#export T3PIO_ROOT=$PROJWORK/$proj/lib/t3pio
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$T3PIO_ROOT/lib

module load zlib-ng/2.1.4
module load libpng/1.6.39 freetype/2.11.1
# optionally install pngwriter yourself:
#   https://github.com/pngwriter/pngwriter#install
# export PNGwriter_ROOT=<your pngwriter install directory>  # e.g., ${HOME}/sw/
↪pngwriter
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PNGwriter_ROOT/lib

# helper variables and tools ##################################################
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/summit-ornl
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# fix MPI collectives by disabling IBM's optimized barriers
# https://github.com/ComputationalRadiationPhysics/picongpu/issues/3814
export OMPI_MCA_coll_ibm_skip_barrier=true

alias getNode="bsub -P $proj -W 2:00 -nnodes 1 -Is /bin/bash"

# "tbg" default options #######################################################
export TBG_SUBMIT="bsub"
export TBG_TPLFILE="etc/picongpu/summit-ornl/gpu_batch.tpl"

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
```

(continues on next page)

(continued from previous page)

```
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.6 Piz Daint (CSCS)

System overview: link

User guide: link

Production directory: `$SCRATCH` (link).

For this profile to work, you need to download the *PIConGPU source code* and install *boost, libpng, PNGwriter and ADIOS2* manually.

Note: The MPI libraries are lacking Fortran bindings (which we do not need anyway). During the install of ADIOS, make sure to add to `configure` the `--disable-fortran` flag.

Note: Please find a Piz Daint quick start from August 2018 here.

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit those lines)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
# module load nano
#export EDITOR="nano"

# Programming Environment #####################################################
#
# if the wrong environment is loaded we switch to the gnu environment
# note: this loads gcc/5.3.0 (6.0.4 is the version of the programming env!)
CRAYENV_FOUND=$(module li 2>&1 | grep "PrgEnv-cray" > /dev/null && { echo 0; } || {␣
↪echo 1; })
if [ $CRAYENV_FOUND -eq 0 ]; then
    module swap PrgEnv-cray PrgEnv-gnu/6.0.4
else
    module load PrgEnv-gnu/6.0.4
fi

module load daint-gpu
# currently loads CUDA 8.0
module load craype-accel-nvidia60

# Compile for cluster nodes
#   (CMake likes to unwrap the Cray wrappers)
```

(continues on next page)

(continued from previous page)

```
export CC=$(which cc)
export CXX=$(which CC)

# define cray compiler target architecture
# if not defined the linker crashed because wrong from */lib instead
# of */lib64 are used
export CRAY_CPU_TARGET=x86-64

# Libraries ###################################################################
module load CMake

module load cray-mpich/7.6.0
module load cray-hdf5-parallel/1.10.0.3

# Self-Build Software #########################################################
#
# needs to be compiled by the user
export PIC_LIBS="$HOME/lib"
export BOOST_ROOT=$PIC_LIBS/boost-1.66.0
export ZLIB_ROOT=$PIC_LIBS/zlib-1.2.11
export PNG_ROOT=$PIC_LIBS/libpng-1.6.34
export BLOSC_ROOT=$PIC_LIBS/blosc-1.12.1
export PNGwriter_DIR=$PIC_LIBS/pngwriter-0.7.0

export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$ZLIB_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$PNG_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$PNGwriter_DIR/lib:$LD_LIBRARY_PATH

export PATH=$PNG_ROOT/bin:$PATH

export CMAKE_PREFIX_PATH=$ZLIB_ROOT:$CMAKE_PREFIX_PATH
export CMAKE_PREFIX_PATH=$PNG_ROOT:$CMAKE_PREFIX_PATH

export MPI_ROOT=$MPICH_DIR
export HDF5_ROOT=$HDF5_DIR

# Environment #################################################################
#
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:60"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/pizdaint-
→cscs"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
```

(continues on next page)

(continued from previous page)

```
#   - "normal" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/pizdaint-cscs/normal.tpl"

# helper tools ################################################################

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    # --ntasks-per-core=2  # activates intel hyper threading
    salloc --time=1:00:00 --nodes="$numNodes" --ntasks-per-node=12 --ntasks-per-
↪core=2 --partition normal --gres=gpu:1 --constraint=gpu
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.7 Taurus (TU Dresden)

System overview: link

User guide: link

Production directory: /scratch/$USER/ and /scratch/$proj/

For these profiles to work, you need to download the *PIConGPU source code* and install *PNGwriter* manually.

4.7.1 Queue: gpu2 (Nvidia K80 GPUs)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj=$(groups | awk '{print $1}')
```

(continues on next page)

(continued from previous page)

```
# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Modules ######################################################################
#
module purge
module load modenv/scs5

module load CUDA/11.0.2-GCC-9.3.0
module load OpenMPI/4.0.4-GCC-9.3.0
module load CMake
module load libpng/1.6.37-GCCcore-9.3.0
module load git/2.23.0-GCCcore-9.3.0-nodocs
module load Python/3.8.2-GCCcore-9.3.0

# Self-Build Software ##########################################################
#
# needs to be compiled by the user
# Check the install script at
# https://gist.github.com/finnolec/3004deeb6251d1e76f2250f99a8a2170
#
# path to own libraries:
export PIC_LIBS=$HOME/lib

export BLOSC_ROOT=$PIC_LIBS/blosc-1.21.1
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

export PNGwriter_ROOT=$PIC_LIBS/pngwriter-0.7.0
export CMAKE_PREFIX_PATH=$PNGwriter_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$PNGwriter_ROOT/lib:$LD_LIBRARY_PATH

export ADIOS2_ROOT=$PIC_LIBS/adios2-2.7.1
export LD_LIBRARY_PATH=$ADIOS2_ROOT/lib64:$LD_LIBRARY_PATH

export OPENPMD_ROOT=$PIC_LIBS/openpmd-0.14.3
export LD_LIBRARY_PATH=$OPENPMD_ROOT/lib64:$LD_LIBRARY_PATH

export HDF5_ROOT=$PIC_LIBS/hdf5-1.12.1
export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH

export BOOST_ROOT=$PIC_LIBS/BOOST_1_74_0
export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH

# Environment ##################################################################

export PROJECT=/projects/$proj

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:37"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/taurus-tud"}
```

(continues on next page)

(continued from previous page)

```
export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "gpu2" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/taurus-tud/k80.tpl"

alias getDevice='srun -p gpu2-interactive --gres=gpu:1 --ntasks=1 --pty --mem=8G -t␣
↪1:00:00 bash'
alias getNode='srun -p gpu2-interactive --gres=gpu:4 -n 1 --pty --mem=0 -t 2:00:00␣
↪bash'

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.7.2 Queue: ml (NVIDIA V100 GPUs on Power9 nodes)

For this profile, you additionally need to compile and install everything for the power9-architecture including your own *boost*, *HDF5*, c-blosc and *ADIOS*.

Note: Please find a Taurus ml quick start here.

Note: You need to compile the libraries and PIConGPU on an `ml` node since only nodes in the `ml` queue are Power9 systems.

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Modules #####################################################################
#
module switch modenv/ml
```

(continues on next page)

(continued from previous page)

```
# load CUDA/9.2.88-GCC-7.3.0-2.30, also loads GCC/7.3.0-2.30, zlib, OpenMPI and others
module load fosscuda/2018b
module load CMake
module load libpng/1.6.34-GCCcore-7.3.0

printf "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n"
printf "@ Note: You need to compile picongpu on a node. @\n"
printf "@       Likewise for building the libraries.    @\n"
printf "@       Get a node with the getNode command.    @\n"
printf "@       Then source %s again.@\n" "$(basename $PIC_PROFILE)"
printf "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n"

# Self-Build Software #########################################################
#
# needs to be compiled by the user
# Check the install script at
# https://gist.github.com/steindev/cc02eae81f465833afa27fc8880f3473#file-picongpu_0-4-
↪3_taurus-tud-sh
#
export PIC_LIBS=$HOME/lib/power9
export BOOST_ROOT=$PIC_LIBS/boost-1.68.0-Power9
export PNGwriter_DIR=$PIC_LIBS/pngwriter-0.7.0-Power9
export HDF5_ROOT=$PIC_LIBS/hdf5-1.8.20-Power9
export BLOSC_ROOT=$PIC_LIBS/blosc-1.16.2-Power9

export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$PNGwriter_DIR/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH
export CMAKE_PREFIX_PATH=$HDF5_ROOT:$CMAKE_PREFIX_PATH

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/taurus-tud"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

# python not included yet
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# This is necessary in order to make alpaka compile.
# The workaround is from Axel Huebl according to alpaka PR #702.
export CXXFLAGS="-Dlinux"

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "ml" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/taurus-tud/V100.tpl"
```

(continues on next page)

(continued from previous page)

```
# allocate an interactive shell for two hours
#   getNode 2  # allocates 2 interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    export OMP_NUM_THREADS=7
    srun --time=2:00:00 --nodes=$numNodes --ntasks=$((6 * $numNodes)) --ntasks-per-
↪node=6 --cpus-per-task=7 --mem=0 --exclusive --gres=gpu:6 -p ml --pty bash
}

# allocate an interactive shell for two hours
#   getDevice 2  # allocates 2 interactive devices on one node (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 6 ] ; then
            echo "The maximal number of devices per node is 6." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    export OMP_NUM_THREADS=7
    srun --time=2:00:00 --nodes=1 --ntasks=$numDevices --ntasks-per-node=$((
↪$numDevices)) --cpus-per-task=7 --mem=$((254000 / $numDevices)) --gres=gpu:
↪$numDevices -p ml --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.8 Cori (NERSC)

System overview: link

User guide: link

Production directory: `$SCRATCH` (link).

For these profiles to work, you need to download the *PIConGPU source code* and install *PNGwriter* manually.

4.8.1 Queue: dgx (DGX - A100)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj="<yourProject>"
# Project reservation (can be empty if no reservation exists)
export RESERVATION=""

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #############################################################
#
module purge
module load dgx
module swap PrgEnv-intel PrgEnv-gnu
module load cuda openmpi
module load cmake
module load boost/1.70.0

# Other Software ##############################################################
#
module load png/1.6.34
module load zlib/1.2.11

export ADIOS2_ROOT=/global/cfs/cdirs/ntrain/dgx/openmpi/adios-2.7.1
export HDF5_ROOT=/global/cfs/cdirs/ntrain/dgx/openmpi/hdf5-1.10.7
export openPMD_ROOT=/global/cfs/cdirs/ntrain/dgx/openmpi/openPMD-api-0.13.4
export PNGwriter_ROOT=/global/cfs/cdirs/ntrain/dgx/pngwriter-0.7.0-25-g0c30be5

# Environment #################################################################
#
export CC="$(which gcc)"
export CXX="$(which g++)"
export CUDACXX=$(which nvcc)
export CUDAHOSTCXX=$(which g++)

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:80"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/cori-nersc"}
```

(continues on next page)

(continued from previous page)

```
export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export LD_LIBRARY_PATH=$ADIOS2_ROOT/lib64:$HDF5_ROOT/lib:$openPMD_ROOT/lib64:
↪$PNGwriter_ROOT/lib64:$LD_LIBRARY_PATH
export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "defq" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/cori-nersc/a100.tpl"

if [ -z "$RESERVATION" ] ; then
  SLURM_RESERVATION=""
else
  SLURM_RESERVATION="--reservation=$RESERVATION"
fi

# allocate an interactive node for one hour to execute a mpi parallel application
#   getNode 2  # allocates two interactive A100 GPUs (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    echo "Hint: please use 'srun --cpu_bind=cores <COMMAND>' for launching multiple␣
↪processes in the interactive mode."
    echo "             use 'srun -n 1 --pty bash' for launching a interactive shell␣
↪for compiling."
    salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node 8 --gpus 8 --cpus-per-
↪task=16 -A $proj -C dgx -q shared $SLURM_RESERVATION
}

# allocate an interactive device for one hour to execute a mpi parallel application
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numGPUs=1
    else
        if [ "$1" -gt 8 ] ; then
            echo "The maximal number of devices per node is 8." 1>&2
            return 1
        else
            numGPUs=$1
        fi
    fi
    echo "Hint: please use 'srun --cpu_bind=cores <COMMAND>' for launching multiple␣
↪processes in the interactive mode."
    echo "             use 'srun -n 1 --pty bash' for launching a interactive shell␣
↪for compiling."
    salloc --time=1:00:00 --ntasks-per-node=$numGPUs --cpus-per-task=16 --gpus=
↪$numGPUs --mem=$((1010000 / 8) * $numGPUs) -A $proj -C dgx -q shared $SLURM_
↪RESERVATION
}
```

(continues on next page)

(continued from previous page)

```
# allocate an interactive shell for compilation (without gpus)
function getShell() {
    srun --time=1:00:00 --nodes=1 --ntasks 1 --cpus-per-task=16 -A $proj -C dgx -q␣
↪shared $SLURM_RESERVATION --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.9 Draco (MPCDF)

System overview: link

User guide: link

Production directory: `/ptmp/$USER/`

For this profile to work, you need to download the *PIConGPU source code* and install *libpng and PNGwriter* manually.

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General Modules #############################################################
#
module purge

module load git/2.14
module load gcc
module load cmake
module load boost/gcc/1.64
module load impi/2017.3
module load hdf5-mpi/gcc/1.8.18

# Other Software ##############################################################
#
# needs to be compiled by the user
export PNGWRITER_ROOT=$HOME/lib/pngwriter-0.7.0
```

(continues on next page)

(continued from previous page)

```
export LD_LIBRARY_PATH=$PNGWRITER_ROOT/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$BOOST_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HDF5_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$I_MPI_ROOT/lib64:$LD_LIBRARY_PATH

export HDF5_ROOT=$HDF5_HOME

export CXX=$(which g++)
export CC=$(which gcc)

# PIConGPU Helper Variables ###################################################
#
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:haswell"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/draco-mpcdf
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "normal" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/draco-mpcdf/general.tpl"

# helper tools ################################################################

# allocate an interactive shell for one hour
alias getNode='salloc --time=1:00:00 --nodes=1 --exclusive --ntasks-per-node=2 --cpus-
↪per-task=32 --partition general'

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.10 D.A.V.I.D.E (CINECA)

System overview: link

User guide: link

Production directory: `$CINECA_SCRATCH/` (link)

For this profile to work, you need to download the *PIConGPU source code* manually.

4.10.1 Queue: dvd_usr_prod (Nvidia P100 GPUs)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj=$(groups | awk '{print $2}')

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #############################################################
#
module purge
module load gnu/6.4.0
module load cmake
module load cuda/9.2.88
module load openmpi/3.1.0--gnu--6.4.0
module load boost/1.68.0--openmpi--3.1.0--gnu--6.4.0

export CMAKE_PREFIX_PATH=$CUDA_HOME:$OPENMPI_HOME:$CMAKE_PREFIX_PATH
export CMAKE_PREFIX_PATH=$BOOST_HOME:$CMAKE_PREFIX_PATH

# Other Software ##############################################################
#
module load zlib/1.2.11--gnu--6.4.0
module load szip/2.1.1--gnu--6.4.0
module load blosc/1.12.1--gnu--6.4.0

module load hdf5/1.10.4--openmpi--3.1.0--gnu--6.4.0

module load libpng/1.6.35--gnu--6.4.0
module load freetype/2.9.1--gnu--6.4.0
module load pngwriter/0.7.0--gnu--6.4.0

export CMAKE_PREFIX_PATH=$ZLIB_HOME:$SZIP_HOME:$BLOSC_HOME:$CMAKE_PREFIX_PATH
export CMAKE_PREFIX_PATH=$LIBPNG_HOME:$FREETYPE_HOME:$PNGWRITER_HOME:$CMAKE_PREFIX_
```

(continues on next page)

(continued from previous page)

```
↪PATH

# Work-Arounds ###############################################################
#
# fix for Nvidia NVCC bug id 2448610
# see https://github.com/ComputationalRadiationPhysics/alpaka/issues/701
export CXXFLAGS="-Dlinux"

# Environment ################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:60"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/davide-
↪cineca"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options ######################################################
#   - SLURM (sbatch)
#   - "gpu" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/davide-cineca/gpu.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    srun --time=0:30:00 --nodes=$numNodes --ntasks-per-socket=8 --ntasks-per-node=16 -
↪-mem=252000 --gres=gpu:4 -A $proj -p dvd_usr_prod --pty bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numGPUs=1
    else
        if [ "$1" -gt 4 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numGPUs=$1
        fi
```

(continues on next page)

(continued from previous page)

```
    fi
    srun  --time=1:00:00 --ntasks-per-node=$numGPUs --cpus-per-task=$((4 * $numGPUs))␣
↪--gres=gpu:$numGPUs --mem=$((63000 * numGPUs)) -A $proj -p dvd_usr_prod --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.11 JURECA (JSC)

System overview: link

User guide: link

Production directory: `$SCRATCH` (link)

For these profiles to work, you need to download the *PIConGPU source code* and install *PNGwriter and openPMD*, for the gpus partition also *Boost and HDF5*, manually.

4.11.1 Queue: batch (2 x Intel Xeon E5-2680 v3 CPUs, 12 Cores + 12 Hyperthreads/CPU)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj=$(groups | awk '{print $5}')

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
jutil env activate -p $proj

# General modules #############################################################
#
module purge
module load Intel/2019.0.117-GCC-7.3.0
module load CMake
```

(continues on next page)

(continued from previous page)

```
module load IntelMPI/2018.4.274
module load Python/3.6.6
module load Boost/1.68.0-Python-3.6.6

# Other Software ##############################################################
#
module load zlib/.1.2.11
module load HDF5/1.10.1
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH

PARTITION_LIB=$PROJECT/lib_batch
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:haswell"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/jureca-jsc"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export CC=$(which icc)
export CXX=$(which icpc)

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "batch" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/jureca-jsc/batch.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates 2 interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    if [ $numNodes -gt 8 ] ; then
        echo "The maximal number of interactive nodes is 8." 1>&2
        return 1
```

(continues on next page)

(continued from previous page)

```
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    export OMP_NUM_THREADS=24
    salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=2 --mem=126000 -A $proj␣
↪-p devel bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates 2 interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 2 ] ; then
            echo "The maximal number of devices per node is 2." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    export OMP_NUM_THREADS=24
    salloc --time=1:00:00 --ntasks-per-node=$(($numDevices)) --mem=126000 -A $proj -p␣
↪devel bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.11.2 Queue: gpus (2 x Nvidia Tesla K80 GPUs)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj=$(groups | awk '{print $5}')

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
```

(continues on next page)

(continued from previous page)

```
#export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
jutil env activate -p $proj

# General modules #############################################################
#
module purge
module load GCC/7.3.0
module load CUDA/9.2.88
module load CMake
module load MVAPICH2/2.3-GDR
module load Python/3.6.6

# Other Software ##############################################################
#
module load zlib/.1.2.11
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH

PARTITION_LIB=$PROJECT/lib_gpus
BOOST_ROOT=$PARTITION_LIB/boost
export CMAKE_PREFIX_PATH=$BOOST_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BOOST_ROOT/lib:$LD_LIBRARY_PATH

HDF5_ROOT=$PARTITION_LIB/hdf5
export PATH=$HDF5_ROOT/bin:$PATH
export CMAKE_PREFIX_PATH=$HDF5_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$HDF5_ROOT/lib:$LD_LIBRARY_PATH

PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:37" # Nvidia K80 architecture

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/jureca-jsc"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
```

(continues on next page)

(continued from previous page)

```
#   - "gpus" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/jureca-jsc/gpus.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates 2 interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    if [ $numNodes -gt 8 ] ; then
        echo "The maximal number of interactive nodes is 8." 1>&2
        return 1
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --gres=gpu:4 --
↪mem=126000 -A $proj -p develgpus bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates 2 interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 4 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    salloc --time=1:00:00 --ntasks-per-node=$(($numDevices)) --gres=gpu:4 --
↪mem=126000 -A $proj -p develgpus bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.11.3 Queue: booster (Intel Xeon Phi 7250-F, 68 cores + Hyperthreads)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj=$(groups | awk '{print $5}')

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
jutil env activate -p $proj

# General modules #############################################################
#
module purge
module load Architecture/KNL
module load Intel/2019.0.117-GCC-7.3.0
module load CMake
module load IntelMPI/2018.4.274
module load Python/3.6.6
module load Boost/1.68.0-Python-3.6.6

# Other Software ##############################################################
#
module load zlib/.1.2.11
module load HDF5/1.10.1
module load libpng/.1.6.35
export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH

PARTITION_LIB=$PROJECT/lib_booster
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:MIC-AVX512"

```

(continues on next page)

(continued from previous page)

```
# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/jureca-jsc"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export CC=$(which icc)
export CXX=$(which icpc)

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #####################################################
#   - SLURM (sbatch)
#   - "booster" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/jureca-jsc/booster.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates 2 interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    if [ $numNodes -gt 8 ] ; then
        echo "The maximal number of interactive nodes is 8." 1>&2
        return 1
    fi
    export OMP_NUM_THREADS=34
    salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --mem=94000 -A $proj -
↪p develbooster bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates 2 interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 1 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    export OMP_NUM_THREADS=34
    salloc --time=1:00:00 --ntasks-per-node=$(($numDevices)) --mem=94000 -A $proj -p␣
↪develbooster bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
```

(continues on next page)

(continued from previous page)

```
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.12 JUWELS (JSC)

System overview: link

User guide: link

Production directory: `$SCRATCH` (link)

For these profiles to work, you need to download the *PIConGPU source code* and install *PNGwriter and openPMD*, for the gpus partition also *Boost and HDF5*, manually.

4.12.1 Queue: batch (2 x Intel Xeon Platinum 8168 CPUs, 24 Cores + 24 Hyperthreads/CPU)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project and account for allocation
#
#   `jutil user projects` will return a table of project associations.
#   Each row contains: project,unixgroup,PI-uid,project-type,budget-accounts
#   We need the first and last entry.
#   Here: select the last available project.
#   Alternative: Set proj, account manually
export proj=$( jutil user projects --noheader | awk '{print $1}' | tail -n 1 )
export account=$(jutil user projects -n | awk '{print $NF}' | tail -n 1)
# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"
# Set up environment, including $SCRATCH and $PROJECT
# Handle a case where the budgeting account is not set.
if [ $accountt = "-" ]; then
    jutil env activate --project $proj;
else
    jutil env activate --project $proj --budget $account
fi


# General modules #############################################################
#
```

(continues on next page)

(continued from previous page)

```
module purge
module load Intel/2020.2.254-GCC-9.3.0
module load CMake
module load IntelMPI/2019.8.254
module load Python/3.8.5

module load Boost/1.73.0

# Other Software ##############################################################
#
module load HDF5/1.10.6
#export CMAKE_PREFIX_PATH=$EBROOTZLIB:$EBROOTLIBPNG:$CMAKE_PREFIX_PATH

PARTITION_LIB=$PROJECT/lib_batch
PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH

# Environment #################################################################
#
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BOOST_LIB

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="omp2b:skylake"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/juwels-jsc"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export CC=$(which icc)
export CXX=$(which icpc)

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "batch" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/juwels-jsc/batch.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates 2 interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    if [ $numNodes -gt 8 ] ; then
```

(continues on next page)

(continued from previous page)

```
        echo "The maximal number of interactive nodes is 8." 1>&2
        return 1
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    export OMP_NUM_THREADS=48
    salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=2 --mem=94000 -A
↪$account -p batch bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates 2 interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 2 ] ; then
            echo "The maximal number of devices per node is 2." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    export OMP_NUM_THREADS=48
    salloc --time=1:00:00 --ntasks-per-node=$(($numDevices)) --mem=94000 -A $account -
↪p batch bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.12.2 Queue: gpus (4 x Nvidia V100 GPUs)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project and account for allocation
#   jutil user projects will return a table of project associations.
#   Each row contains: project,unixgroup,PI-uid,project-type,budget-accounts
```

(continues on next page)

(continued from previous page)

```
#   We need the first and last entry.
#   Here: select the last available project.
export proj=$( jutil user projects --noheader | awk '{print $1}' | tail -n 1 )
export account=$(jutil user projects -n | awk '{print $NF}' | tail -n 1)

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# Set up environment, including $SCRATCH and $PROJECT
# Handle a case where the budgeting account is not set.
if [ "$account" = "-" ]; then
    jutil env activate --project $proj;
else
    jutil env activate --project $proj --budget $account
fi

# General modules #############################################################
#
module purge
module load GCC/9.3.0
module load CUDA/11.0
module load CMake
module load ParaStationMPI/5.4.7-1
module load mpi-settings/CUDA
module load Python/3.8.5

module load Boost/1.74.0
module load HDF5/1.10.6
# necessary for evaluations (NumPy, SciPy, Matplotlib, SymPy, Pandas, IPython)
module load SciPy-Stack/2020-Python-3.8.5

# Other Software ##############################################################
#
# Manually installed libraries are stored in PARTITION_LIB
PARTITION_LIB=$PROJECT/lib_gpus


PNGWRITER_ROOT=$PARTITION_LIB/pngwriter
export CMAKE_PREFIX_PATH=$PNGWRITER_ROOT:$CMAKE_PREFIX_PATH

BLOSC_ROOT=$PARTITION_LIB/c-blosc
export CMAKE_PREFIX_PATH=$BLOSC_ROOT:$CMAKE_PREFIX_PATH
export LD_LIBRARY_PATH=$BLOSC_ROOT/lib:$LD_LIBRARY_PATH


# Environment #################################################################
#

export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70" # Nvidia V100 architecture

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/juwels-jsc"}
```

(continues on next page)

(continued from previous page)

```
export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "gpus" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/juwels-jsc/gpus.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates 2 interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    if [ $numNodes -gt 8 ] ; then
        echo "The maximal number of interactive nodes is 8." 1>&2
        return 1
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    salloc --time=1:00:00 --nodes=$numNodes --ntasks-per-node=4 --gres=gpu:4 --
↪mem=180000 -A $account -p gpus bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates 2 interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numDevices=1
    else
        if [ "$1" -gt 4 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numDevices=$1
        fi
    fi
    echo "Hint: please use 'srun --cpu_bind=sockets <COMMAND>' for launching multiple␣
↪processes in the interactive mode"
    salloc --time=1:00:00 --ntasks-per-node=$(($numDevices)) --gres=gpu:4 --
↪mem=180000 -A $account -p gpus bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.13 ARIS (GRNET)

System overview: link

User guide: link

Production directory: `$WORKDIR` (link)

For these profiles to work, you need to download the *PIConGPU source code*.

4.13.1 Queue: gpu (2 x NVIDIA Tesla k40m GPUs)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ######################################### (edit those lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on batch system jobs: NONE, BEGIN, END, FAIL, REQUEUE, ALL,
#     TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_80 and/or TIME_LIMIT_50
export MY_MAILNOTIFY="NONE"
export MY_MAIL="your email"
export MY_NAME="Name, name <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
export proj=$(groups | awk '{print $2}')

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#export EDITOR="nano"

# General modules #############################################################
#
module purge
module load gnu/6.4.0
module load cmake
module load cuda/9.2.148
module load make
module load utils
module load python/2.7.13
module load git
module load picongpu
#module load boost/1.62.0
#module load hdf5/1.8.17/gnu
# Other Software ##############################################################
#
# module load zlib/1.2.8
# module load pngwriter/0.7.0
# module load hdf5-parallel/1.8.20

# Work-Arounds ################################################################
#
# fix for Nvidia NVCC bug id 2448610
# see https://github.com/ComputationalRadiationPhysics/alpaka/issues/701
#export CXXFLAGS="-Dlinux"

# Environment #################################################################
```

(continues on next page)

(continued from previous page)

```
#

export CMAKE_PREFIX_PATH=$PICONGPUROOT
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:35"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/aris-grnet"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

#export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

# "tbg" default options #######################################################
#   - SLURM (sbatch)
#   - "gpu" queue
export TBG_SUBMIT="sbatch"
export TBG_TPLFILE="etc/picongpu/aris-grnet/gpu.tpl"

# allocate an interactive shell for one hour
#   getNode 2  # allocates two interactive nodes (default: 1)
function getNode() {
    if [ -z "$1" ] ; then
        numNodes=1
    else
        numNodes=$1
    fi
    srun --time=0:30:00 --nodes=$numNodes --ntasks-per-socket=8 --ntasks-per-node=16 -
↪-mem=252000 --gres=gpu:4 -A $proj -p dvd_usr_prod --pty bash
}

# allocate an interactive shell for one hour
#   getDevice 2  # allocates two interactive devices (default: 1)
function getDevice() {
    if [ -z "$1" ] ; then
        numGPUs=1
    else
        if [ "$1" -gt 4 ] ; then
            echo "The maximal number of devices per node is 4." 1>&2
            return 1
        else
            numGPUs=$1
        fi
    fi
    srun  --time=1:00:00 --ntasks-per-node=$numGPUs --cpus-per-task=$((4 * $numGPUs))␣
↪--gres=gpu:$numGPUs --mem=$((63000 * numGPUs)) -A $proj -p dvd_usr_prod --pty bash
}

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
```

(continues on next page)

(continued from previous page)

```
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

4.14 Ascent (ORNL)

System overview and user guide: link

Production directory: usually `$PROJWORK/$proj/` (as on summit link).

For this profile to work, you need to download the *PIConGPU source code* and install *openPMD-api and PNG-writer* manually or use pre-installed libraries in the shared project directory.

4.14.1 V100 GPUs (recommended)

```
# Name and Path of this Script ############################### (DO NOT change!)
export PIC_PROFILE=$(cd $(dirname $BASH_SOURCE) && pwd)"/"$(basename $BASH_SOURCE)

# User Information ################################# (edit the following lines)
#   - automatically add your name and contact to output file meta data
#   - send me a mail on job (-B)egin, Fi(-N)ish
export MY_MAILNOTIFY=""
export MY_MAIL="someone@example.com"
export MY_NAME="$(whoami) <$MY_MAIL>"

# Project Information ######################################## (edit this line)
#   - project account for computing time
# export proj=[TODO: fill with: `groups | awk '{print $2}'`]

# Text Editor for Tools ###################################### (edit this line)
#   - examples: "nano", "vim", "emacs -nw", "vi" or without terminal: "gedit"
#module load nano
#export EDITOR="emacs -nw"

# basic environment ################################################
module load gcc/8.1.1
module load spectrum-mpi/10.3.1.2-20200121

export CC=$(which gcc)
export CXX=$(which g++)

# required tools and libs
module load git/2.20.1
module load cmake
module load cuda/11.2.0
module load boost/1.66.0

# plugins (optional) ###############################################
module load hdf5/1.10.4
module load c-blosc/1.12.1 zfp/0.5.2 sz/2.0.2.0 lz4/1.8.1.2
module load adios2/2.7.0
module load zlib/1.2.11
module load libpng/1.6.34 freetype/2.9.1
module load nsight-compute/2021.1.0
```

(continues on next page)

(continued from previous page)

```
# shared libs
#export PIC_LIBS=$PROJWORK/$proj/picongpu/lib/

# openPMD-api
#export OPENPMD_ROOT=$PIC_LIBS/openPMD-api/
#export LD_LIBRARY_PATH=$OPENPMD_ROOT/lib64:$LD_LIBRARY_PATH

# pngWriter
#export CMAKE_PREFIX_PATH=$PIC_LIBS/pngwriter:$CMAKE_PREFIX_PATH
#export LD_LIBRARY_PATH=$PIC_LIBS/pngwriter/lib:$LD_LIBRARY_PATH


# helper variables and tools ##################################################
export PICSRC=$HOME/src/picongpu
export PIC_EXAMPLES=$PICSRC/share/picongpu/examples
export PIC_BACKEND="cuda:70"

# Path to the required templates of the system,
# relative to the PIConGPU source code of the tool bin/pic-create.
export PIC_SYSTEM_TEMPLATE_PATH=${PIC_SYSTEM_TEMPLATE_PATH:-"etc/picongpu/ascent-ornl
↪"}

export PATH=$PICSRC/bin:$PATH
export PATH=$PICSRC/src/tools/bin:$PATH

export PYTHONPATH=$PICSRC/lib/python:$PYTHONPATH

alias getNode="bsub -P $proj -W 2:00 -nnodes 1 -Is /bin/bash"

# "tbg" default options #######################################################
export TBG_SUBMIT="bsub"
export TBG_TPLFILE="etc/picongpu/ascent-ornl/gpu_batch.tpl"

# Load autocompletion for PIConGPU commands
BASH_COMP_FILE=$PICSRC/bin/picongpu-completion.bash
if [ -f $BASH_COMP_FILE ] ; then
    source $BASH_COMP_FILE
else
    echo "bash completion file '$BASH_COMP_FILE' not found." >&2
fi
```

CHAPTER

FIVE

CHANGELOG

5.1 0.7.0

Date: 2023-10-19

C++17, lasers via incidentField, DispersivePulse laser profile, sub-stepping field solver, and PICMI support

With this release, the laser initialization switches to an incidentField approach, which allows easier initialization of arbitrary, Maxwell conform electromagnetic fields in a simulation. Old `laser.param` files won't work anymore. See the documentation! This update of the laser implementation has further been used to add the new profile 'DispersivePulse' and to rename the existing laser profiles. The new profile allows initialization of Gaussian laser pulses with arbitrary first, second, and third-order dispersion, while the renaming aims at expressing a laser's profile more precisely.

Furthermore, basic PICMI support is added to PIConGPU, as well as the possibility of time sub-stepping for field solvers, and improved handling of currents in PML boundaries. Again, please see the documentation for details.

Larger code refactorings come with this release, too. To these belong a cleaning of the code base, in particular removal of unused plugins and CuSTL, performance improvements, and a switch to C++17 as the minimum required version.

As usual, this release features fixes of PIConGPU and extensions and clarifications of the documentation.

Thanks to Sergei Bastrakov, Finn-Ole Carstens, Alexander Debus, Fabia Dietrich, Simeon Ehrig, Marco Garten, Bernhard Manfred Gruber, Axel Huebl, Jeffrey Kelling, Anton Lebedev,Brian Edward Marre, Paweł Ordyna, Richard Pausch, Franz Pöschel, Klaus Steiniger, Jan Stephan, Hannes Tröpgen, Mika Soren Voß, René Widera for their contributions to this release!

User Input Changes:

- Remove deprecated laser profiles #4502
- Remove flylite #4360
- Remove FieldToParticleInterpolationNative #4369
- Remove SliceFieldPrinter plugin #4112
- Remove pusher `Axel` #4115
- Remove unused plugin Intensity #4361
- Remove PIConGPUs docker file #4374
- Remove bremsstrahlung.unitless #4494
- Remove deprecated laser profiles #4502
- Incident field laser profiles #4038 #4050 #4053 #4064 #4069 #4080 #4077 #4076 #4120 #4105 #4172 #4302 #4507 #3860
- Remove derived attribute/field MomentumComponent #4191
- Add a new functor for start position and weightings of macroparticles #3882

- Add a runtime option to set particle density file #3972
- Implement phase-coefficients in Gauss-Laguerre laser modes #3992
- Extend comments in density.param #4021
- Comments to declarations of density profile types #4024
- Add EZ current deposition type as alias to EmZ #4036
- Change way of controlling position of the Huygens surface #4056
- Specify BP5 default BufferChunkSize as 2GB #4127
- Unify default openPMD file extension in plugins #4155
- Add derived particle attributes for component of momentum and related density #4169
- Add custom normalization option to PNG output #4165
- Add a new functor to sample macroparticles in a cell #4111
- Auto adjust particle exchange buffers for double precision #4236
- Disallow binomial current interpolation together with current background #4252
- Add free temperature functor #4256
- Change user interface of free temperature functor to match free density functor #4273
- KHI memory species buffer scaling based on simulation volume #4422
- Refactor particle filters #4432
- Replace mpl by mp11 (part 1/2) #3834

New Features:

- PIC:
  - Add readthedocs description of TF/SF field generation #3877
  - Add a new functor for start position and weightings of macroparticles #3882
  - Optimize Jz calculation in EmZ 2d #3904
  - Arbitrary order incident field solver #3860 #4038 #4050 #4082 #4105
  - Add a runtime option to set particle density file #3972
  - fix warnings and check for warnings in CI #3878
  - Implement phase-coefficients in Gauss-Laguerre laser modes #3992
  - Add a check and warning printed when no volume inside the Huygens surface #4049
  - Add a warning for auto-adjusting domain to fit field absorber #4047
  - Change way of controlling position of the Huygens surface #4056
  - Check uniqueness of particle attributes #4078
  - Improve reporting of wrong MPI/devices configuration #4122
  - Check that averagely derived attributes are weighted #4154
  - Add derived particle attributes for component of momentum and related density #4169
  - Allow fractional indices in TWTSfast functors #4184
  - Calculate and output laser phase velocity #4174
  - Remove derived attribute/field MomentumComponent #4191
  - HIP: use compute current strategy `strategy::CachedSupercellsScaled<2>` #4218
  - Add a new functor to sample macroparticles in a cell #4111

  - Add time-substepping versions of existing field solvers #3804
  - Add time interpolation/extrapolation of J in substepping field solver #4258
  - Improve treating currents in PML #4293
  - Add a new particle attribute for weighting damping factor #4300
  - FLYonPIC atomic physics for PIConGPU #4134
  - fix warning in `Derivative.hpp` #4508
- PMacc:
  - allow time dependent checkpointing #4042
  - Clarify performance warning on send/receive buffer being too small #4171
  - provide a pmacc CMake target #4257
  - PMacc tests: use the CI provided GPU architecture #4493
  - Replace mpl by mp11 (part 1/2) #3834
- plugins:
  - openPMD plugin: Set default backend depending on available options #4008
  - openPMD plugin: throw error when neither defining period or toml #4028
  - Clarify an inaccurate exception message in calorimeter plugin #4027
  - Specify BP5 default BufferChunkSize as 2GB #4127
  - Add custom normalization option to PNG output #4165
  - OpenPMD plugin data range selection #4096
  - Switch to .bp4 extension for ADIOS2 if available, print available extensions in help message #4234
- tools:
  - Crusher hipcc Profile: Fix export of env variables #3976
  - Crusher hipcc profile #3973
  - Fix EnergyHistogram and clarify usage #3970
  - Fix PICSRC in bash_picongpu.profile #3999
  - Enable –mpiDirect for crusher-ornl #4007
  - Add profiles for A100 GPUs on Taurus #3998
  - hzdr dev server profile #4086
  - crusher: add slurm fault tolerance #4230
  - `pic-configure` validate preset selection #4235
  - crusher tpl: avoid false error message #4261
  - Activate cuda_memtest on crusher #4216
  - Add template for async jobs on Crusher #4413
  - Async template for Juwels Booster, Crusher and Summit #4431
  - Added script to edit copied PIConGPU checkpoints. #4520
  - gpu profile for dev-server #4183

Bug Fixes:

- PIC:
  - Fix missing euler number in ADK formula #4696

  - Fix incorrect assignment of Jz in 2d EmZ implementation #3892
  - Fix getExternalCellsTotal() to properly account for guard cells #4030
  - Fix incident field math around corners #4102
  - Fix index shift in 2d incident field kernel #4202
  - Fix fieldBackground constructors to be HINLINE #4013
  - Better separate CFL and pusher/current deposition requirements #4237
  - Skip J contribution to E for None field solver #4243
  - Properly account for current background when deciding to skip adding J #4247
  - Disallow binomial current interpolation together with current background #4252
  - Change TYPICAL_NUM_PARTICLES_PER_MACROPARTICLE to always be 64-bit #4263
  - Update binary collisions, dynamic Coulomb log #4295
  - Remove explicit definition of RngType in filter::generic::FreeRng #4440
  - Fix particle position rounding issue #4463
- PMacc:
  - Fix copying of cuSTL device buffers #4029
  - Fix function `toTimeSlice` #4040
  - Fix implementation of pmacc::makeDeepCopy() and field background #4192
  - Fix soft restart #4034
  - Fix PMacc reduce #4328
- plugins:
  - Add missing communication in XrayScattering #3937
  - Fix rad restart #3961
  - Fix calculation of total current in the SumCurrents plugin for 2d case #4118
  - Fix radiation trees #4231
- tools:
  - Add missing double quotes in handleSlurmSignals #4161
  - Fix wrong memory requirment calculation in .tpl files #4308
  - Fix PICSRC in bash_picongpu.profile #3999
- Fix that PMacc depends on PIConGPU type definitions #3925
- HIP: allow full architecture definition #3941
- Make the span-based storeChunk API opt-in in various openPMD-based IO routines #3933
- Fix openPMD outputs negative particle patch offset #4037
- Fix CUDA container apt keys #4085
- Add HIP and mallocMC + HIP to PIConGPU version output #4139
- Do not forceinline recursive functions #4147
- PNG plugin: fix invalid memory access in 2D #4225
- Fix using uninitilized variable #4303
- Fix openPMD particles `positionOffset` **breaking change** #4283
- Fix precision cast is returning a reference #4337

Misc:

- Refactoring:
  - PIC:
    - Refactor absorbing and reflecting particle BC #3875
    - Move yee::StencilFunctor to fdtd::StencilFunctor #3894
    - Improve performance of MoveParticle #3908
    - Cleanup current deposition implementation #3944
    - Use `if constexpr()` to check for simulation/object dimension #4107
    - Avoid using macro define `SIMDIM` #4116
    - Switch to alpaka::Complex<> #4114
    - Add a helper type for unique typelist of enabled incident field profiles #4177
    - Cached shapes #4124
    - Use cached shapes for current deposition #4170
    - Add device side `ForEachParticle` #4173
    - Workaround wrong reported free memory #4219
    - Simplify implementation of incident field kernel #4226
    - Auto adjust particle exchange buffers for double precision #4236
    - Make adding J term part of field solver #4245
    - Refactor adding current density to electomagnetic field #4255
    - Add free temperature functor #4256
    - Add singleton-like access for AbsorberImpl #4262
    - Simplify PmlImpl interface for getting update functors #4266
    - Remove Cell2Particle #4289
    - Remove getGlobalSuperCells #4325
    - Move include collision.unitless to the default loader #4326
    - Refactor lockstep programming #4321
    - Switch to std::optional in compute field value #4345
    - Use cupla/alpaka's min/max function #4350
    - Remove flylite #4360
    - Remove cuSTL #4363 #4398 #4402
    - Remove Bremsstrahlung and Synchrotron radition extension #4395
    - Remove unused vectors transition radiation #4427
    - Fix incident field calculation order #4451
    - Improve command line argument validation #4447
    - Change class RelayPoint to function #4464
    - OnePositionImpl refactoring #4465
    - Refactor particle pusher launcher #4471
    - Refactor differentiation trait #4473
    - Refactor GetChargeState and GetCharge #4474

    - Remove usage of `boost::result_of` #4505
  - PMacc:
    - Avoid overaligning particle frame data #3807
    - Remove `MappedBufferIntern` #4018
    - Use `if constexpr` in `lockstep::ForEach` #4095
    - Rework implementation of pmacc::Unique #4176
    - Remove HIP `atomicAdd(float)` emulation #4269
    - Split PMacc objects into hpp and cpp files #4260
    - DataBox remove method `reduceZ()` #4291
    - Refactor CountParticles and particle filters #4313
    - PMacc: refactor kernel en-queuing #4317
    - Use alpaka `mem_fence()` #4349
    - Introduce a constexpr value identifier #4344
    - Remove boost workaround #4373
    - Remove unused cuSTL classes #4415
    - Extend global reduce interface #4425
    - Remove cuSTL #4434
    - Cleanup PMacc's event system #4443
    - Remove buffer `*Intern` #4448
    - Topic lockstep for each refactoring #4460
    - PMacc: remove unused class math::Tuple #4461
    - PMacc: remove invokeIf #4467
    - PMacc: refactor meta::ForEach #4469
    - Refactor `TaskSetValue` #4470
    - Add HDINLINE to Selection methods #4500
    - PMacc: header include validation #3924
    - Remove dependency to `boost::math` #4503
    - PMacc: DataConnector change `get()` interface #4506
    - Refactor `math::Vector`, use unsigned dimension #4499
    - Remove `boost::filesystem usage` #4504
    - Remove particle merger from `TBG_macros.cfg` #4512
    - Rename `abs2` -> `l2norm2` and `abs` for vectors to `l2norm` #4509
    - Replace mpl by mp11 (part 1.5/2) #4516
  - Plugins:
    - Add optionDefined() to plugins::multi::Option struct #3906
    - Read IO plugin configuration (openPMD plugin) from reading applications (TOML) #3720
    - openPMD-based plugins: choose default backend depending on what is available #4014
    - Remove cupla compatibility macro definition #4017
    - Refactor openPMD plugin #4054

    * Improve comments and naming in PNG output #4164
    * openPMD plugin: add IO file version #4287
    * openPMD plugin: Flush data to disk within a step #4002
    * openPMD plugin: JSON parameters for restarting #4329
    * Use cuplas/alpakas mapped memory #4348
    * Remove xray scattering #4352
    * Remove plugin: PositionsParticle #4371
    * Remove plugin: recource log #4372
    * Remove particle merging plugins #4388
    * Radiation plugin: Enable streaming #4387
    * Plugin radiation: use PMaccs forEachFrame #4377
    * Plugin emittance: remove cuSTL #4426
    * Code clean-up: avoid using new and plane pointers #4428
    * Charge conservation plugin: remove cuSTL #4424
    * Refactor particle filters #4432
    * Radiation plugin: refactor particle filter execution #4466
    * Radiation plugin: refactor getRadiationMask #4468
    * Plugin: CountMacroParticlesPerSupercell and ChargeConservation #4496
    * Plugin sum current refactoring #4497
    * Fix logging message in openPMD plugin #4044
  - Tools:
    * Update to cupla0.4.0 dev, cmake-modules and cuda_memtest #3868
    * cupla 0.4.0-dev + alpaka 0.8.0-dev #3915
    * Optimize CI #4437
    * Switch PICMI to incidentField #4459
    * Refactor `pic-create` #4515
  - Unroll interpolation #3859
  - Update to clang-format-12 #3826
  - PML: avoid `stack frames` in GPU kernel #3881
  - Version bump: openPMD-api from 0.12.0 to 0.14.3 #3913
  - Switch to C++17 #3949
  - Add combined derived attributes #3943
  - Remove pusher `Axel` #4115
  - Unify default openPMD file extension in plugins #4155
  - Improve comments and naming in PNG output in docs and examples #4168
  - Switch to std::void_t #4195
  - Change user interface of free temperature functor to match free density functor #4273
  - Change particle attribute for thermal boundaries to openPMD-conformant name #4278
  - PIConGPU use RPATH by default #4312

   - Remove FieldToParticleInterpolationNative #4369
   - Remove unused file `LinearInterpolationWithUpper` #4370
   - Switch to CI container version 3 #4378
   - Plugin: phase space remove cuSTL #4404
   - Clean Bunch example #4419
   - Example foilLCT: use incident field laser #4488
- Documentation:
   - Upgrade to Boost 1.66 #3939
   - Docs: Fix Stale LSF Links #3981
   - Expand documentation on running interactively #3967
   - Extend comments in density.param #4021
   - Add doxygen comments to declarations of density profile types #4024
   - Add a doc section on general usage of CPU systems #4039
   - Merge 0.6.0 changes back to the development branch #4079
   - Fix comments on axis definitions in incident field profiles #4083
   - Switch to alpaka 0.9.0 #4081
   - Fix a few instances of calling the Taflove-Hagness book just Taflove #4103
   - Fix some typos in the reference doc page #4117
   - Fix readthedocs example of context variable default initialization #4133
   - Fix incorrect doxygen includes in readthedocs #4132
   - Remove zlib from required dependencies in the docs #4128
   - Update documentation of dependencies #4136
   - Clarify description of numerical dispersion for Yee solver #4156
   - Remove field slice printer arguments from TBG_macros.cfg #4188
   - Documentation: mention `CMAKE_BUILD_TYPE` #4229
   - Fix copyright years ending in 2021 in some files #4242
   - Clarify docs on setting same sources for incident field on each side #4232
   - Clarify/fix comment for AOFDTD derivative functors #4246
   - Clarify approximation order of J in AOFDTD docs #4248
   - Add docs on substepping field solver #4259
   - Add Python code formatting guidelines to the docs #4277
   - Install dependencies for memoryPerDevice example #4327
   - Add papers Cristian Bontoiu #4445
- Fix warning: add explicit braces to avoid dangling else [-Wdangling-else] #3869
- Fix field absorber example that violated CFL for AOFDTD #3871
- Spack load: no more "-r" #3873
- CI: update to container version 2.0 #3934
- Add –ntasks-per-node and -np to .tpl files on Hemera #3940
- Adjust taurus k80 modules to work with c++17 #3962

- Don't install toml11 internally shipped library #3986
- Change the container URL of the GitLab CI to the new registry #4001
- Fix fieldBackground constructors to be HINLINE #4013
- Add ARM architecture templates #4033
- Add EZ current deposition type as alias to EmZ #4036
- Adjust laser in incident field example for a more realistic setup #3955
- System DICC: add a system description and a profile for the GPU queue #4015
- Remove example: ThermalTest #4100
- Add Thermal electron benchmark #4099
- CUDA: fix false warning #4113
- Bugfix JUWELS Booster profile #4151
- Add new benchmark TWEAC-FOM #4167
- Update to cupla with alpaka 1.0.0-dev and mallocMC #4166
- Extend TWEAC-FOM benchmark with incident field option for laser generation #4197
- Add componentwise incident field calculation option to TWEAC-FOM benchmark #4200
- Update cuda_memtest #4214
- Update mallocMC to fix possible out of memory access #4220
- CI: disable all optional dependencies for the Empty test case #4254
- Add and enable UCX_RC_TIMEOUT workaround #4288
- Add compilation of particle merger plugins to compile-time tests #4297
- Update to latest cupla dev branch #4309
- CI: add CUDA 11.3 - 11.4, HIP 4.5, 5.0 - 5.2 #4315
- Add new openPMD-api module for hemera kepler #4320
- Fix HIP CI tests #4353
- Remove PIConGPU install description using spack #4375
- KHI memory species buffer scaling based on simulation volume #4422
- Remove warm copper example #4433
- More variants: test KHI growth rate #4452
- Example bunch: use incident field laser #4449
- Remove old cluster profiles #4514
- Fix single particle test example #4482

5.2 0.6.0

Date: 2021-12-21

C++14, New Solvers, I/O via openPMD API, HIP Support

This release switches to C++14 as minimum required version. Transition to C++17 is planned for upcoming releases.

We extended PIConGPU with a few new solvers. Binary collisions are now available. We added arbitrary-order FDTD Maxwell's solver. All field solvers are now compatible with perfectly matched layer absorber, which became default. Yee solver now supports incident field generation using total field/scattered field technique. We added Higuera-Cary particle pusher and improved compatibility of pushers with probe species. Implementation of particle boundaries was extended to support custom positions, reflecting and thermal boundary kinds were added.

With this release, PIConGPU fully switches to openPMD API library for performing I/O. The native HDF5 and ADIOS output plugins were replaced with a new openPMD plugin. All other plugins were updated to use openPMD API. Plugins generally support HDF5 and ADIOS2 backends of openPMD API, a user can choose file format based on their installation of openPMD API. We also added new plugins for SAXS and particle merging.

We added support for HIP as a computational backend. In particular, it allows running on AMD GPUs. Several performance optimizations were added. Some functors and plugins now have performance-influencing parameters exposed to a user.

The code was largely modernized and refactored, documentation was extended.

Thanks to Sergei Bastrakov, Kseniia Bastrakova, Brian Edward Marre, Alexander Debus, Marco Garten, Bernhard Manfred Gruber, Axel Huebl, Jakob Trojok, Jeffrey Kelling, Anton Lebedev, Felix Meyer, Paweł Ordyna, Franz Poeschel, Lennert Sprenger, Klaus Steiniger, Manhui Wang, Sebastian Starke, Maxence Thévenet, Richard Pausch, René Widera for contributions to this release!

5.2.1 Changes to "0.5.0"

User Input Changes:

- Remove HDF5 I/O plugin (replaced with new openPMD plugin) #3361
- Remove ADIOS I/O plugin (replaced with new openPMD plugin) #3691
- Decouple field absorber selection from field solver #3635
- Move all field absorber compile-time parameters to fieldAbsorber.param #3645
- Change default field absorber to PML #3672
- Change current interpolation from compile-time to command-line parameter #3552
- Remove directional splitting field solver #3363
- Remove compile-time movePoint value from grid.param #3793
- Switch to cupla math functions #3245
- Scale particle exchange buffer #3465
- Update to new mallocMC version and parameters #3856

New Features:

- PIC:
  - Add strategy parameter for current deposition algorithms #3221
  - Add Higuera-Cary particle pusher #3280 #3371
  - Add support for PML absorber with Lehe field solver #3301
  - Add support for Lehe field solver in 2D #3321

  - Add arbitrary-order FDTD field solver #3338
  - Add ionization current #3355
  - Add PML support to arbitrary-order FDTD field solver #3417
  - Faster TWTS laser implementation #3439
  - Parametrize FieldBackground for accuracy vs. memory #3527
  - Add a check for Debye length resolution #3446
  - Add user-defined iteration start pipeline
  - Add binary collisions #3416
  - Expose compute current worker multiplier #3539
  - Add a new way to generate incident field using TF/SF #3592
  - Add support for device oversubscription #3632
  - Signal handling #3633
  - Replace FromHDF5Impl density with FromOpenPMDImpl #3655
  - Introduce filtered particleToGrid algorithm #3574
  - Record probe data with all pushers #3714
  - Compatibility check for laser and field solver #3734
  - Extend particles::Manipulate with area parameter #3747
  - Enable runtime kernel mapping #3750
  - Optimize particle pusher kernel #3775
  - Enable particle boundaries at custom positions #3763 #3776
  - Adjust physics log output #3825
  - Add a new particle manipulator combining total cell offset and RNG #3832
  - Add reflective particle boundary conditions #3806
  - Add thermal particle boundary conditions #3858
  - HIP support #3356 #3456 #3500
- PMacc:
  - MPI direct support #3195
  - Improve output of kernel errors in PMacc with PMACC_BLOCKING_KERNEL=ON #3396
  - Optimize atomic functor for HIP #3457
  - Add device-side assert macro #3488
  - Add support for alpaka OpenMP target/OpenACC #3512
  - Additional debug info in guard check for supercell size #3267
  - Clarify error message and add comments in GameOfLife #3553
  - Print milliseconds in output #3606
  - Enable KernelShiftParticles to operate in guard area #3772
- plugins:
  - Add a new openPMD plugin to replace the native HDF5 and ADIOS1 ones #2966
  - Add probabilistic particle merger plugin #3227
  - Add SAXS plugin using electron density #3134

  - Add work distribution parameter in radiation plugin #3354
  - Add field solver parameters attribute to output #3364
  - Update required openPMD version to 0.12.0 #3405
  - Use Streaming API in openPMD plugin #3485
  - Make radiation Amplitude class template #3519
  - Add compatibility to ISAAC GLM #3498
  - Extend JSON patterns in openPMD plugin #3513
  - Compatibility to new unified ISAAC naming scheme #3545
  - Use span-based storeChunk API in openPMD plugin #3609
  - Optimize radiation plugin math #3711
  - Deactivate support for writing to ADIOS1 via the openPMD plugin #3395
  - Avoid hardcoding of floating-point type in openPMD plugin #3759
  - Add checkpointing of internal RNG states #3758
  - Make the span-based storeChunk API opt-in in various openPMD-based IO routines #3933
- tools:
  - Update to C++14 #3242
  - Add JetBrains project dir to .gitignore. #3265
  - Convert path to absolute inside tbg #3190
  - Add verbose output for openPMD-api #3281
  - Switch to openPMD API in `plot_chargeConservation_overTime.py` #3505
  - Switch to openPMD API in `plot_chargeConservation.py` #3504
  - Save the used cmakeFlags setup in output directory #3537
  - Add JUWELS Booster profile #3341
  - Check for doxygen style in CI #3629
  - Update Summit profile #3680
  - Remove old CI helper #3745
  - Remove uncrustify #3746
  - Add python env to gitignore #3805

Bug Fixes:

- PIC:
  - Fix compilation with clang-cuda and Boost #3295
  - Modify FreeFormulaImpl density profile to evalute a functor in centers of cells #3415
  - Fix deprecation warnings #3467
  - Fix the scheme of updating convolutional B in PML by half time step #3475
  - Fix and modernize the generation scheme for MPI communication tags #3558
  - Fix TWTS implementations #3704
  - Fix domain adjuster for absorber in case there is a single domain along a direction #3760
  - Fix unreachable code warnings #3575
  - Fix incorrect assignment of Jz in 2d EmZ implementation #3893

  - Fix restarting with moving window #3902
  - Fix performance issue with HIP 4.3+ #3903
  - Fix using host function in device code #3911
- PMacc:
  - Fix missing override for virtual functions #3315
  - Fix unsafe access to a vector in cuSTL MPI reduce #3332
  - Fix cuSTL CartBuffer dereferencing a null pointer when CUDA is enabled #3330
  - Remove usage of `atomicAddNoRet` for HIP 4.1 #3572
  - Fix compilation with icc #3628
  - Fix warning concerning used C++17 extension #3318
  - Fix unused variable warning #3803
- plugins:
  - Fix checkpointing and output of PML fields for uneven domains #3276
  - Fix warnings in the openPMD plugin #3289
  - fix clang-cuda compile #3314
  - Fix shared memory size in the PhaseSpace plugin #3333
  - Fix observation direction precision to use float_X #3638
  - Fix crash in output after moving window stopped #3743
  - Fix openPMD warning in XrayScatteringWriter #3358
  - Fix warning due to missing virtual destructor #3499
  - Fix warning due to missing override #3855
  - Remove deprecated openPMD::AccessType and replace with openPMD::Access #3373
  - Fix processing of outgoing particles by multi plugins #3619
  - Fix getting unitSI for amplitude #3688
  - Fix outdated exception message for openPMD plugin with ADIOS1 #3730
  - Remove usused variable in ISAAC plugin #3756
  - Fix warning in radiation plugin #3771
  - Fix internal linkage of private JSON header in openPMD plugin #3863
  - Fix treatment of bool particle attributes in openPMD plugin #3890
  - Add missing communication in XrayScattering #3937
- tools:
  - Fix plotting tool for numerical heating #3324
  - Fix typos in pic-create output #3435
  - No more "-r" in spack load #3873
  - Fix docker recipe #3921
- Fix KHI for non-CUDA devices #3285
- Fix clang10-cuda compile #3310
- Fix segfault in thermal test #3517
- Fix jump on uninitialized variable #3523

- Fix EOF whitespace test #3555
- Disable PMacc runtime tests for HIP in CI #3650

Misc:

- refactoring:
  - PIC:
    - Use cupla instead of cuda prefix for function calls #3211
    - Abstract PML definitions from YeePML field solver #3283
    - Refactor implementations of derivatives and curls for fields #3309
    - Refactor Lehe solver to use the new derivative and curl functors #3317
    - Add pmacc functions to get basis unit vectors #3319
    - Refactor margin traits for field derivatives #3320
    - Add a new trait GetCellType of field solvers #3322
    - Remove outdated pmacc/nvidia/rng/* #3351
    - Add generic utilities to get absorber thickness #3348
    - Remove pmacc::memory::makeUnique #3353
    - Remove unused HasIonizersWithRNG and UsesRNG #3367
    - Cleanup particle shape structs #3376
    - Refactoring and cleanup of species.param #3431
    - Rename picongpu::MySimulation to picongpu::Simulation #3476
    - Avoid access to temporary variable #3508
    - Remove unused enum picongpu::FieldType #3556
    - Switch internal handling of current interpolation from compile-time to run-time #3551
    - Refactor and update comments of GetMargin #3564
    - Remove enum picongpu::CommunicationTag #3561
    - Add a version of GetTrait taking only field solver type as a parameter #3571
    - Remove interface `DataConnector::releaseData()` #3582
    - Remove unused directory pmacc/nvidia #3604
    - Rename picongpu::particles::CallFunctor to pmacc::functor::Call #3608
    - Refactor common field absorber implementation #3611
    - Move PML implementation to a separate directory #3617
    - Move the basic implementation of FDTD field solver to the new eponymous template #3643
    - Use IUnary with filters in collisions and FieldTmp #3687
    - Refactor field update functor #3648
    - Replace leftover usage of boost/type_traits with std:: counterparts #3812
    - Relocate function to move particles #3802
    - Fix a typo in domain adjuster output #3851
    - Replace C-style stdlib includes with C++ counterparts #3852
    - Replace raw pointers used for memory management with smart pointers #3854
    - Refactor and document laser profiles #3798

    * Remove unused variable boundaryKind #3769

- PMacc:
    * Fix pedantic warnings #3255
    * Refactor ConstVector #3274
    * Set default values for some PMacc game of life example arguments #3352
    * Refactor PMacc warp and atomic functions #3343
    * Change SharedMemAllocator function qualifiers from DEVICEONLY to DINLINE #3520
    * Delete unused file MultiGridBuffer.hpp #3565
    * Refactor lockstep::ForEach #3630
    * Refactor lockstep programming #3616
    * Change ExchangeTypeToRank from protected to public #3718
    * Clarify naming and comments for pmacc kernel mapping types #3765
    * Separate notification of plugins into a new function in SimulationHelper #3788
    * Add a factory and a factory function for pmacc::StrideMapping #3785
    * Replace custom compile-time string creation by BOOST_METAPARSE_STRING #3792
    * Refactor CachedBox #3813
    * Refactor and extend Vector #3817
    * Drop unused TwistedAxesNavigator #3821
    * Refactor DataBox #3820
    * Replace PitchedBox ctor with offset by DataBox.shift() #3828
    * Rename type to Reference in Cursor #3827
    * Fix a warning in MapperConcept #3777
    * Remove leftover mentions of ADIOS1 #3795
- plugins:
    * Remove unused ParticlePatches class #3419
    * Switch to openPMD API in PhaseSpace plugin #3468
    * Switch to openPMD API in MacroParticleCounter plugin #3570
    * Switch to openPMD API in ParticleCalorimeter plugin #3560
    * Vector field vis vis compatility and benchmarking in ISAAC plugin #3719
    * Remove libSplash #3744
    * Remove unnecessary parameter in writeField function template of openPMD plugin #3767
    * Remove –openPMD.compression parameter #3764
    * Rename instance interface for multiplugins #3822
    * Use multidim access instead of two subscripts #3829
    * Use Iteration::close() in openPMD plugin #3408
- tools:
    * Add cmake option to enforce dependencies #3586
    * Update to mallocMC 2.5.0crp-dev #3325
    * Clarify output concerning cuda_memtest not being available #3345

  - Reduce complexity of examples #3323
  - Avoid species dependency in field solver compile test #3430
  - Switch from Boost tests to Catch2 #3447
  - Introduce clang format #3440
  - Use ubuntu 20.04 with CUDA 11.2 in docker image #3502
  - Apply sorting of includes according to clang-format #3605
  - Remove leftover boost::shared_ptr #3800
  - Modernize according to clang-tidy #3801
  - Remove more old boost includes and replace with std:: counterparts #3819
  - Remove boost::result_of #3830
  - Remove macro __deleteArray() #3839
- documentation:
  - Fix a typo in the reference to [Pausch2018] #3214
  - Remove old HZDR systems #3246
  - Document C++14 requirement #3268
  - Add reference in LCT Example #3297
  - Fix minor issues with the sphinx rst formating #3290
  - Fix the name of probe fieldE and fieldB attribute in docs #3385
  - Improve clarity of comments concerning density calculation and profile interface #3414
  - Clarify comments for generating momentums based on temperature #3423
  - Replace travis badges with gitlab ones #3451
  - Update supported CUDA versions #3458
  - Juwels profile update #3359
  - Extend reminder to load environment in the docs with instructions for spack #3478
  - Fix typo in FieldAbsorberTest #3528
  - Improve documentation of clang-format usage #3522
  - Fix some outdated docs regarding output and checkpoint backends #3535
  - Update version to 0.6.0-dev #3542
  - Add installation instructions for ADIOS2 and HDF5 backends of openPMD API #3549
  - Add support for boost 1.74.0 #3583
  - Add brief user documentation for boundary conditions #3600
  - Update grid.param file doxygen string #3601
  - Add missing arbitrary order solver to the list of PML-enabled field solvers #3550
  - Update documentation of AOFDTD #3578
  - Update spack installation guide #3640
  - Add fieldAbsorber runtime parameter to TBG_macros.cfg #3646
  - Extend docs for the openPMD plugin regarding –openPMD.source #3667
  - Update documentation of memory calculator #3669
  - Summit template and documentation for asynchronous writing via ADIOS2 SST #3698

  - Add DGX (A100) profile on Cori #3694
  - Include clang-format in suggested contribution pipeline #3710
  - Extend TBG_macros and help string for changing field absorber #3736
  - Add link to conda picongpu-analysis-environment file to docs #3738
  - Add a brief doc page on ways of adding a laser to a simulation #3739
  - Update cupla to 0.3.0 release #3748
  - Update to malloc mc2.6.0crp-dev #3749
  - Mark openPMD backend libraries as optional dependencies in the docs #3752
  - Add a documentation page for developers that explains how to extend PIConGPU #3791
  - Extend doc section on particle filter workflows #3782
  - Add clarification on pluginUnload #3836
  - Add a readthedocs page on debugging #3848
  - Add a link to the domain definitions wiki in the particle global filter workflow #3849
  - add reference list #3273
  - Fix docker documentation #3544
  - Extend comments of CreateDensity #3837
  - Refactor and document laser profiles #3798
  - Mark CUDA 11.2 as supported version #3503
  - Document intention of Pointer #3831
  - Update ISAAC plugin documentation #3740
  - Add a doxygen warning to not remove gamma filter in transition radiation #3857
  - Extend user documentation with a warning about tools being Linux-only #3462
- Update hemera modules and add openPMDapi module #3270
- Separate project and account for Juwels profile #3369
- Adjust tpl for juwels #3368
- Delete superfluous `fieldSolver.param` in examples #3374
- Build CI tests with all dependencies #3316
- Update openPMD and ADIOS module #3393
- Update summit profile to include openPMD #3384
- Add both adios and adios2 module on hemera #3411
- Use openPMD::getVersion() function #3427
- Add SPEC benchmark example #3466
- Update the FieldAbsorberTest example to match the Taflove book #3487
- Merge mainline changes back into dev #3540
- SPEC bechmark: add new configurations #3597
- Profile for spock at ORNL #3627
- CI: use container version 1.3 #3644
- Update spock profile #3653
- CI: use HIP 4.2 #3662

- Compile for MI100 architecture only using spock in ORNL #3671
- Fix compile warning using spock in ORNL #3670
- Add radiation cases to SPEC benchmark #3683
- Fix taurus-tud k80 profile #3729
- Update spock profile after update to RHEL8 #3755
- Allow interrupting job in CI #3761
- Fix drift manipulator in SingleParticle example #3766
- Compile on x86 runners in CI #3762
- Update gitignore for macOS system generated files #3779
- Use stages for downstream pipe in CI #3773
- Add Acceleration pusher to compile-time test suite #3844
- Update mallocMC #3897

5.3 0.5.0

Date: 2020-06-03

Perfectly Matched Layer (PML) and Bug Fixes

This release adds a new field absorber for the Yee solver, convolutional perfectly matched layer (PML). Compared to the still supported exponential dampling absorber, PML provides better absorption rate and much less spurious reflections.

We added new plugins for computing emittance and transition radiation, particle rendering with the ISAAC plugin, Python tools for reading and visualizing output of a few plugins.

The release also adds a few quality-of-life features, including a new memory calculator, better command-line experience with new options and bashcompletion, improved error handling, cleanup of the example setups, and extensions to documentation.

Thanks to Igor Andriyash, Sergei Bastrakov, Xeinia Bastrakova, Andrei Berceanu, Finn-Ole Carstens, Alexander Debus, Jian Fuh Ong, Marco Garten, Axel Huebl, Sophie Rudat (Koßagk), Anton Lebedev, Felix Meyer, Pawel Ordyna, Richard Pausch, Franz Pöschel, Adam Simpson, Sebastian Starke, Klaus Steiniger, René Widera for contributions to this release!

5.3.1 Changes to "0.4.0"

User Input Changes:

- Particle pusher acceleration #2731
- stop moving window after N steps #2792
- Remove unused ABSORBER_FADE_IN_STEPS from .param files in examples #2942
- add namespace "radiation" around code related to radiation plugin #3004
- Add a runtime parameter for window move point #3022
- Ionization: add silicon to pre-defines #3078
- Make dependency between boundElectrons and atomicNumbers more explicit #3076
- openPMD: use particle id naming #3165
- Docs: update `species.param` #2793 #2795

New Features:

- PIC:
  - Particle pusher acceleration #2731
  - Stop moving window after N steps #2792
  - Auto domain adjustment #2840
  - Add a wrapper around main() to catch and report exceptions #2962
  - Absorber perfectly matched layer PML #2950 #2967
  - Make dependency between boundElectrons and atomicNumbers more explicit #3076
- PMacc:
  - `ExchangeTypeNames` Verify Parameter for Access #2926
  - Name directions in species buffer warnings #2925
  - Add an implementation of exp for pmacc vectors #2956
  - SimulationFieldHelper: getter method to access cell description #2986
- plugins:
  - PhaseSpaceData: allow multiple iterations #2754
  - Python MPL Visualizer: plot for several simulations #2762
  - Emittance Plugin #2588
  - DataReader: Emittance & PlotMPL: Emittance, SliceEmittance, EnergyWaterfall #2737
  - Isaac: updated for particle rendering #2940
  - Resource Monitor Plugin: Warnings #3013
  - Transition radiation plugin #3003
  - Add output and python module doc for radiation plugin #3052
  - Add reference to thesis for emittance plugin doc #3101
  - Plugins: ADIOS & PhaseSpace Wterminate #2817
  - Calorimeter Plugin: Document File Suffix #2800
  - Fix returning a stringstream by value #3251
- tools:
  - Support alpaka accelerator `threads` #2701
  - Add getter for omega and n to python module #2776
  - Python Tools: Incorporate sim_time into readers and visualizers #2779
  - Add PIConGPU memory calculator #2806
  - Python visualizers as jupyter widgets #2691
  - pic-configure: add `--force/-f` option #2901
  - Correct target thickness in memory calculator #2873
  - CMake: Warning in 3.14+ Cache List #3008
  - Add an option to account for PML in the memory calculator #3029
  - Update profile hemera-hzdr: CMake version #3059
  - Travis CI: OSX sed Support #3073
  - CMake: mark cuda 10.2 as tested #3118
  - Avoid bash completion file path repetition #3136

  - Bashcompletion #3069
  - Jupyter widgets output capture #3149
  - Docs: Add ionization prediction plot #2870
  - pic-edit: clean cmake file cache if new param added #2904
  - CMake: Honor _ROOT Env Hints #2891
  - Slurm: Link stdout live #2839

Bug Fixes:

- PIC:
  - fix EveryNthCellImpl #2768
  - Split `ParserGridDistribution` into `hpp/cpp` file #2899
  - Add missing inline qualifiers potentially causing multiple definitions #3006
  - fix wrong used method prefix #3114
  - fix wrong constructor call #3117
  - Fix calculation of omega_p for logging #3163
  - Fix laser bug in case focus position is at the init plane #2922
  - Fix binomial current interpolation #2838
  - Fix particle creation if density zero #2831
  - Avoid two slides #2774
  - Fix warning: comparison of unsigned integer #2987
- PMacc:
  - Typo fix in Send/receive buffer warning #2924
  - Explicitly specify template argument for std::forward #2902
  - Fix signed int overflow in particle migration between supercells #2989
  - Boost 1.67.0+ Template Aliases #2908
  - Fix multiple definitions of PMacc identifiers and aliases #3036
  - Fix a compilation issue with ForEach lookup #2985
- plugins:
  - Fix misspelled words in plugin documentation #2705
  - Fix particle merging #2753
  - OpenMPI: Use ROMIO for IO #2857
  - Radiation Plugin: fix bool conditions for hdf5 output #3021
  - CMake Modules: Update ADIOS FindModule #3116
  - ADIOS Particle Writer: Fix timeOffset #3120
  - openPMD: use particle id naming #3165
  - Include int16 and uint16 types as traits for ADIOS #2929
  - Fix observation direction of transition radiation plugin #3091
  - Fix doc transition radiation plugin #3089
  - Fix doc rad plugin units and factors #3113
  - Fix wrong underline in TransRad plugin doc #3102

  - Fix docs for radiation in 2D #2772
  - Fix radiation plugin misleading filename #3019
- tools:
  - Update cuda_memtest: NVML Noise #2785
  - Dockerfile: No SSH Deamon & Keys, Fix Flex Build #2970
  - Fix hemera k80_restart.tpl #2938
  - Templates/profile for hemera k20 queue #2935
  - Splash2txt Build: Update deps #2914
  - splash2txt: fix file name trimming #2913
  - Fix compile splash2txt #2912
  - Docker CUDA Image: Hwloc Default #2906
  - Fix Python EnergyHistogramData: skip of first iteration #2799
- Spack: Fix Compiler Docs #2997
- Singularity: Workaround Chmod Issue, No UCX #3017
- Fix examples particle filters #3065
- Fix CUDA device selection #3084
- Fix 8.cfg for Bremsstrahlung example #3097
- Fix taurus profile #3152
- Fix a typo in density ratio value of the KHI example #3162
- Fix GCC constexpr lambda bug #3188
- CFL Static Assert: new grid.param #2804
- Fix missing exponent in fieldIonization.rst #2790
- Spack: Improve Bootstrap #2773
- Fix python requirements: remove sys and getopt #3172

Misc:

- refactoring:
  - PIC:
    - Eliminate M_PI (again) #2833
    - Fix MappingDesc name hiding #2835
    - More fixes for MSVC capturing constexpr in lambdas #2834
    - Core Particles: C++11 Using for Typedef #2859
    - Remove unused getCommTag() in FieldE, FieldB, FieldJ #2947
    - Add a using declaration for Difference type to yee::Curl #2955
    - Separate the code processing currents from MySimulation #2964
    - Add DataConnector::consume(), which shares and consumes the input #2951
    - Move picongpu/simulationControl to picongpu/simulation/control #2971
    - Separate the code processing particles from MySimulation #2974
    - Refactor cell types #2972
    - Rename `compileTime` into `meta` #2983

    * Move fields/FieldManipulator to fields/absorber/ExponentialDamping #2995
    * Add picongpu::particles::manipulate() as a high-level interface to particle manipulation #2993
    * `particles::forEach` #2991
    * Refactor and modernize implementation of fields #3005
    * Modernize ArgsParser::ArgsErrorCode #3023
    * Allow constructor for density free formular functor #3024
    * Reduce PML memory consumption #3122
    * Bremsstrahlung: use more constexpr #3176
    * Pass mapping description by value instead of pointer from simulation stages #3014
    * Add missing inline specifiers for functions defined in header files #3051
    * Remove ZigZag current deposition #2837
    * Fix style issues with particlePusherAcceleration #2781
  - PMacc:
    * Supercell particle counter #2637
    * ForEachIdx::operator(): Use Universal Reference #2881
    * Remove duplicated definition of `BOOST_MPL_LIMIT_VECTOR_SIZE` #2883
    * Cleanup `pmacc/types.hpp` #2927
    * Add pmacc::memory::makeUnique similar to std::make_unique #2949
    * PMacc Vector: C++11 Using #2957
    * Remove pmacc::forward and pmacc::RefWrapper #2963
    * Add const getters to ParticleBox #2941
    * Remove unused pmacc::traits::GetEmptyDefaultConstructibleType #2976
    * Remove pmacc::traits::IsSameType which is no longer used #2979
    * Remove template parameter for initialization method of Pointer and FramePointer #2977
    * Remove pmacc::expressions which is no longer used #2978
    * Remove unused pmacc::IDataSorter #3030
    * Change PMACC_C_STRING to produce a static constexpr member #3050
    * Refactor internals of pmacc::traits::GetUniqueTypeId #3049
    * rename "counterParticles" to "numParticles" #3062
    * Make pmacc::DataSpace conversions explicit #3124
  - plugins:
    * Small update for python visualizers #2882
    * Add namespace "radiation" around code related to radiation plugin #3004
    * Remove unused includes of pthread #3040
    * SpeciesEligibleForSolver for radiation plugin #3061
    * ADIOS: Avoid unsafe temporary strings #2946
  - tools:
    * Update cuda_memtest: CMake CUDA_ROOT Env #2892
    * Update hemera tpl after SLURM update #3123

  - Add pillow as dependency #3180
  - Params: remove `boost::vector<>` usage #2769
  - Use _X syntax in OnceIonized manipulator #2745
  - Add missing const to some GridController getters #3154
- documentation:
  - Containers: Update 0.4.0 #2750
  - Merge 0.4.0 Changelog #2748
  - Update Readme & License: People #2749
  - Add .zenodo.json #2747
  - Fix species.param docu (in all examples too) #2795
  - Fix species.param example doc and grammar #2793
  - Further improve wording in docs #2710
  - MemoryCalculator: fix example output for documentation #2822
  - Manual: Plugin & Particle Sections, Map #2820
  - System: D.A.V.I.D.E #2821
  - License Header: Update 2019 #2845
  - Docs: Memory per Device Spelling #2868
  - CMake 3.11.0+ #2959
  - CUDA 9.0+, GCC 5.1+, Boost 1.65.1+ #2961
  - CMake: CUDA 9.0+ #2965
  - Docs: Update Sphinx #2969
  - CMake: CUDA 9.2-10.1, Boost <= 1.70.0 #2975
  - Badge: Commits Since Release & Good First #2980
  - Update info on maintainers in README.md #2984
  - Fix grammar in all `.profile.example` #2930
  - Docs: Dr.s #3009
  - Fix old file name in radiation doc #3018
  - System: ARIS #3039
  - fix typo in getNode and getDevice #3046
  - Window move point clean up #3045
  - Docs: Cori's KNL Nodes (NERSC) #3043
  - Fix various sphinx issues not related to doxygen #3056
  - Extend the particle merger plugin documentation #3057
  - Fix docs using outdated ManipulateDeriveSpecies #3068
  - Adjust cores per gpu on taurus after multicore update #3071
  - Docs: create conda env for building docs #3074
  - Docs: add missing checkpoint options #3080
  - Remove titan ornl setup and doc #3086
  - Summit: Profile & Templates #3007

  - Update URL to ADIOS #3099
  - License Header: Update 2020 #3138
  - Add PhD thesis reference in radiation plugin #3151
  - Spack: w/o Modules by Default #3182
  - Add a brief description of simulation output to basics #3183
  - Fix a typo in exchange communication tag status output #3141
  - Add a link to PoGit to the docs #3115
  - fix optional install instructions in the Summit profile #3094
  - Update the form factor documentation #3083
  - Docs: Add New References #3072
  - Add information about submit.cfg and submit.tpl files to docs. #3070
  - Fix style (underline length) in profile.rst #2936
  - Profiles: Section Title Length #2934
  - Contributor name typo in LICENSE.md #2880
  - Update modules and memory in gpu_picongpu.profile #2923
  - Add k80_picongpu.profile and k80.tpl #2919
  - Update taurus-tud profiles for the `ml` partition #2903
  - Hypnos: CMake 3.13.4 #2887
  - Docs: Install Blosc #2829
  - Docs: Source Intro Details #2828
  - Taurus Profile: Project #2819
  - Doc: Add System Links #2818
  - remove grep file redirect #2788
  - Correct jupyter widget example #3191
  - fix typo: `UNIT_LENGHT` to `UNIT_LENGTH` #3194
  - Change link to CRP group @ HZDR #2814
- Examples: Unify .cfg #2826
- Remove unused ABSORBER_FADE_IN_STEPS from .param files in examples #2942
- Field absorber test example #2948
- Singularity: Avoid Dotfiles in Home #2981
- Boost: No std::auto_ptr #3012
- Add YeePML to comments for field solver selection #3042
- Add a runtime parameter for window move point #3022
- Ionization: add silicon to pre-defines #3078
- Add 1.cfg to Bremsstrahlung example #3098
- Fix cmake flags for MSVS #3126
- Fix missing override flags #3156
- Fix warning #222-D: floating-point operation result is out of range #3170
- Update alpaka to 0.4.0 and cupla to 0.2.0 #3175

- Slurm update taurus: workdir to chdir #3181
- Adjust profiles for taurus-tud #2990
- Update mallocMC to 2.3.1crp #2893
- Change imread import from scipy.misc to imageio #3192

5.4 0.4.3

Date: 2019-02-14

System Updates and Bug Fixes

This release adds updates and new HPC system templates. Important bug fixes include I/O work-arounds for issues in OpenMPI 2.0-4.0 (mainly with HDF5), guards for particle creation with user-defined profiles, a fixed binomial current smoothing, checks for the number of devices in grid distributions and container (Docker & Singularity) modernizations.

Thanks to Axel Huebl, Alexander Debus, Igor Andriyash, Marco Garten, Sergei Bastrakov, Adam Simpson, Richard Pausch, Juncheng E, Klaus Steiniger, and René Widera for contributions to this release!

5.4.1 Changes to "0.4.2"

Bug Fixes:

- fix particle creation if density <= zero #2831
- fix binomial current interpolation #2838
- Docker & Singularity updates #2847
- OpenMPI: use ROMIO for IO #2841 #2857
- `--gridDist`: verify devices and blocks #2876
- Phase space plugin: unit of colorbar in 2D3V #2878

Misc:

- `ionizer.param`: fix typo in "Aluminium" #2865
- System Template Updates:
  - Add system links #2818
  - Taurus:
    - add project #2819
    - add Power9 V100 nodes #2856
  - add D.A.V.I.D.E (CINECA) #2821
  - add JURECA (JSC) #2869
  - add JUWELS (JSC) #2874
  - Hypnos (HZDR): CMake update #2887
  - Slurm systems: link `stdout` to `simOutput/output` #2839
- Docs:
  - Change link to CRP group @ HZDR #2814
  - `FreeRng.def`: typo in example usage #2825
  - More details on source builds #2828

  - Dependencies: Blosc install #2829
  - Ionization plot title linebreak #2867
- plugins:
  - ADIOS & phase space `-Wterminate` #2817
  - Radiation: update documented options #2842
- Update versions script: containers #2846
- pyflakes: `str/bytes/int` compares #2866
- Travis CI: Fix Spack CMake Install #2879
- Contributor name typo in `LICENSE.md` #2880
- Update mallocMC to 2.3.1crp #2893
- CMake: Honor `_ROOT` Env Hints #2891 #2892 #2893

5.5 0.4.2

Date: 2018-11-19

CPU Plugin Performance

This release fixes a performance regression for energy histograms and phase space plugins on CPU with our OpenMP backend on CPU. At least OpenMP 3.1 is needed to benefit from this. Additionally, several small documentation issues have been fixed and the energy histogram python tool forgot to return the first iteration.

Thanks to Axel Huebl, René Widera, Sebastian Starke, and Marco Garten for contributions to this release!

5.5.1 Changes to "0.4.1"

Bug Fixes:

- Plugin performance regression:
  - Speed of plugins `EnergyHistogram` and `PhaseSpace` on CPU (`omp2b`) #2802
- Tools:
  - Python `EnergyHistogramData`: skip of first iteration #2799

Misc:

- update Alpaka to 0.3.5 to fix #2802
- Docs:
  - CFL Static Assert: new grid.param #2804
  - missing exponent in fieldIonization.rst #2790
  - remove grep file redirect #2788
  - Calorimeter Plugin: Document File Suffix #2800

5.6 0.4.1

Date: 2018-11-06

Minor Bugs and Example Updates

This release fixes minor bugs found after the 0.4.0 release. Some examples were slightly outdated in syntax, the new "probe particle" `EveryNthCell` initialization functor was broken when not used with equal spacing per dimension. In some rare cases, sliding could occur twice in moving window simulations.

Thanks to Axel Huebl, René Widera, Richard Pausch and Andrei Berceanu for contributions to this release!

5.6.1 Changes to "0.4.0"

Bug Fixes:

- PIConGPU:
  - avoid sliding twice in some corner-cases #2774
  - EveryNthCell: broken if not used with same spacing #2768
  - broken compile with particle merging #2753
- Examples:
  - fix outdated derive species #2756
  - remove current deposition in bunch example #2758
  - fix 2D case of single electron init (via density) #2766
- Tools:
  - Python Regex: r Literals #2767
  - `cuda_memtest`: avoid noisy output if NVML is not found #2785

Misc:

- `.param` files: refactor `boost::vector<>` usage #2769
- Docs:
  - Spack: Improve Bootstrap #2773
  - Fix docs for radiation in 2D #2772
  - Containers: Update 0.4.0 #2750
  - Update Readme & License: People #2749
  - Add `.zenodo.json` #2747

5.7 0.4.0

Date: 2018-10-19

CPU Support, Particle Filter, Probes & Merging

This release adds CPU support, making PIConGPU a many-core, single-source, performance portable PIC code for all kinds of supercomputers. We added particle filters to initialization routines and plugins, allowing fine-grained in situ control of physical observables. All particle plugins now support those filters and can be called multiple times with different settings.

Particle probes and more particle initialization manipulators have been added. A particle merging plugin has been added. The Thomas-Fermi model has been improved, allowing to set empirical cut-offs. PIConGPU input and output (plugins) received initial Python bindings for efficient control and analysis.

User input files have been dramatically simplified. For example, creating the PIConGPU binary from input files for GPU or CPU is now as easy as `pic-build -b cuda` or `pic-build -b omp2b` respectively.

Thanks to Axel Huebl, René Widera, Benjamin Worpitz, Sebastian Starke, Marco Garten, Richard Pausch, Alexander Matthes, Sergei Bastrakov, Heiko Burau, Alexander Debus, Ilja Göthel, Sophie Rudat, Jeffrey Kelling, Klaus Steiniger, and Sebastian Hahn for contributing to this release!

5.7.1 Changes to "0.3.0"

User Input Changes:

- (re)move directory `simulation_defines/` #2331
- add new param file `particleFilters.param` #2385
- `components.param`: remove define `ENABLE_CURRENT` #2678
- `laser.param`: refactor Laser Profiles to Functors #2587 #2652
- `visualization.param`: renamed to `png.param` #2530
- `speciesAttributes.param`: format #2087
- `fieldSolver.param`: doxygen, refactored #2534 #2632
- `mallocMC.param`: file doxygen #2594
- `precision.param`: file doxygen #2593
- `memory.param`:
  - `GUARD_SIZE` docs #2591
  - exchange buffer size per species #2290
  - guard size per dimension #2621
- `density.param`:
  - Gaussian density #2214
  - Free density: fix `float_X` #2555
- `ionizer.param`: fixed excess 5p shell entry in gold effective Z #2558
- `seed.param`:
  - renamed to `random.param` #2605
  - expose random number method #2605
- `isaac.param`: doxygen documentation #2260
- `unit.param`:
  - doxygen documentation #2467
  - move conversion units #2457
  - earlier normalized speed of light in `physicalConstants.param` #2663
- `float_X` constants to literals #2625
- refactor particle manipulators #2125
- new tools:
  - `pic-edit`: adjust `.param` files #2219

  - `pic-build`: combine pic-configure and make install #2204
- `pic-configure`:
  - select CPU/GPU backend and architecture with `-b` #2243
  - default backend: CUDA #2248
- `tbg`:
  - `.tpl` no `_profile` suffix #2244
  - refactor `.cfg` files: devices #2543
  - adjust LWFA setup for 8GPUs #2480
- `SliceField` plugin: Option `.frequency` to `.period` #2034
- particle filters:
  - add filter support to phase space plugin #2425
  - multi plugin energy histogram with filter #2424
  - add particle filter to `EnergyParticles` #2386
- Default Inputs: C++11 `using` for `typedef` #2315
- Examples: C++11 `using` for `typedef` #2314
- Python: Parameter Ranges for Param Files (LWFA) #2289
- `FieldTmp`: `SpeciesEligibleForSolver` Traits #2377
- Particle Init Methods: Unify API & Docs #2442
- get species by name #2464
- remove template dimension from current interpolator's #2491
- compile time string #2532

New Features:

- PIC:
  - particle merging #1959
  - check cells needed for stencils #2257
  - exchange buffer size per species #2290
  - push with `currentStep` #2318
  - `InitController`: unphysical particles #2365
  - New Trait: `SpeciesEligibleForSolver` #2364
  - Add upper energy cut-off to ThomasFermi model #2330
  - Particle Pusher: Probe #2371
  - Add lower ion density cut-off to ThomasFermi model #2361
  - CT Factory: `GenerateSolversIfSpeciesEligible` #2380
  - add new param file `particleFilters.param` #2385
  - Probe Particle Usage #2384
  - Add lower electron temperature cut-off to ThomasFermi model #2376
  - new particle filters #2418 #2659 #2660 #2682
  - Derived Attribute: Bound Electron Density #2453
  - get species by name #2464

  - New Laser Profile: Exp. Ramps with Prepulse #2352
  - Manipulator: `UnboundElectronsTimesWeighting` #2398
  - Manipulator: `unary::FreeTotalCellOffset` #2498
  - expose random number method to the user #2605
  - seed generator for RNG #2607
  - FLYlite: initial interface & helper fields #2075
- PMacc:
  - cupla compatible RNG #2226
  - generic `min()` and `max()` implementation #2173
  - Array: store elements without a default constructor #1973
  - add array to hold context variables #1978
  - add `ForEachIdx` #1977
  - add trait `GetNumWorker` #1985
  - add index pool #1958
  - Vector `float1_X` to `float_X` cast #2020
  - extend particle handle #2114
  - add worker config class #2116
  - add interfaces for functor and filter #2117
  - Add complex logarithm to math #2157
  - remove unused file `BitData.hpp` #2174
  - Add Bessel functions to math library #2156
  - Travis: Test PMacc Unit Tests #2207
  - rename CUDA index names in `ConcatListOfFrames` #2235
  - cuSTL `Foreach` with lockstep support #2233
  - Add complex `sin()` and `cos()` functions. #2298
  - Complex `BesselJ0` and `BesselJ1` functions #2161
  - CUDA9 default constructor warnings #2347
  - New Trait: HasIdentifiers #2363
  - RNG with reduced state #2410
  - PMacc RNG 64bit support #2451
  - PhaseSpace: add lockstep support #2454
  - signed and unsigned comparison #2509
  - add a workaround for MSVC bug with capturing `constexpr` #2522
  - compile time string #2532
  - `Vector`: add method `remove<...>()` #2602
  - add support for more cpu alpaka accelerators #2603 #2701
  - Vector `sumOfComponents` #2609
  - `math::CT::max` improvement #2612
- plugins:

  - ADIOS: allow usage with accelerator `omp2b` #2236
  - ISAAC:
    - alpaka support #2268 #2349
    - require version 1.4.0+ #2630
  - `InSituVolumeRenderer`: removed (use ISAAC instead) #2238
  - HDF5: Allow Unphysical Particle Dump #2366
  - `SpeciesEligibleForSolver` Traits #2367
  - PNG:
    - lockstep kernel refactoring `Visualisation.hpp` #2225
    - require PNGwriter version 0.7.0+ #2468
  - `ParticleCalorimeter`:
    - add particle filter #2569
    - fix usage of uninitialized variable #2320
  - Python:
    - Energy Histogram Reader #2209 #2658
    - Phase Space Reader #2334 #2634 #2679
    - Move SliceField Module & add Python3 support #2354 #2718
    - Multi-Iteration Energy Histogram #2508
    - MPL Visualization modules #2484 #2728
    - migrated documentation to Sphinx manual #2172 #2726 #2738
    - shorter python imports for postprocessing tools #2727
    - fix energy histogram deprecation warning #2729
    - `data`: base class for readers #2730
    - `param_parser` for JSON parameter files #2719
- tools:
  - Tool: New Version #2080
  - Changelog & Left-Overs from 0.3.0 #2120
  - TBG: Check Modified Input #2123
  - Hypnos (HZDR) templates:
    - `mpiexec` and `LD_LIBRARY_PATH` #2149
    - K20 restart #2627
    - restart `.tpl` files: new `checkpoints.period` syntax #2650
  - Travis: Enforce PEP8 #2145
  - New Tool: pic-build #2204
  - Docker:
    - `Dockerfile` introduced #2115 #2286
    - `spack clean` & `load` #2208
    - update ISAAC client URL #2565
  - add HZDR cluster `hydra` #2242

  - pic-configure: default backend CUDA #2248
  - New Tool: pic-edit #2219
  - FoilLCT: Plot Densities #2259
  - tbg: Add `-f|--force` #2266
  - Improved the cpuNumaStarter.sh script to support not using all hw threads #2269
  - Removed libm dependency for Intel compiler… #2278
  - CMake: Same Boost Min for Tools #2293
  - HZDR tpl: killall return #2295
  - PMacc: Set CPU Architecture #2296
  - ThermalTest: Flake Dispersion #2297
  - Python: Parameter Ranges for Param Files (LWFA) #2289
  - LWFA: GUI .cfg & Additional Parameters #2336
  - Move mpiInfo to new location #2355
  - bracket test for external libraries includes #2399
  - Clang-Tidy #2303
  - tbg -f: mkdir -p submitAction #2413
  - Fix initial setting of Parameter values #2422
  - Move TBG to bin/ #2537
  - Tools: Move pic-* to bin/ #2539
  - Simpler Python Parameter class #2550

Bug Fixes:

- PIC:
  - fix restart with background fields enabled #2113
  - wrong border with current background field #2326
  - remove usage of pure `float` with `float_X` #2606
  - fix stencil conditions #2613
  - fix that guard size must be one #2614
  - fix dead code #2301
  - fix memory leaks #2669
- PMacc:
  - event system:
    - fix illegal memory access #2151
    - fix possible deadlock in blocking MPI ops #2683
  - cuSTL:
    - missing `#include` in `ForEach` #2406
    - `HostBuffer` 1D Support #2657
  - fix warning concerning forward declarations of `pmacc::detail::Environment` #2489
  - `pmacc::math::Size_t<0>::create()` in Visual Studio #2513
  - fix V100 deadlock #2600

  - fix missing include #2608
  - fix gameOfLife #2700
  - Boost template aliases: fix older CUDA workaround #2706
- plugins:
  - energy fields: fix reduce #2112
  - background fields: fix restart `GUARD` #2139
  - Phase Space:
    * fix weighted particles #2428
    * fix momentum meta information #2651
  - ADIOS:
    * fix 1 particle dumps #2437
    * fix zero size transform writes #2561
    * remove `adios_set_max_buffer_size` #2670
    * require 1.13.1+ #2583
  - IO fields as source #2461
  - ISAAC: fix gcc compile #2680
  - Calorimeter: Validate minEnergy #2512
- tools:
  - fix possible linker error #2107
  - cmakeFlags: Escape Lists #2183
  - splash2txt: C++98 #2136
  - png2gas: C++98 #2162
  - tbg env variables escape \ and & #2262
  - XDMF Scripts: Fix Replacements & Offset #2309
  - pic-configure: cmakeFlags return code #2323
  - tbg: fix wrong quoting of `'` #2419
  - CMake in-source builds: too strict #2407
- `--help` to stdout #2148
- Density: Param Gaussian Density #2214
- Fixed excess 5p shell entry in gold effective Z #2558
- Hypnos: Zlib #2570
- Limit Supported GCC with nvcc 8.0-9.1 #2628
- Syntax Highlighting: Fix RTD Theme #2596
- remove extra typename in documentation of manipulators #2044

Misc:

- new example: Foil (LCT) TNSA #2008
- adjust LWFA setup for 8 GPUs #2480
- `picongpu --version` #2147
- add internal Alpaka & cupla #2179 #2345

- add alpaka dependency #2205 #2328 #2346 #2590 #2501 #2626 #2648 #2684 #2717
- Update mallocMC to `2.3.0crp` #2350 #2629
- cuda_memtest:
  - update #2356 #2724
  - usage on hypnos #2722
- Examples:
  - remove unused loaders #2247
  - update `species.param` #2474
- Bunch: no `precision.param` #2329
- Travis:
  - stages #2341
  - static code analysis #2404
- Visual Studio: ERROR macro defined in `wingdi.h` #2503
- Compile Suite: update plugins #2595
- refactoring:
  - PIC:
    - `const` POD Default Constructor #2300
    - `FieldE`: Fix Unreachable Code Warning #2332
    - Yee solver lockstep refactoring #2027
    - lockstep refactoring of `KernelComputeCurrent` #2025
    - `FieldJ` bash/insert lockstep refactoring #2054
    - lockstep refactoring of `KernelFillGridWithParticles` #2059
    - lockstep refactoring `KernelLaserE` #2056
    - lockstep refactoring of `KernelBinEnergyParticles` #2067
    - remove empty `init()` methods #2082
    - remove `ParticlesBuffer::createParticleBuffer()` #2081
    - remove init method in `FieldE` and `FieldB` #2088
    - move folder `fields/tasks` to libPMacc #2090
    - add `AddExchangeToBorder`, `CopyGuardToExchange` #2091
    - lockstep refactoring of `KernelDeriveParticles` #2097
    - lockstep refactoring of `ThreadCollective` #2101
    - lockstep refactoring of `KernelMoveAndMarkParticles` #2104
    - Esirkepov: reorder code order #2121
    - refactor particle manipulators #2125
    - Restructure Repository Structure #2135
    - lockstep refactoring `KernelManipulateAllParticles` #2140
    - remove all lambda expressions. #2150
    - remove usage of native CUDA function prefix #2153
    - use `nvidia::atomicAdd` instead of our old wrapper #2152

- lockstep refactoring `KernelAbsorbBorder` #2160
- functor interface refactoring #2167
- lockstep kernel refactoring `KernelAddCurrentToEMF` #2170
- lockstep kernel refactoring `KernelComputeSupercells` #2171
- lockstep kernel refactoring `CopySpecies` #2177
- Marriage of PIConGPU and cupla/alpaka #2178
- Ionization: make use of generalized particle creation #2189
- use fast `atomicAllExch` in `KernelFillGridWithParticles` #2230
- enable ionization for CPU backend #2234
- ionization: speedup particle creation #2258
- lockstep kernel refactoring `KernelCellwiseOperation` #2246
- optimize particle shape implementation #2275
- improve speed to calculate number of ppc #2274
- refactor `picongpu::particles::startPosition` #2168
- Particle Pusher: Clean-Up Interface #2359
- create separate plugin for checkpointing #2362
- Start Pos: OnePosition w/o Weighting #2378
- rename filter: `IsHandleValid` -> `All` #2381
- FieldTmp: `SpeciesEligibleForSolver` Traits #2377
- use lower case begin for filter names #2389
- refactor PMacc functor interface #2395
- PIConGPU: C++11 `using` #2402
- refactor particle manipulators/filter/startPosition #2408
- rename `GuardHandlerCallPlugins` #2441
- activate synchrotron for CPU back-end #2284
- `DifferenceToLower/Upper` forward declaration #2478
- Replace usage of M_PI in picongpu with Pi #2492
- remove template dimension from current interpolator's #2491
- Fix issues with name hiding in Particles #2506
- refactor: field solvers #2534
- optimize stride size for update `FieldJ` #2615
- guard size per dimension #2621
- Lasers: `float_X` Constants to Literals #2624
- `float_X`: C++11 Literal #2622
- log: per "device" instead of "GPU" #2662 #2677
- earlier normalized speed of light #2663
- fix GCC 7 fallthrough warning #2665 #2671
- `png.unitless`: static asserts `clang` compatible #2676
- remove define `ENABLE_CURRENT` #2678

- PMacc:
  - refactor `ThreadCollective` #2021
  - refactor reduce #2015
  - lock step kernel `KernelShiftParticles` #2014
  - lockstep refactoring of `KernelCountParticles` #2061
  - lockstep refactoring `KernelFillGapsLastFrame` #2055
  - lockstep refactoring of `KernelFillGaps` #2083
  - lockstep refactoring of `KernelDeleteParticles` #2084
  - lockstep refactoring of `KernelInsertParticles` #2089
  - lockstep refactoring of `KernelBashParticles` #2086
  - call `KernelFillGaps*` from device #2098
  - lockstep refactoring of `KernelSetValue` #2099
  - Game of Life lockstep refactoring #2142
  - `HostDeviceBuffer` rename conflicting type defines #2154
  - use c++11 move semantic in cuSTL #2155
  - lockstep kernel refactoring `SplitIntoListOfFrames` #2163
  - lockstep kernel refactoring `Reduce` #2169
  - enable cuSTL CartBuffer on CPU #2271
  - allow update of a particle handle #2382
  - add support for particle filters #2397
  - RNG: Normal distribution #2415
  - RNG: use non generic place holder #2440
  - extended period syntax #2452
  - Fix buffer cursor dim #2488
  - Get rid of `<sys/time.h>` #2495
  - Add a workaround for `PMACC_STRUCT` to work in Visual Studio #2502
  - Fix type of index in OpenMP-parallelized loop #2505
  - add support for CUDA9 `__shfl_snyc, __ballot_sync` #2348
  - Partially replace compound literals in PMacc #2494
  - fix type cast in `pmacc::exec::KernelStarter::operator()` #2518
  - remove modulo in 1D to ND index transformation #2542
  - Add Missing Namespaces #2579
  - Tests: Add Missing Namespaces #2580
  - refactor RNG method interface #2604
  - eliminate `M_PI` from PMacc #2486
  - remove empty last frame #2649
  - no `throw` in destructors #2666
  - check minimum GCC & Clang versions #2675
- plugins:

    * SliceField Plugin: Option .frequency to .period #2034
    * change notifyFrequency(s) to notifyPeriod #2039
    * lockstep refactoring `KernelEnergyParticles` #2164
    * remove `LiveViewPlugin` #2237
    * Png Plugin: Boost to std Thread #2197
    * lockstep kernel refactoring `KernelRadiationParticles` #2240
    * generic multi plugin #2375
    * add particle filter to `EnergyParticles` #2386
    * PluginController: Eligible Species #2368
    * IO with filtered particles #2403
    * multi plugin energy histogram with filter #2424
    * lockstep kernel refactoring `ParticleCalorimeter` #2291
    * Splash: 1.7.0 #2520
    * multi plugin `ParticleCalorimeter` #2563
    * Radiation Plugin: Namespace #2576
    * Misc Plugins: Namespace #2578
    * EnergyHistogram: Remove Detector Filter #2465
    * ISAAC: unify the usage of period #2455
    * add filter support to phase space plugin #2425
    * Resource Plugin: `fix boost::core::swap` #2721
  - tools:
    * Python: Fix Scripts PEP8 #2028
    * Prepare for Python Modules #2058
    * pic-compile: fix internal typo #2186
    * Tools: All C++11 #2194
    * CMake: Use Imported Targets Zlib, Boost #2193
    * Python Tools: Move lib to / #2217
    * pic-configure: backend #2243
    * tbg: Fix existing-folder error message to stderr #2288
    * Docs: Fix Flake8 Errors #2340
    * Group parameters in LWFA example #2417
    * Python Tools (PS, Histo): Filter Aware #2431
    * Clearer conversion functions for Parameter values between UI scale and internal scale #2432
    * tbg:
      · add content of -o arg to env #2499
      · better handling of missing egetopt error message #2712
  - Format speciesAttributes.param #2087
  - Reduce # photons in Bremsstrahlung example #1979
  - TBG: .tpl no `_profile` suffix #2244

  - Default Inputs: C++11 Using for Typedef #2315
  - Examples: C++11 Using for Typedef #2314
  - LWFA Example: Restore a0=8.0 #2324
  - add support for CUDA9 `__shfl_snyc` #2333
  - add support for CUDA10 #2732
  - Update cuda_memtest: no cuBLAS #2401
  - Examples: Init of Particles per Cell #2412
  - Travis: Image Updates #2435
  - Particle Init Methods: Unify API & Docs #2442
  - PIConGPU use tiny RNG #2447
  - move conversion units to `unit.param` #2457
  - (Re)Move simulation_defines/ #2331
  - CMake: Project Vars & Fix Memtest #2538
  - Refactor .cfg files: devices #2543
  - Free Density: Fix float_X #2555
  - Boost: Format String Version #2566
  - Refactor Laser Profiles to Functors #2587
  - Params: float_X Constants to Literals #2625
- documentation:
  - new subtitle #2734
  - Lockstep Programming Model #2026 #2064
  - `IdxConfig` append documentation #2022
  - `multiMask`: Refactor Documentation #2119
  - `CtxArray` #2390
  - Update openPMD Post-Processing #2322 #2733
  - Checkpoints Backends #2387
  - Plugins:
    * HDF5: fix links, lists & MPI hints #2313 #2711
    * typo in libSplash install #2735
    * External dependencies #2175
    * Multi & CPU #2423
    * Update PS & Energy Histo #2427
    * Memory Complexity #2434
  - Image Particle Calorimeter #2470
  - Update EnergyFields #2559
  - Note on Energy Reduce #2584
  - ADIOS: More Transport & Compression Doc #2640
  - ADIOS Metafile #2633
  - radiation parameters #1986

- CPU Compile #2185
- `pic-configure` help #2191
- Python yt 3.4 #2273
- Namespace `ComputeGridValuePerFrame` #2567
- Document ionization param files for issue #1982 #1983
- Remove ToDo from `ionizationEnergies.param` #1989
- Parameter Order in Manual #1991
- Sphinx:
  * Document Laser Cutoff #2000
  * Move Author Macros #2005
  * PDF Radiation #2184
  * Changelog in Manual #2527
- PBS usage example #2006
- add missing linestyle to ionization plot for documentation #2032
- fix unit ionization rate plot #2033
- fix mathmode issue in ionization plot #2036
- fix spelling of guard #2644
- param: extended description #2041
- fix typos found in param files and associated files #2047
- Link New Coding Style #2074
- Install: Rsync Missing #2079
- Dev Version: 0.4.0-dev #2085
- Fix typo in ADK documentation #2096
- Profile Preparations #2095
- SuperConfig: Header Fix #2108
- Extended $SCRATCH Info #2093
- Doxygen: Fix Headers #2118
- Doxygen: How to Build HTML #2134
- Badge: Docs #2144
- CMake 3.7.0 #2181
- Boost (1.62.0-) 1.65.1 - 1.68.0 #2182 #2707 #2713
- Bash Subshells: `cmd` to $(cmd) #2187
- Boost Transient Deps: date_time, chrono, atomic #2195
- Install Docs: CUDA is optional #2199
- Fix broken links #2200
- PIConGPU Logo: More Platforms #2190
- Repo Structure #2218
- Document KNL GCC -march #2252
- Streamline Install #2256

- Added doxygen documentation for isaac.param file #2260
- License Docs: Update #2282
- Heiko to Former Members #2294
- Added an example profile and tpl file for taurus' KNL #2270
- Profile: Draco (MPCDF) #2308
- $PIC_EXAMPLES #2327
- Profiles for Titan & Taurus #2201
- Taurus:
  - CUDA 8.0.61 #2337
  - Link KNL Profile #2339
  - SCS5 Update #2667
- Move ParaView Profile #2353
- Spack: Own GitHub Org #2358
- LWFA Example: Improve Ranges #2360
- fix spelling mistake in checkpoint #2372
- Spack Install: Clarify #2373 #2720
- Probe Pusher #2379
- CI/Deps: CUDA 8.0 #2420
- Piz Daint (CSCS):
  - Update Profiles #2306 #2655
  - ADIOS Build #2343
  - ADIOS 1.13.0 #2416
  - Update CMake #2436
  - Module Update #2536
  - avoid `pmi_alps` warnings #2581
- Hypnos (HZDR): New Modules #2521 #2661
- Hypnos: PNGwriter 0.6.0 #2166
- Hypnos & Taurus: Profile Examples Per Queue #2249
- Hemera: tbg templates #2723
- Community Map #2445
- License Header: Update 2018 #2448
- Docker: Nvidia-Docker 2.0 #2462 #2557
- Hide Double ToC #2463
- Param Docs: Title Only #2466
- New Developers #2487
- Fix Docs: `FreeTotalCellOffset` Filter #2493
- Stream-line Intro #2519
- Fix HDF5 Release Link #2544
- Minor Formatting #2553

  - PIC Model #2560
  - Doxygen: Publish As Well #2575
  - Limit Filters to Eligible Species #2574
  - Doxygen: Less XML #2641
  - NVCC 8.0 GCC <= 5.3 && 9.0/9.1: GCC <= 5.5 #2639
  - typo: element-wise #2638
  - fieldSolver.param doxygen #2632
  - `memory.param`: `GUARD_SIZE` docs #2591
  - changelog script updated to python3 #2646
- not yet supported on CPU (Alpaka): #2180
  - core:
    - Bremsstrahlung
  - plugins:
    - PositionsParticles
    - ChargeConservation
    - ParticleMerging
    - count per supercell (macro particles)
    - field intensity

5.8 0.3.2

Date: 2018-02-16

Phase Space Momentum, ADIOS One-Particle Dumps & Field Names

This release fixes a bug in the phase space plugin which derived a too-low momentum bin for particles below the typical weighting (and too-high for above it). ADIOS dumps crashed on one-particle dumps and in the name of on-the-fly particle-derived fields species name and field name were in the wrong order. The plugins libSplash (1.6.0) and PNGwriter (0.6.0) need exact versions, later releases will require a newer version of PIConGPU.

5.8.1 Changes to "0.3.1"

Bug Fixes:

- PIConGPU:
  - wrong border with current background field #2326
- libPMacc:
  - cuSTL: missing include in `ForEach` #2406
  - warning concerning forward declarations of `pmacc::detail::Environment` #2489
  - `pmacc::math::Size_t<0>::create()` in Visual Studio #2513
- plugins:
  - phase space plugin: weighted particles' momentum #2428
  - calorimeter: validate `minEnergy` #2512
  - ADIOS:

    - one-particle dumps #2437
    - `FieldTmp`: derived field name #2461
  - exact versions of libSplash 1.6.0 & PNGwriter 0.6.0
- tools:
  - tbg: wrong quoting of `'` #2419
  - CMake: false-positive on in-source build check #2407
  - pic-configure: cmakeFlags return code #2323

Misc:

- Hypnos (HZDR): new modules #2521 #2524

Thanks to Axel Huebl, René Widera, Sergei Bastrakov and Sebastian Hahn for contributing to this release!

5.9 0.3.1

Date: 2017-10-20

Field Energy Plugin, Gaussian Density Profile and Restarts

This release fixes the energy field plugin diagnostics and the "downramp" parameter of the pre-defined Gaussian density profile. Restarts with enabled background fields were fixed. Numerous improvements to our build system were added to deal more gracefully with co-existing system-wide default libraries. A stability issue due to an illegal memory access in the PMacc event system was fixed.

5.9.1 Changes to "0.3.0"

.param file changes:

- `density.param`: in `Gaussian` profile, the parameter `gasSigmaRight` was not properly honored but `gasCenterRight` was taken instead #2214
- `fieldBackground.param`: remove micro meters usage in default file #2138

Bug Fixes:

- PIConGPU:
  - `gasSigmaRight` of `Gaussian` density profile was broken since 0.2.0 release #2214
  - restart with enabled background fields #2113 #2139
  - KHI example: missing constexpr in input #2309
- libPMacc:
  - event system: illegal memory access #2151
- plugins:
  - energy field reduce #2112
- tools:
  - CMake:
    - Boost dependency:
      - same minimal version for tools #2293
      - transient dependenciens: `date_time`, `chrono`, `atomic` #2195
    - use targets of boost & zlib #2193 #2292

    - possible linker error #2107
  - XDMF script: positionOffset for openPMD #2309
  - cmakeFlags: escape lists #2183
  - tbg:
    - `--help` exit with 0 return code #2213
    - env variables: proper handling of \ and & #2262

Misc:

- PIConGPU: `--help` to stdout #2148
- tools: all to C++11 #2194
- documentation:
  - Hypnos .tpl files: remove passing `LD_LIBRARY_PATH` to avoid warning #2149
  - fix plasma frequency and remove German comment #2110
  - remove micro meters usage in default background field #2138
  - README: update links of docs badge #2144

Thanks to Axel Huebl, Richard Pausch and René Widera for contributions to this release!

5.10 0.3.0

Date: 2017-06-16

C++11: Bremsstrahlung, EmZ, Thomas-Fermi, Improved Lasers

This is the first release of PIConGPU requiring C++11. We added a newly developed current solver (EmZ), support for the generation of Bremsstrahlung, Thomas-Fermi Ionization, Laguerre-modes in the Gaussian-Beam laser, in-simulation plane for laser initialization, new plugins for in situ visualization (ISAAC), a generalized particle calorimeter and a GPU resource monitor. Initial support for clang (host and device) has been added and our documentation has been streamlined to use Sphinx from now on.

5.10.1 Changes to "0.2.0"

.param & .unitless file changes:

- use C++11 `constexpr` where possible and update arrays #1799 #1909
- use C++11 `using` instead of `typedef`
- removed `Config` suffix in file names #1965
- `gasConfig` is now `density`
- `speciesDefinition`:
  - simplified `Particles<>` interface #1711 #1942
  - `ionizer< ... >` became a sequence of `ionizers< ... >` #1999
- `radiation`: replace `#defines` with clean C++ #1877 #1930 #1931 #1937

Basic Usage:

We renamed the default tools to create, setup and build a simulation. Please make sure to update your `picongpu.profile` with the latest syntax (e.g. new entries in `PATH`) and use from now on:

- `$PICSRC/createParameterSet` -> `pic-create`
- `$PICSRC/configure` -> `pic-configure`

- `$PICSRC/compile` -> `pic-compile`

See the *Installation* and *Usage* chapters in our new documentation on https://picongpu.readthedocs.io for detailed instructions.

New Features:

- PIConGPU:
  - laser:
    - allow to define the initialization plane #1796
    - add transverse Laguerre-modes to standard Gaussian Beam #1580
  - ionization:
    - Thomas-Fermi impact ionization model #1754 #2003 #2007 #2037 #2046
    - Z_eff, energies, isotope: Ag, He, C, O, Al, Cu #1804 #1860
    - BSI models restructured #2013
    - multiple ionization algorithms can be applied per species, e.g. cut-off barrier suppression ionization (BSI), probabilistic field ionization (ADK) and collisional ionization #1999
  - Add EmZ current deposition solver #1582
  - FieldTmp:
    - Multiple slots #1703
    - Gather support to fill GUARD #2009
  - Particle `StartPosition`: `OnePosition` #1753
  - Add Bremsstrahlung #1504
  - Add kinetic energy algorithm #1744
  - Added species manipulators:
    - `CopyAttribute` #1861
    - `FreeRngImpl` #1866
  - Clang compatible static assert usage #1911
  - Use `PMACC_ASSERT` and `PMACC_VERIFY` #1662
- PMacc:
  - Improve PMacc testsystem #1589
  - Add test for IdProvider #1590
  - Specialize HasFlag and GetFlagType for Particle #1604
  - Add generic atomicAdd #1606
  - Add tests for all RNG generators #1494
  - Extent function `twistVectorFieldAxes<>()` #1568
  - Expression validation/assertion #1578
  - Use PMacc assert and verify #1661
  - GetNComponents: improve error message #1670
  - Define `MakeSeq_t` #1708
  - Add `Array<>` with static size #1725
  - Add shared memory allocator #1726
  - Explicit cast `blockIdx` and `threadIdx` to `dim3` #1742

    - CMake: allow definition of multiple architectures #1729
    - Add trait `FilterByIdentifier` #1859
    - Add CompileTime Accessor: Type #1998
- plugins:
    - HDF5/ADIOS:
        * MacroParticleCounter #1788
        * Restart: Allow disabling of moving window #1668
        * FieldTmp: MidCurrentDensityComponent #1561
    - Radiation:
        * Add pow compile time using c++11 #1653
        * Add radiation form factor for spherical Gaussian charge distribution #1641
    - Calorimeter: generalize (charged & uncharged) #1746
    - PNG: help message if dependency is not compiled #1702
    - Added:
        * In situ: ISAAC Plugin #1474 #1630
        * Resource log plugin #1457
- tools:
    - Add a tpl file for k80 hypnos that automatically restarts #1567
    - Python3 compatibility for plotNumericalHeating #1747
    - Tpl: Variable Profile #1975
    - Plot heating & charge conservation: file export #1637
- Support for clang as host && device compiler #1933

Bug Fixes:

- PIConGPU:
    - 3D3V: missing absorber in z #2042
    - Add missing minus sign wavepacket laser transversal #1722
    - `RatioWeighting` (`DensityWeighting`) manipulator #1759
    - `MovingWindow`: `slide_point` now can be set to zero. #1783
    - `boundElectrons`: non-weighted attribute #1808
    - Verify number of ionization energy levels == proton number #1809
    - Ionization:
        * charge of ionized ions #1844
        * ADK: fix effective principal quantum number `nEff` #2011
    - Particle manipulators: position offset #1852
- PMacc:
    - Avoid CUDA local memory usage of `Particle<>` #1579
    - Event system deadlock on `MPI_Barrier` #1659
    - ICC: `AllCombinations` #1646
    - Device selection: guard valid range #1665

  - `MapTuple`: broken compile with icc #1648
  - Missing '%%' to use ptx special register #1737
  - `ConstVector`: check arguments init full length #1803
  - `ComputeEvent`: cyclic include #1836
  - Add missing `HDINLINE` #1825
  - Remove `BOOST_BIND_NO_PLACEHOLDERS` #1849
  - Remove CUDA native static shared memory #1929
- plugins:
  - Write openPMD meta data without species #1718
  - openPMD: iterationFormat only Basename #1751
  - ADIOS trait for `bool` #1756
  - Adjust `radAmplitude` python module after openPMD changes #1885
  - HDF5/ADIOS: ill-placed helper `#include` #1846
  - `#include`: never inside namespace #1835
- work-around for bug in boost 1.64.0 (odeint) + CUDA NVCC 7.5 & 8.0 #2053 #2076

Misc:

- refactoring:
  - PIConGPU:
    - Switch to C++11 only #1649
    - Begin kernel names with upper case letter #1691
    - Maxwell solver, use curl instance #1714
    - Lehe solver: optimize performance #1715
    - Simplify species definition #1711
    - Add missing `math::` namespace to `tan()` #1740
    - Remove usage of pmacc and boost auto #1743
    - Add missing `typenames` #1741
    - Change ternary if operator to `if` condition #1748
    - Remove usage of `BOOST_AUTO` and `PMACC_AUTO` #1749
    - mallocMC: organize setting #1779
    - `ParticlesBase` allocate member memory #1791
    - `Particle` constructor interface #1792
    - Species can omit a current solver #1794
    - Use constexpr for arrays in `gridConfig.param` #1799
    - Update mallocMC #1798
    - `DataConnector`: `#includes` #1800
    - Improve Esirkepov speed #1797
    - Ionization Methods: Const-Ness #1824
    - Missing/wrong includes #1858
    - Move functor `Manipulate` to separate file #1863

    * Manipulator `FreeImpl` #1815
    * Ionization: clean up params #1855
    * MySimulation: remove particleStorage #1881
    * New `DataConnector` for fields (& species) #1887 #2045
    * Radiation filter functor: remove macros #1877
    * Topic use remove shared keyword #1727
    * Remove define `ENABLE_RADIATION` #1931
    * Optimize `AssignedTrilinearInterpolation` #1936
    * `Particles<>` interface #1942
    * Param/Unitless files: remove "config" suffix #1965
    * Kernels: Refactor Functions to Functors #1669
    * Gamma calculation #1857
    * Include order in defaut loader #1864
    * Remove `ENABLE_ELECTRONS/IONS` #1935
    * Add `Line<>` default constructor #1588
  - PMacc:
    * Particles exchange: avoid message spamming #1581
    * Change minimum CMake version #1591
    * CMake: handle PMacc as separate library #1692
    * ForEach: remove boost preprocessor #1719
    * Refactor `InheritLinearly` #1647
    * Add missing `HDINLINE` prefix #1739
    * Refactor .h files to .hpp files #1785
    * Log: make events own level #1812
    * float to int cast warnings #1819
    * DataSpaceOperations: Simplify Formula #1805
    * DataConnector: Shared Pointer Storage #1801
    * Refactor `MPIReduce` #1888
    * Environment refactoring #1890
    * Refactor `MallocMCBuffer` share #1964
    * Rename `typedefs` inside `ParticleBuffer` #1577
    * Add typedefs for `Host/DeviceBuffer` #1595
    * DeviceBufferIntern: fix shadowed member variable #2051
  - plugins:
    * Source files: remove non-ASCII chars #1684
    * replace old analyzer naming #1924
    * Radiation:
      · Remove Nyquist limit switch #1930
      · Remove precompiler flag for form factor #1937

    - compile-time warning in 2D live plugin #2063
  - tools:
    - Automatically restart from ADIOS output #1882
    - Workflow: rename tools to set up a sim #1971
    - Check if binary `cuda_memtest` exists #1897
  - C++11 constexpr: remove boost macros #1655
  - Cleanup: remove EOL white spaces #1682
  - .cfg files: remove EOL white spaces #1690
  - Style: more EOL #1695
  - Test: remove more EOL white spaces #1685
  - Style: replace all tabs with spaces #1698
  - Pre-compiler spaces #1693
  - Param: Type List Syntax #1709
  - Refactor Density Profiles #1762
  - Bunch Example: Add Single e- Setup #1755
  - Use Travis `TRAVIS_PULL_REQUEST_SLUG` #1773
  - ManipulateDeriveSpecies: Refactor Functors & Tests #1761
  - Source Files: Move to Headers #1781
  - Single Particle Tests: Use Standard MySimulation #1716
  - Replace NULL with C++11 nullptr #1790
- documentation:
  - Wrong comment random->quiet #1633
  - Remove `sm_20` Comments #1664
  - Empty Example & `TBG_macros.cfg` #1724
  - License Header: Update 2017 #1733
  - speciesInitialization: remove extra typename in doc #2044
  - INSTALL.md:
    - List Spack Packages #1764
    - Update Hypnos Example #1807
    - grammar error #1941
  - TBG: Outdated Header #1806
  - Wrong sign of `delta_angle` in radiation observer direction #1811
  - Hypnos: Use CMake 3.7 #1823
  - Piz Daint: Update example environment #2030
  - Doxygen:
    - Warnings Radiation #1840
    - Warnings Ionization #1839
    - Warnings PMacc #1838
    - Warnings Core #1837

    - Floating Docstrings #1856
    - Update `struct.hpp` #1879
    - Update FieldTmp Operations #1789
    - File Comments in Ionization #1842
    - Copyright Header is no Doxygen #1841
  - Sphinx:
    - Introduce Sphinx + Breathe + Doxygen #1843
    - PDF, Link rst/md, png #1944 #1948
    - Examples #1851 #1870 #1878
    - Models, PostProcessing #1921 #1923
    - PMacc Kernel Start #1920
    - Local Build Instructions #1922
    - Python Tutorials #1872
    - Core Param Files #1869
    - Important Classes #1871
    - .md files, tbg, profiles #1883
    - `ForEach` & Identifier #1889
    - References & Citation #1895
    - Slurm #1896 #1952
    - Restructure Install Instructions #1943
    - Start a User Workflows Section #1955
  - ReadTheDocs:
    - Build PDF & EPUB #1947
    - remove linenumbers #1974
  - Changelog & Version 0.2.3 (master) #1847
  - Comments and definition of `radiationObserver` default setup #1829
  - Typos plot radiation tool #1853
  - doc/ -> docs/ #1862
  - Particles Init & Manipulators #1880
  - INSTALL: Remove gimli #1884
  - BibTex: Change ShortHand #1902
  - Rename `slide_point` to `movePoint` #1917
  - Shared memory allocator documenation #1928
  - Add documentation on slurm job control #1945
  - Typos, modules #1949
  - Mention current solver `EmZ` and compile tests #1966
- Remove assert.hpp in radiation plugin #1667
- Checker script for `__global__` keyword #1672
- Compile suite: GCC 4.9.4 chain #1689

- Add TSC and PCS rad form factor shapes #1671
- Add amend option for tee in k80 autorestart tpl #1681
- Test: EOL and suggest solution #1696
- Test: check & remove pre-compiler spaces #1694
- Test: check & remove tabs #1697
- Travis: check PR destination #1732
- Travis: simple style checks #1675
- PositionFilter: remove (virtual) Destructor #1778
- Remove namespace workaround #1640
- Add Bremsstrahlung example #1818
- WarmCopper example: FLYlite benchmark #1821
- Add compile tests for radiation methods #1932
- Add visual studio code files to gitignore #1946
- Remove old QT in situ volume visualization #1735

Thanks to Axel Huebl, René Widera, Alexander Matthes, Richard Pausch, Alexander Grund, Heiko Burau, Marco Garten, Alexander Debus, Erik Zenker, Bifeng Lei and Klaus Steiniger for contributions to this release!

5.11 0.2.5

Date: 2017-05-27

Absorber in z in 3D3V, effective charge in ADK ionization

The absorbing boundary conditions for fields in 3D3V simulations were not enabled in z direction. This caused unintended reflections of electro-magnetic fields in z since the 0.1.0 (beta) release. ADK ionization was fixed to the correct charge state (principal quantum number) which caused wrong ionization rates for all elements but Hydrogen.

5.11.1 Changes to "0.2.5"

Bug Fixes:

- ADK ionization: effective principal quantum number nEff #2011
- 3D3V: missing absorber in z #2042

Misc:

- compile-time warning in 2D live plugin #2063
- DeviceBufferIntern: fix shadowed member variable #2051
- speciesInitialization: remove extra typename in doc #2044

Thanks to Marco Garten, Richard Pausch, René Widera and Axel Huebl for spotting the issues and providing fixes!

5.12 0.2.4

Date: 2017-03-06

Charge of Bound Electrons, openPMD Axis Range, Manipulate by Position

This release fixes a severe bug overestimating the charge of ions when used with the `boundElectrons` attribute for field ionization. For HDF5 & ADIOS output, the openPMD axis annotation for fields in simulations with non-cubic cells or moving window was interchanged. Assigning particle manipulators within a position selection was rounded to the closest supercell (`IfRelativeGlobalPositionImpl`).

5.12.1 Changes to "0.2.3"

Bug Fixes:

- ionization: charge of ions with `boundElectrons` attribute #1844
- particle manipulators: position offset, e.g. in `IfRelativeGlobalPositionImpl` rounded to supercell #1852 #1910
- PMacc:
  - remove `BOOST_BIND_NO_PLACEHOLDERS` #1849
  - add missing `HDINLINE` #1825
  - `ComputeEvent`: cyclic include #1836
- plugins:
  - std includes: never inside namespaces #1835
  - HDF5/ADIOS openPMD:
    - GridSpacing, GlobalOffset #1900
    - ill-places helper includes #1846

Thanks to Axel Huebl, René Widera, Thomas Kluge, Richard Pausch and Rémi Lehe for spotting the issues and providing fixes!

5.13 0.2.3

Date: 2017-02-14

Energy Density, Ionization NaNs and openPMD

This release fixes energy density output, minor openPMD issues, corrects a broken species manipulator to derive density weighted particle distributions, fixes a rounding issue in ionization routines that can cause simulation corruption for very small particle weightings and allows the moving window to start immediately with timestep zero. For ionization input, we now verify that the number of arguments in the input table matches the ion species' proton number.

5.13.1 Changes to "0.2.2"

Bug Fixes:

- openPMD:
  - iterationFormat only basename #1751
  - ADIOS trait for bool #1756
  - boundElectrons: non-weighted attribute #1808
- RatioWeighting (DensityWeighting) manipulator #1759
- MovingWindow: slide_point now can be set to zero #1783
- energy density #1750 #1744 (partial)
- possible NAN momenta in ionization #1817
- `tbg` bash templates were outdated/broken #1831

Misc:

- ConstVector:
  - check arguments init full length #1803
  - float to int cast warnings #1819
- verify number of ionization energy levels == proton number #1809

Thanks to Axel Huebl, René Widera, Richard Pausch, Alexander Debus, Marco Garten, Heiko Burau and Thomas Kluge for spotting the issues and providing fixes!

5.14 0.2.2

Date: 2017-01-04

Laser wavepacket, vacuum openPMD & icc

This release fixes a broken laser profile (wavepacket), allows to use icc as the host compiler, fixes a bug when writing openPMD files in simulations without particle species ("vacuum") and a problem with GPU device selection on shared node usage via `CUDA_VISIBLE_DEVICES`.

5.14.1 Changes to "0.2.1"

Bug Fixes:

- add missing minus sign wavepacket laser transversal #1722
- write openPMD meta data without species #1718
- device selection: guard valid range #1665
- PMacc icc compatibility:
  - `MapTuple` #1648
  - `AllCombinations` #1646

Misc:

- refactor `InheritLinearly` #1647

Thanks to René Widera and Richard Pausch for spotting the issues and providing fixes!

5.15 0.2.1

Date: 2016-11-29

QED synchrotron photon & fix potential deadlock in checkpoints

This releases fixes a potential deadlock encountered during checkpoints and initialization. Furthermore, we forgot to highlight that the 0.2.0 release also included a QED synchrotron emission scheme (based on the review in A. Gonoskov et al., PRE 92, 2015).

5.15.1 Changes to "0.2.0"

Bug Fixes:

- potential event system deadlock init/checkpoints #1659

Thank you to René Widera for spotting & fixing and Heiko Burau for the QED synchrotron photon emission implementation!

5.16 0.2.0 "Beta"

Date: 2016-11-24

Beta release: full multiple species support & openPMD

This release of PIConGPU, providing "beta" status for users, implements full multi-species support for an arbitrary number of particle species and refactors our main I/O to be formatted as openPMD (see http://openPMD.org). Several major features have been implemented and stabilized, highlights include refactored ADIOS support (including checkpoints), a classical radiation reaction pusher (based on the work of M. Vranic/IST), parallel particle-IDs, generalized on-the-fly particle creation, advanced field ionization schemes and unification of plugin and file names.

This is our last C++98 compatible release (for CUDA 5.5-7.0). Upcoming releases will be C++11 only (CUDA 7.5+), which is already supported in this release, too.

Thank you to Axel Huebl, René Widera, Alexander Grund, Richard Pausch, Heiko Burau, Alexander Debus, Marco Garten, Benjamin Worpitz, Erik Zenker, Frank Winkler, Carlchristian Eckert, Stefan Tietze, Benjamin Schneider, Maximilian Knespel and Michael Bussmann for contributions to this release!

5.16.1 Changes to "0.1.0"

Input file changes: the generalized versions of input files are as always in `src/picongpu/include/simulation_defines/`.

.param file changes:

- all `const` parameters are now `BOOST_CONSTEXPR_OR_CONST`
- add pusher with radiation reaction (Reduced Landau Lifshitz) #1216
- add manipulator for setting `boundElectrons<>` attribute #768
- add `PMACC_CONST_VECTOR` for ionization energies #768 #1022
- `ionizationEnergies.param` #865
- `speciesAttributes.param`: add ionization model `ADK` (Ammosov-Delone-Krainov) for lin. pol. and circ. pol cases #922 #1541
- `speciesAttributes.param`: rename `BSI` to `BSIHydrogenLike`, add `BSIStarkShifted` and `BSIEffectiveZ` #1423
- `laserConfig.param`: documentation fixed and clearified #1043 #1232 #1312 #1477

- `speciesAttributes.param`: new required traits for for each attribute #1483
- `species*.param`: refactor species mass/charge definition (relatve to base mass/charge) #948
- `seed.param`: added for random number generator seeds #951
- remove use of native `double` and `float` #984 #991
- `speciesConstants.param`: move magic gamma cutoff value from radition plugin here #713
- remove invalid `typename` #926 #944

.unitless file changes:

- add pusher with radiation reaction (Reduced Landau Lifshitz) #1216
- pusher traits simplified #1515
- fieldSolver: numericalCellType is now a namespace not a class #1319
- remove usage of native `double` and `float` #983 #991
- remove invalid `typename` #926
- add new param file: `synchrotronPhotons.param` #1354
- improve the CFL condition depending on dimension in KHI example #774
- add laserPolynom as option to `componentsConfig.param` #772

tbg: template syntax

Please be aware that templates (`.tpl`) used by `tbg` for job submission changed slightly. Simply use the new system-wise templates from `src/picongpu/submit/`. #695 #1609 #1618

Due to unifications in our command line options (plugins) and multi-species support, please update your `.cfg` files with the new namings. Please visit `doc/TBG_macros.cfg` and our wiki for examples.

New Features:

- description of 2D3V simulations is now scaled to a user-defined "dZ" depth looking like a one-z-cell 3D simulation #249 #1569 #1601
- current interpolation/smoothing added #888
- add synchrotron radiation of photons from QED- and classical spectrum #1354 #1299 #1398
- species attributes:
  - particle ids for tracking #1410
  - self-describing units and dimensionality #1261
  - add trait `GetDensityRatio`, add attribute `densityRatio`
  - current solver is now a optinal for a species #1228
  - interpolation is now a optional attribute for a species #1229
  - particle pusher is now a optional attribute for a species #1226
  - add species shape piecewise biqudratic spline `P4S` #781
- species initialization:
  - add general particle creation module #1353
  - new manipulators to clone electrons from ions #1018
  - add manipulator to change the in cell position after gas creation #947 #959
  - documentation #961
- species pushers:
  - enable the way for substepping particle pushers as RLL

    * add pusher with radiation reaction (Reduced Landau Lifshitz) #1216
    * enable substepping in pushers #1201 #1215 #1339 #1210 #1202 #1221
    * add Runge Kutta solver #1177
    * enable use of macro-particle weighting in pushers #1213
  - support 2D for all pushers #1126
- refactor gas profile definitions #730 #980 #1265
- extend `FieldToParticleInterpolation` to 1D- and 2D-valued fields #1452
- command line options:
  - parameter validation #863
  - support for `--softRestarts <n>` to loop simulations #1305
  - a simulation `--author` can be specified (I/O, etc.) #1296 #1297
  - calling `./picongpu` without arguments triggers `--help` #1294
- FieldTmp:
  - scalar fields renamed #1259 #1387 #1523
  - momentum over component #1481
- new traits:
  - `GetStringProperties` for all solvers and species flags #1514 #1519
  - `MacroWeighted` and `WeightingPower` #1445
- speedup current deposition solver ZigZag #927
- speedup particle operations with collective atomics #1016
- refactor particle update call #1377
- enable 2D for single particle test #1203
- laser implementations:
  - add phase to all laser implementations #708
  - add in-plane polarization to TWTS laser #852
  - refactor specific float use in laser polynom #782
  - refactored TWTS laser #704
- checkpoints: now self-test if any errors occured before them #897
- plugins:
  - add 2D support for SliceFieldPrinter plugin #845
  - notify plugins on particles leaving simulation #1394
  - png: threaded, less memory hungry in 2D3V, with author information #995 #1076 #1086 #1251 #1281 #1292 #1298 #1311 #1464 #1465
  - openPMD support in I/O
    * HDF5 and ADIOS plugin refactored #1427 #1428 #1430 #1478 #1517 #1520 #1522 #1529
    * more helpers added #1321 #1323 #1518
    * both write now in a sub-directory in simOutput: h5/ and bp/ #1530
    * getUnit and getUnitDimension in all fields & attributes #1429
  - ADIOS:

    * prepare particles on host side befor dumping #907
    * speedup with `OpenMP` #908
    * options to control striping & meta file creation #1062
    * update to 1.10.0+ #1063 #1557
    * checkpoints & restarts implemented #679 #828 #900
  - speedup radioation #996
  - add charge conservation plugin #790
  - add calorimeter plugin #1376
  - radiation:
    * ease restart on command line #866
    * output is now openPMD compatible #737 #1053
    * enable compression for hdf5 output #803
    * refactor specific float use #778
    * refactor radiation window function for 2D/3D #799
- tools:
  - add error when trying to compile picongpu with CUDA 7.5 w/o C++11 #1384
  - add tool to load hdf5 radiation data into python #1332
  - add uncrustify tool (format the code) #767
  - live visualisation client: set fps panal always visible #1240
  - tbg:
    * simplify usage of `-p|--project` #1267
    * transfers UNIX-permisions from `*.tpl` to submit.start #1140
  - new charge conservation tools #1102, #1118, #1132, #1178
  - improve heating tool to support unfinished and single simulations #729
  - support for python3 #1134
  - improve graphics of numerical heating tool #742
  - speed up sliceFieldReader.py #1399
- ionization models:
  - add possibility for starting simulation with neutral atoms #768
  - generalize BSI: rename BSI to BSIHydrogenLike, add BSIStarkShifted and BSIEffectiveZ #1423
  - add ADK (Ammosov-Delone-Krainov) for lin. pol. and circ. pol cases #922 #1490 #1541 #1542
  - add Keldysh #1543
  - make use of faster RNG for Monte-Carlo with ionization #1542 #1543
- support radiation + ionization in LWFA example #868
- PMacc:
  - running with synchronized (blocking) kernels now adds more useful output #725
  - add RNGProvider for persistent PRNG states #1236, #1493
  - add `MRG32k3a` RNG generator #1487
  - move readCheckpointMasterFile to PMacc #1498

- unify cuda error printing #1484
- add particle ID provider #1409 #1373
- split off HostDeviceBuffer from GridBuffer #1370
- add a policy to GetKeyFromAlias #1252
- Add border mapping #1133, #1169 #1224
- make cuSTL gather accept CartBuffers and handle pitches #1196
- add reference accessors to complex type #1198
- add more rounding functions #1099
- add conversion operator from `uint3` to `Dataspace` #1145
- add more specializations to `GetMPI_StructAsArray` #1088
- implement cartBuffer conversion for HostBuffer #1092
- add a policy for async communication #1079
- add policies for handling particles in guard cells #1077
- support more types in atomicAddInc and warpBroadcast #1078
- calculate better seeds #1040 #1046
- move MallocMCBuffer to PMacc #1034
- move TypeToPointerPair to PMacc #1033
- add 1D, 2D and 3D linear interpolation cursor #1217 #1448
- add method 'getPluginFromType()' to `PluginConnector` #1393
- math:
  * add `abs`, `asin`, `acos`, `atan`, `log10`, `fmod`, `modf`, `floor` to algorithms::math #837 #1218 #1334 #1362 #1363 #1374 #1473
  * `precisionCast<>` for `PMacc::math::Vector<>` #746
  * support for `boost::mpl::integral_c<>` in `math::CT::Vector<>` #802
  * add complex support #664
- add `cuSTL/MapTo1DNavigator` #940
- add 2D support for cuSTL::algorithm::mpi::Gather #844
- names for exchanges #1511
- rename EnvMemoryInfo to MemoryInfo #1301
- mallocMC (*Memory Allocator for Many Core Architectures*) #640 #747 #903 #977 #1171 #1148
  * remove `HeapDataBox`, `RingDataBox`, `HeapBuffer`, `RingBuffer` #640
  * out of heap memory detection #756
  * support to read mallocMC heap on host side #905
- add multi species support for plugins #794
- add traits:
  * `GetDataBoxType` #728
  * `FilterByFlag` #1219
  * `GetUniqueTypeId` #957 #962
  * `GetDefaultConstructibleType` #1045

    * `GetInitializedInstance` #1447
    * `ResolveAliasFromSpecies` #1451
    * `GetStringProperties` #1507
  - add pointer class for particles `FramePointer` #1055
  - independent sizes on device for `GridBuffer<>::addExchange`
  - `Communicator`: query periodic directions #1510
  - add host side support for kernel index mapper #902
  - optimize size of particle frame for border frames #949
  - add pre-processor macro for struct generation #972
  - add warp collective atomic function #1013
  - speedup particle operations with collective atomics #1014
  - add support to `deselect` unknown attributes in a particle #1524
  - add `boost.test` #1245
    * test for `HostBufferIntern` #1258
    * test for `setValue()` #1268
- add resource monitor #1456
- add MSVC compatibility #816 #821 #931
- `const` box'es return `const pointer` #945
- refactor host/device identifier #946

Bug Fixes:

- laser implementations:
  - make math calls more robust & portable #1160
  - amplitude of Gaussian beam in 2D3V simulations #1052 #1090
  - avoid non zero E-field integral in plane wave #851
  - fix length setup of plane wave laser #881
  - few-cycle wavepacket #875
  - fix documentaion of `a_0` conversation #1043
- FieldTmp Lamor power calculation #1287
- field solver:
  - stricter condition checks #880
  - 2D3V `NoSolver` did not compile #1073
  - more experimental methods for DS #894
  - experimental: possible out of memory access in directional splitting #890
- moving window moved not exactly with c #1273 #1337 #1549
- 2D3V: possible race conditions for very small, non-default super-cells in current deposition (`StrideMapping`) #1405
- experimental: 2D3V zigzag current deposition fix for `v_z != 0` #823
- vaccuum: division by zero in `Quiet` particle start #1527
- remove variable length arrays #932

- gas (density) profiles:
  - gasFreeFormula #988 #899
  - gaussianCloud #807 #1136 #1265
- C++ should catch by const reference #1295
- fix possible underflow on low memory situations #1188
- C++11 compatibility: use `BOOST_STATIC_CONSTEXPR` where possible #1165
- avoid CUDA 6.5 int(bool) cast bug #680
- PMacc detection in CMake #808
- PMacc:
  - EventPool could run out of free events, potential deadlock #1631
  - Particle<>: avoid using CUDA lmem #1579
  - possible deadlock in event system could freeze simulation #1326
  - HostBuffer includes & constructor #1255 #1596
  - const references in Foreach #1593
  - initialize pointers with NULL before cudaMalloc #1180
  - report device properties of correct GPU #1115
  - rename `types.h` to `pmacc_types.hpp` #1367
  - add missing const for getter in GridLayout #1492
  - Cuda event fix to avoid deadlock #1485
  - use Host DataBox in Hostbuffer #1467
  - allow 1D in CommunicatorMPI #1412
  - use better type for params in vector #1223
  - use correct sqrt function for abs(Vector) #1461
  - fix `CMAKE_PREFIX_PATH`s #1391, #1390
  - remove unnecessary floating point ops from reduce #1212
  - set pointers to NULL before calling cudaMalloc #1180
  - do not allocate memory if not gather root #1181
  - load plugins in registered order #1174
  - C++11 compatibility: use `BOOST_STATIC_CONSTEXPR` where possible #1176 #1175
  - fix usage of `boost::result_of` #1151
  - use correct device number #1115
  - fix vector shrink function #1113
  - split EventSystem.hpp into hpp and tpp #1068
  - fix move operators of CartBuffer #1091
  - missing includes in MapTuple #627
  - GoL example: fix offset #1023
  - remove deprecated throw declarations #1000
  - cuSTL:
    - `cudaPitchedPtr.xsize` used wrong #1234

    - gather for supporting static load balancing #1244
    - reduce #936
    - throw exception on cuda error #1235
    - `DeviceBuffer` assign operator #1375, #1308, #1463, #1435, #1401, #1220, #1197
    - Host/DeviceBuffers: Contructors (Pointers) #1094
    - let kernel/runtime/Foreach compute best BlockDim #1309
  - compile with CUDA 7.0 #748
  - device selection with `process exclusive` enabled #757
  - `math::Vector`<> assignment #806
  - `math::Vector`<> copy constructor #872
  - operator[] in `ConstVector` #981
  - empty `AllCombinations<...>` #1230
  - racecondition in `kernelShiftParticles` #1049
  - warning in `FieldManipulator` #1254
  - memory pitch bug in `MultiBox` and `PitchedBox` #1096
  - `math::abs()` for the type `double` #1470
  - invalid kernel call in `kernelSetValue`<> #1407
  - data alignment for kernel parameter #1566
  - `rsqrt` usage on host #967
  - invalid namespace qualifier #968
  - missing namespace prefix #971
- plugins:
  - radiation:
    - enable multi species for radiation plugin #1454
    - compile issues with math in radiation #1552
    - documentation of radiation observer setup #1422
    - gamma filter in radiation plugin #1421
    - improve vector type name encapsuling #998
    - saveguard restart #716
  - CUDA 7.0+ warning in `PhaseSpace` #750
  - racecondition in `ConcatListOfFrames` #1278
  - illegal memory acces in `Visualisation` #1526
  - HDF5 restart: particle offset overflow fixed #721
- tools:
  - mpiInfo: add missing include #786
  - actually exit when pression no in compilesuite #1411
  - fix incorrect mangling of params #1385
  - remove deprecated throw declarations #1003
  - make tool python3 compatible #1416

  - trace generating tool #1264
  - png2gas memory leak fixed #1222
  - tbg:
    * quoting interpretation #801
    * variable assignments stay in `.start` files #695 #1609
    * multiple variable use in one line possible #699 #1610
    * failing assignments at template evaluation time keep vars undefined #1611
  - heating tool supports multi species #729
  - fix numerical heating tool normalization #825
  - fix logic behind fill color of numerical heating tool #779
- libSplash minimum version check #1284

Misc:

- 2D3V simulations are now honoring the cell "depth" in z to make density interpretations easier #1569
- update documentation for dependencies and installation #1556, 1557, #1559, #1127
- refactor usage of several math functions #1462, #1468
- FieldJ interface clear() replaced with an explicit assign(x) #1335
- templates for known systems updated:
  - renaming directories into "cluster-insitutition"
  - tbg copies cmakeFlags now #1101
  - tbg aborts if mkdir fails #797
  - `*tpl` & `*.profile.example` files updated
  - system updates: #937 #1266 #1297 #1329 #1364 #1426 #1512 #1443 #1493
    * Lawrencium (LBNL)
    * Titan/Rhea (ORNL)
    * Piz Daint (CSCS)
    * Taurus (TUD) #1081 #1130 #1114 #1116 #1111 #1137
- replace deprecated CUDA calls #758
- remove support for CUDA devices with `sm_10`, `sm_11`, `sm_12` and `sm_13` #813
- remove unused/unsupported/broken plugins #773 843
  - IntensityPlugin, LiveViewPlugin(2D), SumCurrents, divJ #843
- refactor `value_identifier` #964
- remove native type `double` and `float` #985 #990
- remove `__startAtomicTransaction()` #1233
- remove `__syncthreads()` after shared memory allocation #1082
- refactor `ParticleBox` interface #1243
- rotating root in `GatherSlice` (reduce load of master node) #992
- reduce `GatherSlice` memory footprint #1282
- remove `None` type of ionize, pusher #1238 #1227
- remove math function implementations from `Vector.hpp`

- remove unused defines #921
- remove deprecated thow declaration #918
- remove invalid `typename` #917 #933
- rename particle algorithms from `...clone...` to `...derive...` #1525
- remove math functions from Vector.hpp #1472
- raditation plugin remove `unint` with `uint32_t` #1007
- GoL example: CMake modernized #1138
- INSTALL.md
  - moved from `/doc/` to `/`
  - now in root of the repo #1521
  - add environment variable `$PICHOME` #1162
  - more portable #1164
  - arch linux instructions #1065
- refactor ionization towards independence from `Particle` class #874
- update submit templates for hypnos #860 #861 #862
- doxygen config and code modernized #1371 #1388
- cleanup of stdlib includes #1342 #1346 #1347 #1348 #1368 #1389
- boost 1.60.0 only builds in C++11 mode #1315 #1324 #1325
- update minimal CMake version to 3.1.0 #1289
- simplify HostMemAssigner #1320
- add asserts to cuSTL containers #1248
- rename TwistVectorAxes -> TwistComponents (cuSTL) #893
- add more robust namespace qualifiers #839 #969 #847 #974
- cleanup code #885 #814 #815 #915 #920 #1027 #1011 #1009
- correct spelling #934 #938 #941
- add compile test for ALL pushers #1205
- tools:
  - adjust executable rights and shebang #1110 #1107 #1104 #1085 #1143
  - live visualization client added #681 #835 #1408
- CMake
  - modernized #1139
  - only allow out-of-source builds #1119
  - cleanup score-p section #1413
  - add `OpenMP` support #904
- shipped third party updates:
  - restructured #717
  - `cuda_memtest` #770 #1159
  - CMake modules #1087 #1310 #1533
- removed several `-Wshadow` warnings #1039 #1061 #1070 #1071

5.17 0.1.0

Date: 2015-05-21

This is version `0.1.0` of PIConGPU, a *pre-beta* version.

Initial field ionization support was added, including the first model for BSI. The code-base was substantially hardened, fixing several minor and major issues. Especially, several restart related issues, an issue with 2D3V zigzack current calculation and a memory issue with Jetson TK1 boards were fixed. A work-around for a critical CUDA 6.5 compiler bug was applied to all affected parts of the code.

5.17.1 Changes to "Open Beta RC6"

.param file changes: See full syntax for each file at https://github.com/ComputationalRadiationPhysics/picongpu/tree/0.1.0/src/picon

- `componentsConfig.param` & `gasConfig.param` fix typo `gasHomogeneous` #577
- `physicalConstants.param`: new variable `GAMMA_THRESH` #669
- `speciesAttributes.param`: new identifier `boundElectrons` and new aliases `ionizer`, `atomicNumbers`
- `ionizationEnergies.param`, `ionizerConfig.param`: added

.unitless file changes: See full syntax for each file at https://github.com/ComputationalRadiationPhysics/picongpu/tree/0.1.0/src/pico

- `gasConfig.unitless`: typo in `gasHomogeneous` #577
- `speciesAttributes.unitless`: new unit for `boundElectrons` identifier
- `speciesDefinition.unitless`: new traits `GetCharge`, `GetMass`, `GetChargeState` and added `ionizers`
- `ionizerConfig.unitless`: added

New Features:

- initial support for field ionization:
  - basic framework and BSI #595
  - attribute (constant flag) for proton and neutron number #687 #731
  - attribute `boundElectrons` #706
- tools:
  - python scripts:
    - new reader for `SliceFieldPrinter` plugin #578
    - new analyzer tool for numerical heating #672 #692
  - `cuda_memtest`:
    - 32bit host system support (Jetson TK1) #583
    - works without `nvidia-smi`, `grep` or `gawk` - optional with NVML for GPU serial number detection (Jetson TK1) #626
  - `splash2txt`:
    - removed build option `S2T_RELEASE` and uses `CMAKE_BUILD_TYPE` #591
  - `tbg`:
    - allows for defaults for `-s`, `-t`, `-c` via env vars #613 #622
  - 3D live visualization: `server` tool that collects `clients` and simulations was published #641
- new/updated particle traits and attributes:

  - `getCharge`, `getMass` #596
  - attributes are now automatically initialized to their generic defaults #607 #615
- PMacc:
  - machine-dependent `UInt` vector class is now split in explicit `UInt32` and `UInt64` classes #665
  - nvidia random number generators (RNG) refactored #711
- plugins:
  - background fields do now affect plugins/outputs #600
  - `Radiation` uses/requires HDF5 output #419 #610 #628 #646 #716
  - `SliceFieldPrinter` supports `FieldJ`, output in one file, updated command-line syntax #548
  - `CountParticles`, `EnergyFields`, `EnergyParticles` support restarts without overwriting their previous output #636 #703

Bug Fixes:

- CUDA 6.5: `int(bool)` casts were broken (affects plugins `BinEnergyParticles`, `PhaseSpace` and might had an effect on methods of the basic PIC cycle) #570 #651 #656 #657 #678 #680
- the ZigZag current solver was broken for 2D3V if non-zero momentum-components in z direction were used (e.g. warm plasmas or purely transversal KHI) #823
- host-device-shared memory (SoC) support was broken (Jetson TK1) #633
- boost 1.56.0+ support via `Resolve<T>` trait #588 #593 #594
- potential race condition in field update and pusher #604
- using `--gridDist` could cause a segfault when adding additional arguments, e.g., in 2D3V setups #638
- `MessageHeader` (used in `png` and 2D live visualization) leaked memory #683
- restarts with HDF5:
  - static load-balancing via `--gridDist` in y-direction was broken #639
  - parallel setups with particle-empty GPUs hung with HDF5 #609 #611 #642
  - 2D3V field reads were broken (each field's z-component was not initialized with the checkpointed values again, e.g., `B_z`) #688 #689
  - loading more than 4 billion global particles was potentially broken #721
- plugins:
  - `Visualization` (png & 2D live sim) memory bug in double precision runs #621
  - `ADIOS`
    - storing more than 4 billion particles was broken #666
    - default of `adios.aggregators` was broken (now = MPI_Size) #662
    - parallel setups with particle-empty GPUs did hang #661
  - `HDF5`/`ADIOS` output of grid-mapped particle energy for non-relativistic particles was zero #669
- PMacc:
  - CMake: path detection could fail #796 #808
  - `DeviceBuffer<*,DIM3>::getPointer()` was broken (does not affect PIConGPU) #647
  - empty super-cell memory foot print reduced #648
  - `float2int` return type should be int #623
  - CUDA 7:

    * cuSTL prefixed templates with _ are not allowed; usage of static dim member #630
    * explicit call to `template`-ed `operator()` to avoid waring #750
    * `EnvironmentController` caused a warning about `extendend friend syntax` #644
  - multi-GPU nodes might fail to start up when not using `default` compute mode with CUDA 7 drivers #643

Misc:

- HDF5 support requires libSplash 1.2.4+ #642 #715
- various code clean-up for MSVC #563 #564 #566 #624 #625
- plugins:
  - removed `LineSliceFields` #590
  - `png` plugin write speedup 2.3x by increasing file size about 12% #698
- updated contribution guidelines, install, cfg examples #601 #598 #617 #620 #673 #700 #714
- updated module examples and cfg files for:
  - lawrencium (LBL) #612
  - titan (ORNL) #618
  - hypnos (HZDR) #670
- an `Empty` example was added, which defaults to the setup given by all `.param` files in default mode (a standard PIC cycle without lasers nor particles), see `src/picongpu/include/simulation_defines/` #634
- some source files had wrong file permissions #668

5.18 Open Beta RC6

Date: 2014-11-25

This is the 6th release candidate, a *pre-beta* version.

Initial "multiple species" support was added for flexible particles, but is yet still limited to two species. The checkpoint system was refactored and unified, also incooperating extreme high file I/O bandwidth with ADIOS 1.7+ support. The JetsonTK1 development kit (32bit ARM host side) is now experimentally supported by PMacc/PIConGPU. The *ZigZag* current deposition scheme was implemented providing 40% to 50% speedup over our optimized Esirkepov implementation.

5.18.1 Changes to "Open Beta RC5"

.param file changes:

- Restructured file output control (HDF5/ADIOS), new `fileOutput.param` #495
- `componentsConfig.param`: particle pushers and current solvers moved to new files:
  - `species.param`: general definitions to change all species at once (pusher, current solver)
  - `pusherConfig.param`: special tweaks for individual particle pushers, forward declarations restructured
  - `particleConfig.param`: shapes moved to `species.param`, still defines initial momentum/temperature
  - `speciesAttributes.param`: defines *unique* attributes that can be used across all particle species

  - `speciesDefinition.param`: finally, assign common attributes from `speciesAttributes.param` and methods from `species.param` to define individual species, also defines a general compile time "list" of all available species
- `currentConfig.param`: removed (contained only forward declarations)
- `particleDefinition.param`: removed, now in `speciesAttributes.param`
- `laserConfig.param`: new polarization/focus sections for plane wave and wave-packet: `git diff --ignore-space-change beta-rc5..beta-rc6 src/picongpu/include/simulation_defines/param/laserConfig.param`
- `memory.param`: remove `TILE_` globals and define general `SuperCellSize` and `MappingDesc` instead #435

.unitless file changes:

- `fileOutput.unitless`: restructured and moved to `fileOutput.param`
- `checkpoint.unitless`: removed some includes
- `currentConfig.unitless`: removed
- `gasConfig.unitless`: calculate 3D gas density (per volume) and 2D surface charge density (per area) #445
- `gridConfig.unitless`: include changed
- `laserConfig.unitless`: added ellipsoid for wave packet
- `physicalConstatns.unitless`: `GAS_DENSITY_NORMED` fixed for 2D #445
- `pusherConfig.unitless`: restructured, according to `pusherConfig.param`
- `memory.unitless`: removed #435
- `particleDefinition.unitless`: removed
- `speciesAttributes.unitless`: added, contains traits to access species attributes (e.g., position)
- `speciesDefinition.unitless`: added, contains traits to access quasi-fixed attributes (e.g., charge/mass)

New Features:

- ZigZag current deposition scheme #436 #476
- initial multi/generic particle species support #457 #474 #516
- plugins
  - BinEnergy supports clean restarts without loosing old files #540
  - phase space now works in 2D3V, with arbitrary super cells and with multiple species #463 #470 #480
  - radiation: 2D support #527 #530
- tools
  - splash2txt now supports ADIOS files #531 #545
- plane wave & wave packet lasers support user-defined polarization #534 #535
- wave packet lasers can be ellipses #434 #446
- central restart file to store available checkpoints #455
- PMacc
  - added `math::erf` #525
  - experimental 32bit host-side support (JetsonTK1 dev kits) #571
  - `CT::Vector` refactored and new methods added #473
  - cuSTL: better 2D container support #461

Bug Fixes:

- esirkepov + CIC current deposition could cause a deadlock in some situations #475
- initialization for `kernelSetDrift` was broken (traversal of frame lists, CUDA 5.5+) #538 #539
- the particleToField deposition (e.g. in FieldTmp solvers for analysis) forgot a small fraction of the particle #559
- PMacc
  - no `static` keyword for non-storage class functions/members (CUDA 6.5+) #483 #484
  - fix a game-of-life compile error #550
  - ParticleBox `setAsLastFrame`/`setAsFirstFrame` race condition (PIConGPU was not affected) #514
- tools
  - tbg caused errors on empty variables, tabs, ampersands, comments #485 #488 #528 #529
- dt/CFL ratio in stdout corrected #512
- 2D live view: fix out-of-mem access #439 #452

Misc:

- updated module examples and cfg files for:
  - hypnos (HZDR) #573 #575
  - taurus (ZIH/TUDD) #558
  - titan (ORNL) #489 #490 #492
- Esirkepov register usage (stack frames) reduced #533
- plugins
  - EnergyFields output refactored and clarified #447 #502
  - warnings fixed #479
  - ADIOS
    - upgraded to 1.7+ support #450 #494
    - meta attributes synchronized with HDF5 output #499
- tools
  - splash2txt updates
    - requires libSplash 1.2.3+ #565
    - handle exceptions more transparently #556
    - fix listing of data sets #549 #555
    - fix warnings #553
  - BinEnergyPlot: refactored #542
  - memtest: warnings fixed #521
  - pic2xdmf: refactor XDMF output format #503 #504 #505 #506 #507 #508 #509
  - paraview config updated for hypnos #493
- compile suite
  - reduce verbosity #467
  - remove virtual machine and add access-control list #456 #465
- upgraded to ADIOS 1.7+ support #450 #494
- boost 1.55.0 / nvcc <6.5 work around only applied for affected versions #560

- `boost::mkdir` is now used where necessary to increase portability #460
- PMacc
  - `ForEach` refactored #427
  - plugins: `notify()` is now called *before* `checkpoint()` and a getter method was added to retrieve the last call's time step #541
  - `DomainInfo` and `SubGrid` refactored and redefined #416 #537
  - event system overhead reduced by 3-5% #536
  - warnings fixed #487 #515
  - cudaSetDeviceFlags: uses `cudaDeviceScheduleSpin` now #481 #482
  - `__delete` makro used more consequently #443
  - static asserts refactored and messages added #437
- coding style / white space cleanups #520 #522 #519
- git / GitHub / documentation
  - pyc (compiled python files) are now ignored #526
  - pull requests: description is mandatory #524
- mallocMC cmake `find_package` module added #468

5.19 Open Beta RC5

Date: 2014-06-04

This is the 5th release candidate, a *pre-beta* version.

We rebuild our complete plugin and restart scheme, most of these changes are not backwards compatible and you will have to upgrade to libSplash 1.2+ for HDF5 output (this just means: you can not restart from a beta-rc4 checkpoint with this release).

HDF5 output with libSplash does not contain *ghost/guard* data any more. These information are just necessary for checkpoints (which are now separated from the regular output).

5.19.1 Changes to "Open Beta RC4"

.param file changes:

- Added selection of optional window functions in `radiationConfig.param` #286
- Added more window functions in `radiationConfig.param` #320
- removed double `#define __COHERENTINCOHERENTWEIGHTING__ 1` in some examples `radiationConfig.param` #323
- new file: `seed.param` allows to vary the starting conditions of "identical" runs #353
- Updated a huge amount of `.param` files to remove outdated comments #384
- Update `gasConfig.param/gasConfig.unitless` and doc string in `componentsConfig.param` with new gasFromHdf5 profile #280

.unitless file changes:

- update `fileOutput.unitless` and add new file `checkpoints.unitless` #387
- update `fieldSolver.unitless` #314
- Update `radiationConfig.unitless`: adjust to new supercell size naming #394

- Corrected CFL criteria (to be less conservative) in `gridConfig.unitless` #371

New Features:

- Radiation plugin: add optional window functions to reduce ringing effects caused by sharp boundaries #286 #323 #320
- load gas profiles from png #280
- restart mechanism rebuild #326 #375 #358 #387 #376 #417
- new unified naming scheme for domain and window sizes/offsets #128 #334 #396 #403 #413 #421
- base seed for binary idential simulations now exposed in seed.param #351 #353
- particle kernels without "early returns" #359 #360
- lowered memory foot print during shiftParticles #367
- ShiftCoordinateSystem refactored #414
- tools:
  - tbg warns about broken line continuations in tpl files #259
  - new CMake modules for: ADIOS, libSplash, PNGwriter #271 #304 #307 #308 #406
  - pic2xdmf
    - supports information tags #290 #294
    - one xdmf for grids and one for particles #318 #345
  - Vampir and Score-P support updated/added #293 #291 #399 #422
  - ParaView remote server description for Hypnos (HZDR) added #355 #397
- plugins
  - former name: "modules" #283
  - completely refactored #287 #336 #342 #344
  - restart capabilites added (partially) #315 #326 #425
  - new 2D phase space analysis added (for 3D sims and one species at a time) #347 #364 #391 #407
  - libSplash 1.2+ upgrade (incompatible output to previous versions) #388 #402
- PMacc
  - new Environment class provides all singletons #254 #276 #404 #405
  - new particle traits, methods and flags #279 #306 #311 #314 #312
  - cuSTL ForEach on 1-3D data sets #335
  - cuSTL twistVectorAxes refactored #370
  - NumberOfExchanges replaced numberOfNeighbors implementation #362
  - new math functions: tan, float2int_rd (host) #374 #410
  - CT::Vector now supports ::shrink #392

Bug fixes:

- CUDA 5.5 and 6.0 support was broken #401
- command line argument parser messages were broken #281 #270 #309
- avoid deadlock in computeCurrent, remove early returns #359
- particles that move in the absorbing GUARD are now cleaned up #363
- CFL criteria fixed (the old one was too conservative) #165 #371 #379

- non-GPU-aware (old-stable) MPI versions could malform host-side pinned/page-locked buffers for subsequent cudaMalloc/cudaFree calls (core routines not affected) #438
- ADIOS
  - particle output was broken #296
  - CMake build was broken #260 #268
- libSplash
  - output performance drastically improved #297
- PMacc
  - GameOfLife example was broken #295
  - log compile broken for high log level #372
  - global reduce did not work for references/const #448
  - cuSTL assign was broken for big data sets #431
  - cuSTL reduce minor memory leak fixed #433
- compile suite updated and messages escaped #301 #385
- plugins
  - BinEnergyParticles header corrected #317 #319
  - PNG undefined buffer values fixed #339
  - PNG in 2D did not ignore invalid slides #432
- examples
  - Kelvin-Helmholtz example box size corrected #352
  - Bunch/SingleParticleRadiationWithLaser observation angle fixed #424

Misc:

- more generic 2 vs 3D algorithms #255
- experimental PGI support removed #257
- gcc 4.3 support dropped #264
- various gcc warnings fixed #266 #284
- CMake 3.8.12-2 warnings fixed #366
- picongpu.profile example added for
  - Titan (ORNL) #263
  - Hypnos (HZDR) #415
- documentation updated #275 #337 #338 #357 #409
- wiki started: plugins, developer hints, simulation control, examples #288 #321 #328
- particle interfaces clened up #278
- ParticleToGrid kernels refactored #329
- slide log is now part of the SIMULATION_STATE level #354
- additional NGP current implementation removed #429
- PMacc
  - GameOfLife example documented #305
  - compile time vector refactored #349

  - shortened compile time template error messages #277
  - cuSTL inline documentation added #365
  - compile time operators and ForEach refactored #380
  - TVec removed in preference of CT::Vector #394
- new developers added #331 #373
- Attribution text updated and BibTex added #428

5.20 Open Beta RC4

Date: 2014-03-07

This is the 4th release candidate, a *pre-beta* version.

5.20.1 Changes to "Open Beta RC3"

.param file changes:

- Removed unnesseary includes #234 from: `observer.hpp`, `physicalConstants.param`, `visColorScales.param`, `visualization.param`, `particleConfig.param`, `gasConfig.param`, `fieldBackground.param`, `particleDefinition.param` see the lines that should be removed in #234
- Renamed `observer.hpp` -> `radiationObserver.param` #237 #241 Changed variable name `N_theta` to `N_observer` https://github.com/ComputationalRadiationPhysics/picongpu/commit/9e487ec30ade10ece44fc19fd7a815b8dfe5
9
- Added background FieldJ (current) capability #245 Add the following lines to your `fieldBackground.param`: https://github.com/ComputationalRadiationPhysics/picongpu/commit/7b22f37c6a58250d6623cfbc821c4f996145aad9
1

New Features:

- 2D support for basic PIC cycle #212
- hdf5 output xdmf meta description added: ParaView/VisIt support #219
- background current (FieldJ) can be added now #245

Bug fixes:

- beta-rc3 was broken for some clusters due to an init bug #239
- examples/WeibelTransverse 4 GPU example was broken #221
- smooth script was broken for 1D fields #223
- configure non-existing path did not throw an error #229
- compile time vector "max" was broken #224
- cuda_memtest did throw false negatives on hypnos #231 #236
- plugin "png" did not compile for missing freetype #248

Misc:

- documentation updates
  - radiation post processing scripts #222
  - more meta data in hdf5 output #216
  - tbg help extended and warnings to errors #226
  - doc/PARTICIPATE.md is now GitHub's CONTRIBUTING.md #247 #252

  - slurm interactive queue one-liner added #250
  - developers updated #251
- clean up / refactoring
  - cell_size -> cellSize #227
  - typeCast -> precisionCast #228
  - param file includes (see above for details) #234
  - DataConnector interface redesign #218 #232
  - Esirkepov implementation “paper-like” #238

5.21 Open Beta RC3

Date: 2014-02-14

This is the third release candidate, a *pre-beta* version.

5.21.1 Changes to “Open Beta RC2”

.param and .cfg file changes:

- `componentsConfig.param`:
  - remove simDim defines #134 #137 (example how to update your existing `componentsConfig.param`, see https://github.com/ComputationalRadiationPhysics/picongpu/commit/af1f20790ad2aa15e6fc2c9a51d8c870437a5fb
- `dimension.param`: new file with simDim setting #134
  - only add this file to your example/test/config if you want to change it from the default value (3D)
- `fieldConfig.param`: renamed to `fieldSolver.param` #131
- `fieldBackground.param`: new file to add external background fields #131
- cfg files cleaned up #153 #193

New Features:

- background fields for E and B #131
- write parallel hdf5 with libSplash 1.1 #141 #151 #156 #191 #196
- new plugins
  - ADIOS output support #179 #196
  - makroParticleCounter/PerSuperCell #163
- cuda_memtest can check mapped memory now #173
- EnergyDensity works for 2-3D now #175
- new type floatD_X shall be used for position types (2-3D) #184
- PMacc
  - new functors for multiplications and substractions #135
  - opened more interfaces to old functors #197
  - MappedMemoryBuffer added #169 #182
  - unary transformations can be performed on DataBox’es now, allowing for non-commutative operations in reduces #204

Bug fixes:

- PMacc
  - GridBuffer could deadlock if called uninitialized #149
  - TaskSetValue was broken for all arrays with x-size != n*256 #174
  - CUDA 6.0 runs crashed during cudaSetDeviceFlags #200
  - extern shared mem could not be used with templated types #199
- tbg
  - clearify error message if the tpl file does not exist #130
- HDF5Writer did not write ions any more #188
- return type of failing Slurm runs fixed #198 #205
- particles in-cell position fixed with cleaner algorithm #209

Misc:

- documentation improved for
  - cuSTL #116
  - gasConfig.param describe slopes better (no syntax changes) #126
  - agreed on coding guide lines #155 #161 #140
  - example documentation started #160 #162 #176
  - taurus (slurm based HPC cluster) updates #206
- IDE: ignore Code::Blocks files #125
- Esirkepov performance improvement by 30% #139
- MySimulation asserts refactored for nD #187
- Fields.def with field forward declarations added, refactored to provide common ValueType #178
- icc warnings in cuda_memcheck fixed #210
- PMacc
  - refactored math::vector to play with DataSpace #138 #147
  - addLicense script updated #167
  - MPI_CHECK writes to stderr now #168
  - TVec from/to CT::Int conversion #185
  - PositionFilter works for 2-3D now #189 #207
  - DeviceBuffer cudaPitchedPtr handling clean up #186
  - DataBoxDim1Access refactored #202

5.22 Open Beta RC2

Date: 2013-11-27

This is the second release candidate, a *pre-beta* version.

5.22.1 Changes to "Open Beta RC1"

.param file changes:

- `gasConfig.param`:
  - add gasFreeFormula #96 (example how to update your existing `gasConfig.param`, see https://github.com/ComputationalRadiationPhysics/picongpu/pull/96/files#diff-1)
  - add inner radius to gasSphereFlanks #66 (example how to update your existing `gasConfig.param`, see https://github.com/ComputationalRadiationPhysics/picongpu/pull/66/files#diff-0)

New Features:

- A change log was introduced for master releases #93
- new gas profile "gasFreeFormula" for user defined profiles #96
- CMake (config) #79
  - checks for minimal required versions of dependent libraries #92
  - checks for libSplash version #85
  - update to v2.8.5+ #52
  - implicit plugin selection: enabled if found #52
  - throw more warnings #37
  - experimental support for icc 12.1 and PGI 13.6 #37
- PMacc
  - full rewrite of the way we build particle frames # 86
  - cuSTL: ForEach works on host 1D and 2D data now #44
  - math::pow added #54
  - compile time ForEach added #50
- libSplash
  - dependency upgraded to beta (v1.0) release #80
  - type traits for types PIConGPU - libSplash #69
  - splash2txt update to beta interfaces #83
- new particle to grid routines calculating the Larmor energy #68
- dumping multiple FieldTmp to hdf5 now possible #50
- new config for SLURM batch system (taurus) #39

Bug fixes:

- PMacc
  - cuSTL
    - assign was broken for deviceBuffers #103
    - lambda expressions were broken #42 #46 #100
    - icc support was broken #100 #101
    - views were broken #62
  - InheritGenerator and deselect: icc fix #101
  - VampirTrace (CUPTI) support: cudaDeviceReset added #90
  - GameOfLife example fixed #53 #55

  - warnings in __cudaKernel fixed #51
- picongpu
  - removed all non-ascii chars from job scripts #95 #98
  - CMake
    - keep ptx code was broken #82
    - PGI: string compare broken #75
    - MPI: some libs require to load the C++ dependencies, too #64
    - removed deprecated variables #52
    - Threads: find package was missing #34
  - various libSplash bugs #78 #80 #84
  - current calculation speedup was broken #72
  - Cell2Particle functor missed to provide static methods #49
- tools
  - compile: script uses -q now implicit for parallel (-j N) tests
  - plotDensity: update to new binary format #47
- libraries
  - boost 1.55 work around, see trac #9392 (nvcc #391854)

Misc:

- new reference: SC13 paper, Gordon Bell Finals #106
- new flavoured logo for alpha
- Compile Suite: GitHub integration #33 #35
- dropped CUDA sm_13 support (now sm_20+ is required) #42

5.23 Open Beta RC1

Date: 2013-09-05 07:47:03 -0700

This is the first release candidate, a *pre-beta* version. We tagged this state since we started to support `sm_20+` only.

5.23.1 Changes to "Open Alpha"

n/a

5.24 Open Alpha

Date: 2013-08-14 02:25:36 -0700

That's our our open alpha release. The alpha release is **developer** and **power user** release only! **Users** should wait for our beta release!

CHAPTER

SIX

PICONGPU IN 5 MINUTES ON HEMERA

A guide to run, but not understand PIConGPU. It is aimed at users of the high performance computing (HPC) cluster "Hemera" at the HZDR, but should be applicable to other HPC clusters with slight adjustments.

This guide needs **shell access** (probably via **`ssh`**) and **`git`**. Consider getting familiar with the shell (*command line*, usually **`bash`**) and git. Please also read the tutorial for your local HPC cluster.

See also:

resources for the command line (bash)
a tutorial | another tutorial | scripting by examples

resources for git
official tutorial (also available as man page *`gittutorial(7)`*) | w3school tutorial | brief introduction | cheat-sheet (by github)

Hemera at HZDR
official website | presentation internal links: wiki | storage layout

We will use the following directories:

- `~/src/picongpu`: source files from github
- `~/fwkt_v100_picongpu.profile`: load the dependencies for your local environment
- `~/picongpu-projects`: scenarios to simulate
- `/bigdata/hplsim/external/alice`: result data of the simulation runs (*scratch* storage)

Please replace them whenever appropriate.

6.1 Get the Source

Use **`git`** to obtain the source and use the current `dev` branch and put it into `~/src/picongpu`:

```
mkdir -p ~/src
git clone https://github.com/ComputationalRadiationPhysics/picongpu ~/src/picongpu
```

Note: If you get the error `git: command not found` load git by invoking `module load git` and try again. Attention: the example uses the `dev` branch instead of the latest stable release. Due to driver changes on hemera the modules configuration of the last release might be outdated.

6.2 Setup

You need *a lot of dependencies*.

Luckily, other people already did the work and prepared a *profile* that you can use. Copy it to your home directory:

```
cp ~/src/picongpu/etc/picongpu/hemera-hzdr/fwkt_v100_picongpu.profile.example ~/fwkt_
↪v100_picongpu.profile
```

This profile determines which part of the HPC cluster (*partition*, also: *queue*) – and thereby the compute device(s) (type of CPUs/GPUs) – you will use. This particular profile will use NVIDIA Volta V100 GPUs.

You can view the full list of available profiles on github (look for `NAME.profile.example`).

For this guide we will add our scratch directory location to this profile. Edit the profile file using your favorite editor. If unsure use nano: `nano ~/fwkt_v100_picongpu.profile` (save with `Control-o`, exit with `Control-x`). Go to the end of the file and add a new line:

```
export SCRATCH=/bigdata/hplsim/external/alice
```

(Please replace `alice` with your username.)

Note: This is the location where runtime data and all results will be stored. If you're not on Hemera make sure you select the correct directory: Consult the documentation of your HPC cluster where to save your data. **On HPC clusters this is probably not your home directory.**

In the profile file you can also supply additional settings, like your email address and notification settings.

Now activate your profile:

```
source ~/fwkt_v100_picongpu.profile
```

Warning: You will have to repeat this command **every time** you want to use PIConGPU on a new shell, i.e. after logging in.

Now test your new profile:

```
echo $SCRATCH
```

That should print your data directory. If that works make sure that this directory actually exists by executing:

```
mkdir -p $SCRATCH
ls -lah $SCRATCH
```

If you see output similar to this one everything worked and you can carry on:

```
total 0
drwxr-xr-x  2 alice    fwt   40 Nov 12 10:09 .
drwxrwxrwt 17 root     root 400 Nov 12 10:09 ..
```

6.3 Create a Scenario

As an example we will use the predefined LaserWakefield example. Create a directory and copy it:

```
mkdir -p ~/picongpu-projects/tinkering
pic-create $PIC_EXAMPLES/LaserWakefield ~/picongpu-projects/tinkering/try01
cd ~/picongpu-projects/tinkering/try01
```

Usually you would now adjust the files in the newly created directory `~/picongpu-projects/tinkering/try01` – for this introduction we will use the parameters as provided.

Note: The command **`pic-create`** and the variable `$PIC_EXAMPLES` have been provided because you loaded the file `~/fwkt_v100_picongpu.profile` in the previous step. If this fails (printing `pic-create: command not found`), make sure you load the PIConGPU profile by executing `source ~/fwkt_v100_picongpu.profile`.

6.4 Compile and Run

Now use a compute node. Your profile provides a helper command for that:

```
getDevice
```

(You can now run `hostname` to see which node you are using.)

Now build the scenario:

```
# switch to the scenario directory if you haven't already
cd ~/picongpu-projects/tinkering/try01
pic-build
```

This will take a while, go grab a coffee. If this fails, read the manual or ask a colleague.

After a successfull build, run (still on the compute node, still inside your scenario directory):

```
tbg -s bash -t $PICSRC/etc/picongpu/bash/mpiexec.tpl -c /etc/picongpu/1.cfg $SCRATCH/
↪tinkering/try01/run01
```

- **`tbg`**: tool provided by PIConGPU
- `bash`: the "submit system", e.g. use `sbatch` for slurm
- `$PICSRC`: the path to your PIConGPU source code, automatically set when sourcing `fwkt_v100_picongpu.profile`
- `$PICSRC/etc/picongpu/bash/mpiexec.tpl`: options for the chosen submit system
- `etc/picongpu/1.cfg`: runtime options (number of GPUs, etc.)
- `$SCRATCH/tinkering/try01/run01`: not-yet-existing destination for your result files

Note: Usually you would use the *workload manager* (SLURM on Hemera) to submit your jobs instead of running them interactively like we just did. You can try that with:

```
# go back to the login node
exit
hostname
# ...should now display hemera4.cluster or hemera5.cluster
```

(continues on next page)

(continued from previous page)

```
# resubmit your simulation with a new directory:
tbg -s sbatch -c etc/picongpu/1.cfg -t etc/picongpu/hemera-hzdr/fwkt_v100.tpl
↪$SCRATCH/tinkering/try01/run02
```

This will print a confirmation message (e.g. `Submitted batch job 3769365`), but no output of PIConGPU itself will be printed. Using `squeue -u $USER` you can view the current status of your job.

Note that we not only used a different "submit system" `sbatch`, but also changed the template file to `etc/picongpu/hemera-hzdr/fwkt_v100.tpl`. (This template file is directly located in your project directory.`) Both profile and template file are built for the same compute device, the NVIDIA Volta "V100" GPU.

6.5 Examine the Results

Results are located at `$SCRATCH/tinkering/try01/run01`.

To view pretty pictures from a linux workstation you can use the following process (execute on your workstation, **not the HPC cluster**):

```
# Create a "mount point" (empty directory)
mkdir -p ~/mnt/scratch

# Mount the data directory using sshfs
sshfs -o default_permissions -o idmap=user -o uid=$(id -u) -o gid=$(id -g)␣
↪hemera5:DATADIR ~/mnt/scratch/
```

Substitute DATADIR with the full path to your data (*scratch*) directory, e.g. `/bigdata/hplsim/external/alice`.

Browse the directory using a file browser/image viewer. Check out `~/mnt/scratch/tinkering/try01/run01/simOutput/pngElectronsYX/` for image files.

6.6 Further Reading

You now know the process of using PIConGPU. Carry on reading the documentation to understand it.

CHAPTER

SEVEN

REFERENCE

Section author: Axel Huebl

PIConGPU is a more than decade-long scientific project with many people contributing to it. In order to credit the work of others, we expect you to cite our latest paper describing PIConGPU when publishing and/or presenting scientific results.

In addition to that and out of good scientific practice, you should document the version of PIConGPU that was used and any modifications you applied. A list of releases alongside a DOI to reference it can be found here:

https://github.com/ComputationalRadiationPhysics/picongpu/releases

7.1 Citation

BibTeX code:

```
@inproceedings{PIConGPU2013,
 author = {Bussmann, M. and Burau, H. and Cowan, T. E. and Debus, A. and Huebl, A.␣
↪and Juckeland, G. and Kluge, T. and Nagel, W. E. and Pausch, R. and Schmitt, F. and␣
↪Schramm, U. and Schuchart, J. and Widera, R.},
 title = {Radiative Signatures of the Relativistic Kelvin-Helmholtz Instability},
 booktitle = {Proceedings of the International Conference on High Performance␣
↪Computing, Networking, Storage and Analysis},
 series = {SC '13},
 year = {2013},
 isbn = {978-1-4503-2378-9},
 location = {Denver, Colorado},
 pages = {5:1--5:12},
 articleno = {5},
 numpages = {12},
 url = {http://doi.acm.org/10.1145/2503210.2504564},
 doi = {10.1145/2503210.2504564},
 acmid = {2504564},
 publisher = {ACM},
 address = {New York, NY, USA},
}
```

7.2 Acknowledgements

In many cases you receive support and code base maintainance from us or the PIConGPU community without directly justifying a full co-authorship. Additional to the citation, please consider adding an acknowledgement of the following form to reflect that:

> We acknowledge all contributors to the open-source code PIConGPU for enabling our simulations.

or:

> We acknowledge [list of specific persons that helped you] and all further contributors to the open-source code PIConGPU for enabling our simulations.

7.3 Community Map

PIConGPU comes without a registration-wall, with open and re-distributable licenses and without any strings attached. We therefore *rely on you* to show our community, diversity and usefulness, e.g. to funding agencies.

> Please consider adding yourself to our community map!

Thank you and enjoy PIConGPU and our community!

7.4 Publications

The following publications are sorted by topics. Papers covering multiple topics will be listed multiple times. In the end, a list of all publications in chronological order is given with more details. If you want to add your publication to the list as well please feel free to contact us or open a pull request directly.

7.4.1 Application of PIConGPU in various physics scenarios

In the following, a list of publications describing using PIConGPU is given in various cases.

Laser plasma electron acceleration

- Laser wakefield acceleration (LWFA) with self-truncated ionization-injection (STII) [Couperus2017]
- PhD thesis on experimental aspects of LWFA with STII [Couperus2018],
- PhD thesis on theoretical ascpects of self-focusing during LWFA with STII [Pausch2019]
- Hybrid laser-driven/beam-driven plasms acceleration [Kurz2021]
- Acceleration in carbon nanotubes [Perera2020]
- TeV/m catapult acceleration in graphene layers [Bontoiu2023]

Laser plasma ion acceleration

- Proton acceleration from cryogenic hydrogen jets [Obst2017]
- Mass-Limited, Near-Critical, micron-scale, spherical targets [Hilz2018]
- PhD thesis on theoretical aspects of mass-limited, near-critical, micron-scale, spherical targets [Huebl2019]
- All-optical stuctering of laser-accelerated protons [Obst-Huebl2018]
- PhD thesis on laser-driven proton beam structering [Obst-Huebl2019]
- Laser-ion multi-species acceleration [Huebl2020]

Laser plasma light sources and diagnostics

- PhD thesis on radiation from LWFA [Pausch2019]
- Laser-driven x-ray and proton sources [Ostermayr2020]
- Betatron x-ray diagnostic for beam decoherence [Koehler2019], [Koehler2021]

Astrophysics

- Simulating the Kelvin Helmholtz instability (KHI) [Bussmann2013]
- Visualizing the KHI [Huebl2014]
- Theoretical model of the radiation evolution during the KHI [Pausch2017]
- PhD thesis covering the KHI radiation [Pausch2019]

Machine Learning

7.4.2 Methods used in PIConGPU software

In the following, a list of references is given, sorted by topics, that describe PIConGPU as a software. References to publications of implemented methods that were developed by other groups are also given for completeness. These references are marked by an asterisk.

General code references

- First publication on PIConGPU [Burau2010]
- Currently main reference [Bussmann2013]
- Most up-to-date reference [Huebl2019]

Field solvers

- Yee field solver* [Yee1966]
- Lehe field solver* [Lehe2013]
- High-Order Finite Difference solver* [Ghrist2000]

Particle pushers

- Boris pusher* [Boris1970]
- Vay pusher* [Vay2008]
- Reduced Landau-Lifshitz pusher* [Vranic2016]
- Higuera-Cary pusher* [Higuera2017]

Current deposition

- Esirkepov method* [Esirkepov2001]

Ionization-physics extensions

- Barrier suppression ionization (BSI)* [ClementiRaimondi1963], [ClementiRaimondi1967], [MulserBauer2010]
- Tunneling ionization - Keldysh* [Keldysh]
- Tunneling ionization - Ammosov-Delone-Krainov (ADK)* [DeloneKrainov1998], [BauerMulser1999]
- Master thesis - model implementation [Garten2015]
- Collisional ionization [FLYCHK2005], [More1985]
- ionization current* [Mulser1998]

Binary_collisions

- fundamental alogorithm* [Perez2012]
- improvements and corrections* [Higginson2020]

QED code extensions

- Various methods applicable in PIC codes* [Gonoskov2015]
- Diploma thesis - model implementation [Burau2016]

Diagnostic methods

- classical radiation: [Pausch2012], [Pausch2014], [Pausch2018], [Pausch2019]
- phase space analysis: [Huebl2014]
- beam emittance: [Rudat2019]
- coherent transistion radiation (CTR): [Carstens2019]

Visualization

- first post-processing implementation: [Zuehl2011], [Ungethuem2012]
- first in-situ visualization: [Schneider2012a], [Schneider2012b]
- Kelvin-Helmholtz instabilty: [Huebl2014]
- in-situ visualization with ISAAC: [Matthes2016], [Matthes2016b]
- in-situ particle rendering: [Meyer2018]

Input/Output

- parallel HDF5, ADIOS1, compression, data reduction and I/O performance model [Huebl2017]

HPC kernels and benchmarks

- proceedings of the SC'13 [Bussmann2013]

7.4.3 Theses

- Diploma thesis: first post-processing rendering [Zuehl2011]
- Diploma thesis: first in-situ rendering [Schneider2012b]
- Diploma thesis: In-situ radiation calculation [Pausch2012]
- Diploma thesis: Algorithms, LWFA injection, Phase Space analysis [Huebl2014]
- Master thesis: Ionization methods [Garten2015]
- Diploma thesis: QED scattering processes [Burau2016]
- Diploma thesis: In-situ live visualization [Matthes2016]
- PhD thesis: LWFA injection using STII (mainly experiment) [Couperus2018]
- Bachelor thesis: In-situ live visualization [Meyer2018]
- Master thesis: Beam emittance and automated parameter scans [Rudat2019]
- PhD thesis: Radiation during LWFA and KHI, radiative corrections [Pausch2019]
- PhD thesis: LWFA betatron radiation (mainly experiment) [Koehler2019]
- PhD thesis: LWFA Coherent transistion radiation diagnostics (CTR) (mainly experiment) [Zarini2019]
- PhD thesis: Laser ion acceleration (mainly experiment) [Obst-Huebl2019]
- PhD thesis: Exascale simulations with PIConGPU, laser ion acceleration [Huebl2019]
- Bachelor thesis: Synthetic coherent transistion radiation [Carstens2019]

7.4.4 List of PIConGPU references in chronological order

7.4.5 List of other references in chronological order

See also:

You need to have an *environment loaded* (`source $HOME/picongpu.profile` when installing from source) that provides all *PIConGPU dependencies* to complete this chapter.

Warning: PIConGPU source code is portable and can be compiled on all major operating systems. However, helper tools like `pic-create` and `pic-build` described in this section rely on Linux utilities and thus are not expected to work on other platforms out-of-the-box. Note that building and using PIConGPU on other operating systems is still possible but has to be done manually or with custom tools. This case is not covered in the documentation, but we can assist users with it when needed.

CHAPTER

EIGHT

BASICS

Section author: Axel Huebl

8.1 Preparation

First, decide where to store input files, a good place might be `$HOME` (~) because it is usually backed up. Second, decide where to store your output of simulations which needs to be placed on a high-bandwidth, large-storage file system which we will refer to as `$SCRATCH`.

For a first test you can also use your home directory:

```
export SCRATCH=$HOME
```

We need a few directories to structure our workflow:

```
# PIConGPU input files
mkdir $HOME/picInputs

# PIConGPU simulation output
mkdir $SCRATCH/runs
```

8.2 Step-by-Step

8.2.1 1. Create an Input (Parameter) Set

```
# clone the LWFA example to $HOME/picInputs/myLWFA
pic-create $PIC_EXAMPLES/LaserWakefield $HOME/picInputs/myLWFA

# switch to your input directory
cd $HOME/picInputs/myLWFA
```

PIConGPU is controlled via two kinds of textual input sets: compile-time options and runtime options.

Compile-time *.param files* reside in `include/picongpu/param/` and define the physics case and deployed numerics. After creation and whenever options are changed, PIConGPU *requires a re-compile*. Feel free to take a look now, but we will later come back on how to *edit those files*.

Runtime (command line) arguments are set in `etc/picongpu/*.cfg` files. These options do *not* require a re-compile when changed (e.g. simulation size, number of devices, plugins, . . .).

8.2.2 2. Compile Simulation

In our input, `.param` files are build directly into the PIConGPU binary for *performance reasons*. A compile is required after changing or initially adding those files.

In this step you can optimize the simulation for the specific hardware you want to run on. By default, we compile for Nvidia GPUs with the CUDA backend, targeting the oldest compatible architecture.

```
pic-build
```

This step will take a few minutes. Time for a coffee or a sword fight!

We explain in the *details section* below how to set further options, e.g. CPU targets or tuning for newer GPU architectures.

8.2.3 3. Run Simulation

While you are still in `$HOME/picInputs/myLWFA`, start your simulation on one CUDA capable GPU:

```
# example run for an interactive simulation on the same machine
tbg -s bash -c etc/picongpu/1.cfg -t etc/picongpu/bash/mpiexec.tpl $SCRATCH/runs/lwfa_
↪001
```

This will create the directory `$SCRATCH/runs/lwfa_001` where all simulation output will be written to. `tbg` will further create a subfolder `input/` in the directory of the run with the same structure as `myLWFA` to archive your input files. Subfolder `simOutput/` has all the simulation results. Particularly, the simulation progress log is in `simOutput/output`.

8.2.4 4. Creating Own Simulation

For creating an own simulation, we recommend starting with the most fitting example and modifying the *compile-time options in .param files* and *run-time options in .cfg files*. Changing contents of `.param` files requires recompilation of the code, modifying `.cfg` files does not. Note that available run-time options generally depend on the environment used for the build, the chosen compute backend, and the contents of `.param` files. To get the list of all available options for the current configuration, after a successful `pic-build` run

```
.build/picongpu --help
```

8.3 Details on the Commands Above

8.3.1 tbg

The `tbg` tool is explained in detail *in its own section*. Its primary purpose is to abstract the options in runtime `.cfg` files from the technical details on how to run on various supercomputers.

For example, if you want to run on the HPC System "Hemera" at HZDR, your `tbg` submit command would just change to:

```
# request 1 GPU from the PBS batch system and run on the queue "k20"
tbg -s sbatch -c etc/picongpu/1.cfg -t etc/picongpu/hemera-hzdr/k20.tpl $SCRATCH/runs/
↪lwfa_002

# run again, this time on 16 GPUs
tbg -s sbatch -c etc/picongpu/16.cfg -t etc/picongpu/hemera-hzdr/k20.tpl $SCRATCH/
↪runs/lwfa_003
```

Note that we can use the same `1.cfg` file, your input set is *portable*.

8.3.2 pic-create

This tool is just a short-hand to create a new set of input files. It copies from an already existing set of input files (e.g. our examples or a previous simulation) and adds additional helper files.

See `pic-create --help` for more options during input set creation:

```
pic-create create a new parameter set for simulation input
merge default picongpu parameters and a given example's input

usage: pic-create [OPTION] [src_dir] dest_dir
If no src_dir is set picongpu a default case is cloned
If src_dir is not in the current directory, pic-create will
look for it in $PIC_EXAMPLES

-f | --force         - merge data if destination already exists
-h | --help          - show this help message

Dependencies: rsync
```

A run simulation can also be reused to create derived input sets via `pic-create`:

```
pic-create $SCRATCH/runs/lwfa_001/input $HOME/picInputs/mySecondLWFA
```

8.3.3 pic-build

This tool is actually a short-hand for an *out-of-source build with CMake*.

In detail, it does:

```
# go to an empty build directory
mkdir -p .build
cd .build

# configure with CMake
pic-configure $OPTIONS ..

# compile PIConGPU with the current input set (e.g. myLWFA)
# - "make -j install" runs implicitly "make -j" and then "make install"
# - make install copies resulting binaries to input set
make -j install
```

`pic-build` accepts the same command line flags as *pic-configure*. For example, if you want to build for running on CPUs instead of a GPUs, call:

```
# example for running efficiently on the CPU you are currently compiling on
pic-build -b "omp2b"
```

Its full documentation from `pic-build --help` reads:

```
Build new binaries for a PIConGPU input set

Creates or updates the binaries in an input set. This step needs to
be performed every time a .param file is changed.

This tools creates a temporary build directory, configures and
compiles current input set in it and installs the resulting
binaries.
```

(continues on next page)

(continued from previous page)

```
This is just a short-hand tool for switching to a temporary build
directory and running 'pic-configure ..' and 'make install'
manually.

You must run this command inside an input directory.

usage: pic-build [OPTIONS]

-b | --backend       - set compute backend and optionally the architecture
                       syntax: backend[:architecture]
                       supported backends: cuda, hip, omp2b, serial, tbb, threads
                       (e.g.: "cuda:35;37;52;60" or "omp2b:native" or "omp2b")
                       default: "cuda" if not set via environment variable PIC_BACKEND
                       note: architecture names are compiler dependent
-c | --cmake         - overwrite options for cmake
                       (e.g.: "-DPIC_VERBOSE=21 -DCMAKE_BUILD_TYPE=Debug")
-t <presetNumber>    - configure this preset from cmakeFlags
-f | --force         - clear the cmake file cache and scan for new param files
-h | --help          - show this help message
```

8.3.4 pic-configure

This tool is just a convenient wrapper for a call to CMake. It is executed from an *empty build directory*.

You will likely not use this tool directly. Instead, *pic-build* from above calls `pic-configure` for you, forwarding its arguments.

We *strongly recommend* to set the appropriate target compute backend via `-b` for optimal performance. For Nvidia CUDA GPUs, set the compute capability of your GPU:

```
# example for running efficiently on a K80 GPU with compute capability 3.7
pic-configure -b "cuda:37" $HOME/picInputs/myLWFA
```

For running on a CPU instead of a GPU, set this:

```
# example for running efficiently on the CPU you are currently compiling on
pic-configure -b "omp2b:native" $HOME/picInputs/myLWFA
```

Note: If you are compiling on a cluster, the CPU architecture of the head/login nodes versus the actual compute architecture does likely vary! Compiling a backend for the wrong architecture does in the best case dramatically reduce your performance and in the worst case will not run at all!

During configure, the backend's architecture is forwarded to the compiler's `-mtune` and `-march` flags. For example, if you are compiling with GCC for running on *AMD Opteron 6276 CPUs* set `-b omp2b:bdver1` or for *Intel Xeon Phi Knight's Landing CPUs* set `-b omp2b:knl`.

See `pic-configure --help` for more options during input set configuration:

```
Configure PIConGPU with CMake

Generates a call to CMake and provides short-hand access to selected
PIConGPU CMake options.
Advanced users can always run 'ccmake .' after this call for further
compilation options.

```

(continues on next page)

(continued from previous page)

```
usage: pic-configure [OPTIONS] <inputDirectory>

-i | --install       - path were picongpu shall be installed
                       (default is <inputDirectory>)
-b | --backend       - set compute backend and optionally the architecture
                       syntax: backend[:architecture]
                       supported backends: cuda, hip, omp2b, serial, tbb, threads
                       (e.g.: "cuda:35;37;52;60" or "omp2b:native" or "omp2b")
                       default: "cuda" if not set via environment variable PIC_BACKEND
                       note: architecture names are compiler dependent
-c | --cmake         - overwrite options for cmake
                       (e.g.: "-DPIC_VERBOSE=21 -DCMAKE_BUILD_TYPE=Debug")
-t <presetNumber>    - configure this preset from cmakeFlags
-f | --force         - clear the cmake file cache and scan for new param files
-h | --help          - show this help message
```

After running configure you can run `ccmake .` to set additional compile options (optimizations, debug levels, hardware version, etc.). This will influence your build done via `make install`.

You can pass further options to configure PIConGPU directly instead of using `ccmake .`, by passing `-c "-DOPTION1=VALUE1 -DOPTION2=VALUE2"`.

CHAPTER

NINE

.PARAM FILES

Section author: Axel Huebl

Parameter files, `*.param` placed in `include/picongpu/param/` are used to set all **compile-time options** for a PIConGPU simulation. This includes most fundamental options such as numerical solvers, floating precision, memory usage due to attributes and super-cell based algorithms, density profiles, initial conditions etc.

9.1 Editing

For convenience, we provide a tool `pic-edit` to edit the compile-time input by its name. For example, if you want to edit the *grid* and time step resolution, *file output* and add a *laser* to the simulation, open the according files via:

```
# first switch to your input directory
cd $HOME/picInputs/myLWFA

pic-edit grid fileOutput laser
```

See `pic-edit --help` for all available files:

```
Edit compile-time options for a PIConGPU input set

Opens .param files in an input set with the default "EDITOR".
If a .param file is not yet part of the input set but exists in the
defaults, it will be transparently added to the input set.

You must run this command inside an input directory.

The currently selected editor is: /usr/bin/vim.basic
You can change it via the "EDITOR" environment variable.

usage: pic-edit <input>

Available <input>s:
atomicPhysics binningSetup collision components density dimension fieldAbsorber␣
↪fieldBackground fieldSolver fileOutput grid incidentField ionizationEnergies␣
↪ionizer isaac iterationStart mallocMC memory particle particleCalorimeter␣
↪particleFilters physicalConstants png pngColorScales precision pusher radiation␣
↪radiationObserver random shadowgraphy species speciesAttributes speciesConstants␣
↪speciesDefinition speciesInitialization starter transitionRadiation unit
```

9.2 Rationale

High-performance hardware comes with a lot of restrictions on how to use it, mainly memory, control flow and register limits. In order to create an efficient simulation, PIConGPU compiles to **exactly** the numerical solvers (kernels) and physical attributes (fields, species) for the setup you need to run, which will furthermore be specialized for a specific hardware.

This comes at a small cost: when **even one of those settings is changed, you need to recompile**. Nevertheless, wasting about 5 minutes compiling on a single node is nothing compared to the time you save *at scale*!

All options that are less or non-critical for runtime performance, such as specific ranges, observables in *plugins* or how many nodes shall be used, can be set in *run time configuration files (*.cfg)* and do not need a recompile when changed.

9.3 Files and Their Usage

If you use our `pic-configure` *script wrappers*, you do not need to set *all* available parameter files since we will add the missing ones with *sane defaults*. Those defaults are:

- a standard, single-precision, well normalized PIC cycle suitable for relativistic plasmas
- no external forces (no laser, no initial density profile, no background fields, etc.)

9.4 All Files

When setting up a simulation, it is recommended to adjust `.param` files in the following order:

9.4.1 PIC Core

dimension.param

The spatial dimensionality of the simulation.

Defines

`SIMDIM`

Possible values: DIM3 for 3D3V and DIM2 for 2D3V.

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. “Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences.” Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. “A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter” High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Variables

constexpr uint32_t **simDim** = SIMDIM

grid.param

Definition of cell sizes and time step.

Our cells are defining a regular, cartesian grid. Our explicit FDTD field solvers require an upper bound for the time step value in relation to the cell size for convergence. Make sure to resolve important wavelengths of your simulation, e.g. shortest plasma wavelength, Debye length and central laser wavelength both spatially and temporarily.

Units in reduced dimensions

In 2D3V simulations, the CELL_DEPTH_SI (Z) cell length is still used for normalization of densities, etc..

A 2D3V simulation in a cartesian PIC simulation such as ours only changes the degrees of freedom in motion for (macro) particles and all (field) information in z travels instantaneously, making the 2D3V simulation behave like the interaction of infinite "wire particles" in fields with perfect symmetry in Z.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **SI**

Variables

constexpr float_64 **DELTA_T_SI** = 0.8e-16

Duration of one timestep unit: seconds.

constexpr float_64 **CELL_WIDTH_SI** = 0.1772e-6

equals X unit: meter

constexpr float_64 **CELL_HEIGHT_SI** = 0.4430e-7

equals Y - the laser & moving window propagation direction unit: meter

constexpr float_64 **CELL_DEPTH_SI** = *CELL_WIDTH_SI*

equals Z unit: meter

components.param

Select a user-defined simulation class here, e.g.

with strongly modified initialization and/or PIC loop beyond the parametrization given in other .param files.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  -
    R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  -
    H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **simulation_starter**

Simulation Starter Selection: This value does usually not need to be changed.

Change only if you want to implement your own `SimulationHelper` (e.g. `Simulation`) class.

- defaultPIConGPU : default PIConGPU configuration

iterationStart.param

Specify a sequence of functors to be called at start of each time iteration.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Typedefs

using **IterationStartPipeline** = *pmacc*::mp_list<>

IterationStartPipeline defines the functors called at each iteration start.

The functors will be called in the given order.

The functors must be default-constructible and take the current time iteration as the only parameter. These are the same requirements as for functors in particles::InitPipeline.

fieldSolver.param

Configure the field solver.

Select the numerical Maxwell solver (e.g. Yee's method).

Attention

Currently, the laser initialization in PIConGPU is implemented to work with the standard Yee solver. Using a solver of higher order will result in a slightly increased laser amplitude and energy than expected.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  –
    R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  –
    H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **fields**

Typedefs

using **Solver** = maxwellSolver::Yee

FieldSolver.

Field Solver Selection (note <> for some solvers), all in namespace maxwellSolver:
- Yee: Standard Yee solver approximating derivatives with respect to time and space by second order finite differences.
- CKC: Cole-Karkkainen-Cowan Solver, Dispersion free solver in the direction of the smallest grid size.
- Lehe<>: Num. Cherenkov free field solver in a chosen direction
- ArbitraryOrderFDTD<4>: Solver using 4 neighbors to each direction to approximate *spatial* derivatives by finite differences. The number of neighbors can be changed from 4 to any positive, integer number. The order of the solver will be twice the number of neighbors in each direction. Yee's method is a special case of this using one neighbor to each direction.
- Substepping<Solver, 4>: use the given Solver (Yee, etc.) and substep each time step by factor 4
- None: disable the vacuum update of E and B, including no J contribution to E

fieldAbsorber.param

Configure the field absorber parameters.

Field absorber type is set by command-line option &#8212;fieldAbsorber.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published

–

R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)

–

H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace `fields`

namespace `absorber`

Variables

constexpr uint32_t `THICKNESS` = 12

constexpr uint32_t `NUM_CELLS`[3][2] = {{*THICKNESS*, *THICKNESS*}, {*THICKNESS*, *THICKNESS*}, {*THICKNESS*, *THICKNESS*}}

Thickness of the absorbing layer, in number of cells.

This setting applies to applies for all absorber kinds. The absorber layer is located inside the global simulation area, near the outer borders. Setting size to 0 results in disabling absorption at the corresponding boundary. Note that for non-absorbing boundaries the actual thickness will be 0 anyways. There are no requirements on thickness being a multiple of the supercell size.

For PML the recommended thickness is between 6 and 16 cells. For the exponential damping it is 32.

Unit: number of cells.

namespace `exponential`

Settings for the Exponential absorber.

Variables

constexpr float_X `STRENGTH`[3][2] = {{1.0e-3, 1.0e-3}, {1.0e-3, 1.0e-3}, {1.0e-3, 1.0e-3}}

Define the strength of the absorber for all directions.

Elements corredponding to non-absorber borders will have no effect.

Unit: none

namespace **pml**

Settings for the Pml absorber.

These parameters can generally be left with default values. For more details on the meaning of the parameters, refer to the following references. J.A. Roden, S.D. Gedney. Convolution PML (CPML): An efficient FDTD implementation of the CFS - PML for arbitrary media. Microwave and optical technology letters. 27 (5), 334-339 (2000). A. Taflove, S.C. Hagness. Computational Electrodynamics. The Finite-Difference Time-Domain Method. Third Edition. Artech house, Boston (2005), referred to as [Taflove, Hagness].

Variables

constexpr float_64 **SIGMA_KAPPA_GRADING_ORDER** = 4.0

Order of polynomial grading for artificial electric conductivity and stretching coefficient.

The conductivity (sigma) is polynomially scaling from 0 at the internal border of PML to the maximum value (defined below) at the external border. The stretching coefficient (kappa) scales from 1 to the corresponding maximum value (defined below) with the same polynomial. The grading is given in [Taflove, Hagness], eq. (7.60a, b), with the order denoted 'm'. Must be >= 0. Normally between 3 and 4, not required to be integer. Unitless.

constexpr float_64 **SIGMA_OPT_SI**[3] = {0.8 * (*SIGMA_KAPPA_GRADING_ORDER* + 1.0) / (*SI*::*Z0_SI* * *SI*::*CELL_WIDTH_SI*), 0.8 * (*SIGMA_KAPPA_GRADING_ORDER* + 1.0) / (*SI*::*Z0_SI* * *SI*::*CELL_HEIGHT_SI*), 0.8 * (*SIGMA_KAPPA_GRADING_ORDER* + 1.0) / (*SI*::*Z0_SI* * *SI*::*CELL_DEPTH_SI*)}

constexpr float_64 **SIGMA_OPT_MULTIPLIER** = 1.0

constexpr float_64 **SIGMA_MAX_SI**[3] = {*SIGMA_OPT_SI*[0] * *SIGMA_OPT_MULTIPLIER*, *SIGMA_OPT_SI*[1] * *SIGMA_OPT_MULTIPLIER*, *SIGMA_OPT_SI*[2] * *SIGMA_OPT_MULTIPLIER*}

Max value of artificial electric conductivity in PML.

Components correspond to directions: element 0 corresponds to absorption along x direction, 1 = y, 2 = z. Grading is described in comments for SIGMA_KAPPA_GRADING_ORDER. Too small values lead to significant reflections from the external border, too large - to reflections due to discretization errors. Artificial magnetic permeability will be chosen to perfectly match this. Must be >= 0. Normally between 0.7 * SIGMA_OPT_SI and 1.1 * SIGMA_OPT_SI. Unit: siemens / m.

constexpr float_64 **KAPPA_MAX**[3] = {1.0, 1.0, 1.0}

Max value of coordinate stretching coefficient in PML.

Components correspond to directions: element 0 corresponds to absorption along x direction, 1 = y, 2 = z. Grading is described in comments for SIGMA_KAPPA_GRADING_ORDER. Must be >= 1. For relatively homogeneous domains 1.0 is a reasonable value. Highly elongated domains can have better absorption with values between 7.0 and 20.0, for example, see section 7.11.2 in [Taflove, Hagness]. Unitless.

constexpr float_64 **ALPHA_GRADING_ORDER** = 1.0

Order of polynomial grading for complex frequency shift.

The complex frequency shift (alpha) is polynomially downscaling from the maximum value (defined below) at the internal border of PML to 0 at the external border. The grading is given in [Taflove, Hagness], eq. (7.79), with the order denoted 'm_a'. Must be >= 0. Normally values are around 1.0. Unitless.

constexpr float_64 **ALPHA_MAX_SI**[3] = {0.2, 0.2, 0.2}

Complex frequency shift in PML.

Components correspond to directions: element 0 corresponds to absorption along x direction, 1 = y, 2 = z. Setting it to 0 will make PML behave as uniaxial PML. Setting it to a positive value helps to attenuate evanescent modes, but can degrade absorption of propagating modes, as described in section 7.7 and 7.11.3 in [Taflove, Hagness]. Must be >= 0. Normally values are 0 or between 0.15 and 0.3. Unit: siemens / m.

incidentField.param

Configure incident field profile and offset of the Huygens surface for each boundary.

Available profiles:

- profiles::DispersivePulse<> : Gaussian pulse allowing to set first-, second-, and third-order dispersion in focus. That is, SD, AD, GDD, and TOD, respectively.
- profiles::ExpRampWithPrepulse<> : exponential ramp with prepulse wavepacket with given parameters
- profiles::Free<> : custom profile with user-provided functors to calculate incident E and B
- profiles::GaussianPulse<> : Pulse with Gaussian profile in all three dimensions with given parameters
- profiles::None : no incident field
- profiles::PlaneWave<> : plane wave profile with given parameters
- profiles::Polynom<> : wavepacket with a polynomial temporal intensity shape profile with given parameters
- profiles::PulseFrontTilt<> : GaussianPulse with tilted pulse front with given parameters
- profiles::Wavepacket<> : wavepacket with Gaussian spatial and temporal envelope profile with given parameters

All profiles but `Free<>` and `None` are parametrized with a profile-specific structure. Their interfaces are defined in the corresponding `.def` files inside directory picongpu/fields/incidentField/profiles/. Note that all these parameter structures inherit common base structures from `BaseParam.def`. Thus, a user-provided structure must also define all members according to the base struct.

In the end, this file needs to define `XMin`, `XMax`, `YMin`, `YMax`, `ZMin`, `ZMax` (the latter two can be skipped in 2d) type aliases in namespace `picongpu::fields::incidentField`. Each of them could be a single profile or a typelist of profiles created with `MakeSeq_t`. In case a typelist is used, the resulting field is a sum of effects of all profiles in the list. This file also has to define constexpr array `POSITION` that controls positioning of the generating surface relative to total domain. For example:

```
using XMin = profiles::Free<UserFunctorIncidentE, UserFunctorIncidentB>;
using XMax = profiles::None;
using YMin = MakeSeq_t<profiles::PlaneWave<UserPlaneWaveParams>, profiles::Wavepacket
↪<UserWavepacketParams>>;
using YMax = profiles::None;
using ZMin = profiles::Polynom<UserPolynomParams>;
using ZMax = profiles::GaussianPulse<UserGaussianPulseParams>;

constexpr int32_t POSITION[3][2] = { {16, -16}, {16, -16}, {16, -16} };
```

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)

–

H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **fields**

namespace **incidentField**

Unnamed Group

using **XMin** = profiles::None

Incident field profile types along each boundary, these 6 types (or aliases) are required.

using **XMax** = profiles::None

using **YMin** = profiles::None

using **YMax** = profiles::None

using **ZMin** = profiles::None

using **ZMax** = profiles::None

Typedefs

using **MyProfile** = profiles::Free<*UserFunctorIncidentE*, *UserFunctorIncidentB*>

Make a profile with the user-provided functors defined above.

The functors for incident field E and B must be consistent to each other. They should work for all boundaries the profile is applied to.

Variables

constexpr int32_t **POSITION**[3][2] = {{16, -16}, {16, -16}, {16, -16}}

Position in cells of the Huygens surface relative to start of the total domain.

The position is set as an offset, in cells, counted from the start of the total domain. For the max boundaries, negative position values are allowed. These negative values are treated as position at (global_domain_size[d] + POSITION[d][1]). It is also possible to specify the position explicitly as a positive number. Then it is on a user to make sure the position is correctly calculated wrt the grid size.

Except moving window simulations, the position must be inside the global domain. The distance between the Huygens surface and each global domain boundary must be at least absorber_thickness + (FDTD_spatial_order / 2 - 1). However beware of setting position = direction * (absorber_thickness + const), as then changing absorber parameters will affect laser positioning. When all used profiles are None, the check for POSITION validity is skipped.

For moving window simulations, POSITION for the YMax side can be located outside the initially simulated volume. In this case, parts of the generation surface outside of the currently simulated volume is are treated as if they had zero incident field and it is user's responsibility to apply a source matching such a case.

class **`UserFunctorIncidentE`**

User-defined functor to set values of incident E field.

Public Functions

`PMACC_ALIGN`(m_unitField, const float3_64)

inline HDINLINE **`UserFunctorIncidentE`**(float_X const currentStep, const float3_64 unitField)

Create a functor.

Parameters

- **`currentStep`** – current time step index, note that it is fractional
- **`unitField`** – conversion factor from SI to internal units, field_internal = field_SI / unitField

`inline HDINLINE float3_X operator() (const floatD_X &totalCellIdx) const`

Calculate incident field E_inc(r, t) for a source.

Parameters

`totalCellIdx` – cell index in the total domain (including all moving window slides), note that it is fractional

Returns

incident field value in internal units

class **`UserFunctorIncidentB`**

User-defined functor to set values of incident B field.

Public Functions

`PMACC_ALIGN`(m_unitField, const float3_64)

inline HDINLINE **`UserFunctorIncidentB`**(float_X const currentStep, const float3_64 unitField)

Create a functor.

Parameters

- **`currentStep`** – current time step index, note that it is fractional
- **`unitField`** – conversion factor from SI to internal units, field_internal = field_SI / unitField

`inline HDINLINE float3_X operator() (const floatD_X &totalCellIdx) const`

Calculate incident field B_inc(r, t) for a source.

Parameters

`totalCellIdx` – cell index in the total domain (including all moving window slides), note that it is fractional

Returns

incident field value in internal units

pusher.param

Configure particle pushers.

Those pushers can then be selected by a particle species in species.param and speciesDefinition.param

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

struct **particlePusherAccelerationParam**

Subclassed by picongpu::particlePusherAcceleration::UnitlessParam

Public Static Attributes

static constexpr float_64 **AMPLITUDEx_SI** = 0.0

Define strength of constant and homogeneous accelerating electric field in SI per dimension.

unit: Volt / meter

static constexpr float_64 **AMPLITUDEy_SI** = -1.e11

The moving window propagation direction unit: Volt / meter (1e11 V/m = 1 GV/cm)

static constexpr float_64 **AMPLITUDEz_SI** = 0.0

unit: Volt / meter

static constexpr float_64 **ACCELERATION_TIME_SI** = 10000.0 * ::*picongpu*::*SI*::*DELTA_T_SI*

Acceleration duration unit: second.

namespace **particlePusherAxel**

Enums

enum **`TrajectoryInterpolationType`**

Values:

enumerator **`LINEAR`**

enumerator **`NONLINEAR`**

Variables

constexpr *TrajectoryInterpolationType* **`TrajectoryInterpolation`** = LINEAR

namespace **`particlePusherHigueraCary`**

namespace **`particlePusherProbe`**

Typedefs

using **`ActualPusher`** = void

Also push the probe particles?

In many cases, probe particles are static throughout the simulation. This option allows to set an "actual" pusher that shall be used to also change the probe particle positions.

Examples:

- particles::pusher::Boris
- particles::pusher::[all others from above]
- void (no push)

namespace **`particlePusherVay`**

namespace **`particles`**

namespace **`pusher`**

density.param

Configure existing or define new normalized density profiles here.

During particle species creation in `speciesInitialization.param`, those profiles can be translated to spatial particle distributions.

This profile is normalized to units of `picongpu::SI::BASE_DENSITY_SI`, also defined in this file. Note that it only operates with physical density, and does not concern macroparticles. The number and sampling of macroparticles per cell are defined independently of a density profile. Please refer to documentation of *`picongpu::particles::CreateDensity`*<> for further details on this interaction.

Available profiles:

- HomogenousImpl : homogeneous density in whole simulation volume
- GaussianImpl<> : Gaussian profile in 'y', optionally with preceeding vacuum
- GaussianCloudImpl<> : Gaussian profile in all axes, optionally with preceeding vacuum in 'y'
- LinearExponentialImpl<> : linear ramping of density in 'y' into exponential slope after
- SphereFlanksImpl<> : composition of 1D profiles, each in form of exponential increasing flank, constant sphere, exponential decreasing flank

- EveryNthCellImpl<> : checkerboard profile matching the grid, particles are only present in cells with the given stride from one another in all directions
- FreeFormulaImpl<> : apply user-defined functor for calculating density, refer to `picongpu::densityProfiles::IProfile` for interface requirements
- FromOpenPMDImpl<> : load density values from a given file, requires openPMD API dependency

In the end, this file typically defines an alias for each density profile to be used. These aliases do not have to follow any naming convention, but serve as template parameters for invocations of *`picongpu::particles::CreateDensity`*<> in `speciesInitialization.param`.

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **`densityProfiles`**

Typedefs

using **`Gaussian`** = GaussianImpl<*GaussianParam*>

using **`Homogenous`** = HomogenousImpl

using **`LinearExponential`** = LinearExponentialImpl<*LinearExponentialParam*>

using **`GaussianCloud`** = GaussianCloudImpl<*GaussianCloudParam*>

using **`SphereFlanks`** = SphereFlanksImpl<*SphereFlanksParam*>

using **`EveryFourthCell`** = EveryNthCellImpl<mCT::UInt32<4, 4, 4>>

using **`FreeFormula`** = FreeFormulaImpl<*FreeFormulaFunctor*>

struct **GaussianParam**

Profile Formula: `const float_X exponent = abs((y - gasCenter_SI) / gasSigma_SI); const float_X density = exp(gasFactor * pow(exponent, gasPower));`

takes `gasCenterLeft_SI for y < gasCenterLeft_SI`, `gasCenterRight_SI for y > gasCenterRight_SI`, and `exponent = 0.0 for gasCenterLeft_SI < y < gasCenterRight_SI`

Public Static Attributes

static constexpr float_X **gasFactor** = -1.0

static constexpr float_X **gasPower** = 4.0

static constexpr uint32_t **vacuumCellsY** = 50

height of vacuum area on top border

this vacuum is important because of the laser initialization, which is done in the first cells of the simulation and assumes a charge-free volume unit: cells

static constexpr float_64 **gasCenterLeft_SI** = 4.62e-5

The central position of the distribution unit: meter.

static constexpr float_64 **gasCenterRight_SI** = 4.62e-5

static constexpr float_64 **gasSigmaLeft_SI** = 4.62e-5

the distance from gasCenter_SI until the gas density decreases to its 1/e-th part unit: meter

static constexpr float_64 **gasSigmaRight_SI** = 4.62e-5

struct **LinearExponentialParam**

parameter for `LinearExponential` profile

```
* Density Profile: /\
*                 /  -,_
*   linear       /      -,_    exponential
*   slope       /  |       -,_ slope
*                  MAX
*
```

Public Static Attributes

static constexpr uint32_t **vacuumCellsY** = 50

height of vacuum area on top border

this vacuum is important because of the laser initialization, which is done in the first cells of the simulation and assumes a charge-free volume unit: cells

static constexpr float_64 **gasYMax_SI** = 1.0e-3

Y-Position where the linear slope ends and the exponential slope begins unit: meter.

static constexpr float_64 **gasA_SI** = 1.0e-3

Parameters for the linear slope: For Y <= gasYMax_SI: \rho / BASE_DENSITY = A * Y + B = element [0.0; 1.0] unit for A: 1/m unit for B: none.

static constexpr float_64 **gasD_SI** = 1.0e-3

Parameters for the exponential slope For Y > gasYMax_SI: let Y' = Y - gasYMax_SI \rho = exp[- Y' * D] = element [0.0; 1.0] unit: 1/m.

static constexpr float_64 **gasB** = 0.0

struct **GaussianCloudParam**

Public Static Attributes

static constexpr float_X **gasFactor** = -0.5

Profile Formula: exponent = |globalCellPos - center| / sigma density = e^[gasFactor * exponent^gasPower].

static constexpr float_X **gasPower** = 2.0

static constexpr uint32_t **vacuumCellsY** = 50

height of vacuum area on top border

this vacuum is important because of the laser initialization, which is done in the first cells of the simulation and assumes a charge-free volume unit: cells

static constexpr floatD_64 **center_SI** = float3_64(1.134e-5, 1.134e-5, 1.134e-5).shrink<*simDim*>()

The central position of the gas distribution unit: meter.

static constexpr floatD_64 **sigma_SI** = float3_64(7.0e-6, 7.0e-6, 7.0e-6).shrink<*simDim*>()

the distance from gasCenter_SI until the gas density decreases to its 1/e-th part unit: meter

struct **SphereFlanksParam**

The profile consists out of the composition of 3 1D profiles with the scheme: exponential increasing flank, constant sphere, exponential decreasing flank.

```
*            ___
*  1D:  _,./   \.,_   rho(r)
*
*  2D:  ..,x,..   density: . low
*       .,xxx,.            , middle
*       ..,x,..            x high (constant)
*
```

Public Static Attributes

static constexpr uint32_t **vacuumCellsY** = 50

height of vacuum area on top border

this vacuum is important because of the laser initialization, which is done in the first cells of the simulation and assumes a charge-free volume unit: cells

static constexpr float_64 **r_SI** = 1.0e-3

Radius of the constant sphere unit: meter.

static constexpr float_64 **ri_SI** = 0.0

Inner radius if you want to build a shell/ring unit: meter.

static constexpr floatD_64 **center_SI** = float3_64(8.0e-3, 8.0e-3, 8.0e-3).shrink<*simDim*>()

Middle of the constant sphere unit: meter.

static constexpr float_64 **exponent_SI** = 1.0e3

Parameters for the exponential slope For distance > r_SI: let distance' = distance - r \rho = exp[- distance' * exponent] unit: 1/m.

struct **FromOpenPMDParam**

Density values taken from an openPMD file.

The density values must be a scalar dataset of type float_X, type mismatch would cause errors. This implementation would ignore all openPMD metadata but axisLabels. Each value in the dataset defines density in the cell with the corresponding total coordinate minus the given offset. When the functor is instantiated, it will load the part matching the current domain position. Density in points not present in the file would be set to the given default density. Dimensionality of the file indexing must match the simulation dimensionality. Density values are in BASE_DENSITY_SI units.

Public Static Attributes

static constexpr char const ***filename** = "density.h5"

Path to the openPMD input file.

This value can alternatively be controlled at runtime by setting it to "" here and providing command-line option <species>_runtimeDensityFile. Refer to description of this command-line option for details. Note that runtime option only exists for this parameter, and not others in this struct.

File-based iteration format is also supported, with the usual openPMD API naming scheme.

It is recommended to use a full path to make it independent of how PIConGPU is launched. Relative paths require consistency to the current directory when PIConGPU is started. With tbg and the standard .tpl files, relative to the resulting simOutput directory.

static constexpr char const ***datasetName** = "e_density"

Name of the openPMD dataset inside the file.

By default, this dataset indexing is assumed to be in (x, y, z) coordinates. This can be changed by setting openPMD attribute "axisLabels" of the correponding dataset. For example, PIConGPU output uses z-y-x via this attribute, and that automatically works. Note that only C dataOrder is supported.

Warning: it is only the dataset itself, a simple text name and not something like "/[directories]/density/[iteration]/fields/e_density".

static constexpr uint32_t **iteration** = 0

Iteration inside the file (only file, not related to the current simulation time iteration)

static constexpr DataSpace<*simDim*> **offset** = DataSpace<DIM3>(0, 0, 0).shrink<*simDim*>()

Offset in the file in cells: each value is density at (total cell index - offset)

This offset is in (x, y, z) coordinates. Positive offset means the file values "start" from index == offset in the total coordinates. Negative offset is also supported.

static constexpr float_X **defaultDensity** = 0.0_X

Default value to be used for cells with no corresponding file value.

struct **FreeFormulaFunctor**

Public Functions

`inline HDINLINE float_X operator() (const floatD_64 &position_SI, const float3_64 &cellSize_SI)`

This formula uses SI quantities only.

The profile will be multiplied by BASE_DENSITY_SI.

Parameters

- **`position_SI`** – total offset including all slides [meter]
- **`cellSize_SI`** – cell sizes [meter]

Returns

float_X density [normalized to 1.0]

namespace **`SI`**

Variables

constexpr float_64 **`BASE_DENSITY_SI`** = 1.e25

Base density in particles per m^3 in the density profiles.

This is often taken as reference maximum density in normalized profiles. Individual particle species can define a `densityRatio` flag relative to this value.

unit: ELEMENTS/m^3

speciesAttributes.param

This file defines available attributes that can be stored with each particle of a particle species.

Each attribute defined here needs to implement furthermore the traits

- Unit
- UnitDimension
- WeightingPower
- MacroWeighted in speciesAttributes.unitless for further information about these traits see therein.

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. “Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences.” Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. “A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter” High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Functions

`alias`(position)

relative (to cell origin) in-cell position of a particle

With this definition we do not define any type like float3_X, float3_64, ... This is only a name without a specialization.

`value_identifier`(uint64_t, particleId, IdProvider<*simDim*>::getNewId())

unique identifier for a particle

`value_identifier (floatD_X, position_pic, floatD_X::create(0.))`

specialization for the relative in-cell position

`value_identifier (float3_X, momentum, float3_X::create(0.))`

momentum at timestep t

`value_identifier (float3_X, momentumPrev1, float3_X::create(0._X))`

momentum at (previous) timestep t-1

`value_identifier (float_X, weighting, 0._X)`

weighting of the macro particle

`value_identifier (float_X, weightingDampingFactor, 1._X)`

optional damping factor for weighting of the macro particle

This factor is only used for current deposition with absorbing particle boundaries and PML. In case this attribute is defined, affected particles would have their weightings damped only for that procedure. Otherwise, the damping would affect weighting directly and so potentially overall properties like density. It should be generally not a considerable issue, except lots of particles move from PML to internal area.

Thus, for efficiency reasons it is recommended to not enable this attribute by default.

`value_identifier (float3_X, probeE, float3_X::create(0.))`

interpolated electric field with respect to particle shape

Attribute can be added to any species.

`value_identifier (float3_X, probeB, float3_X::create(0.))`

interpolated magnetic field with respect to particle shape

Attribute can be added to any species.

`value_identifier (bool, radiationMask, false)`

masking a particle for radiation

The mask is used by the user defined filter `RadiationParticleFilter` in radiation.param to (de)select particles for the radiation calculation.

`value_identifier (bool, transitionRadiationMask, false)`

masking a particle for transition radiation

The mask is used by the user defined filter `TransitionRadiationParticleFilter` in transitionRadiation.param to (de)select particles for the transition radiation calculation.

`value_identifier (float_X, boundElectrons, 0._X)`

number of electrons bound to the atom / ion

value type is float_X to avoid casts during the runtime

- float_X instead of integer types are reasonable because effective charge numbers are possible
- required for ion species if ionization is enabled
- setting it requires atomicNumbers to also be set

Todo:

connect default to proton number

`alias`(atomicLevels)

alias for one of two representations of the atomic state, see also atomicPhysics.param

`alias`(atomicConfigNumber)

alias for atomic state of ion, see also atomicPhysics.param

`alias`(atomicPhysicsSolver)

`value_identifier`(DataSpace<*simDim*>, totalCellIdx, DataSpace<*simDim*>())

Total cell index of a particle.

The total cell index is a N-dimensional DataSpace given by a GPU's `globalDomain.offset` + `localDomain.offset` added to the N-dimensional cell index the particle belongs to on that GPU.

`alias`(shape)

alias for particle shape, see also species.param

`alias`(particlePusher)

alias for particle pusher, see alsospecies.param

`alias`(ionizers)

alias for particle ionizers, see also ionizer.param

`alias`(ionizationEnergies)

alias for ionization energy container, see also ionizationEnergies.param

`alias`(interpolation)

alias for particle to field interpolation, see also species.param

`alias`(current)

alias for particle current solver, see also species.param

`alias`(atomicNumbers)

alias for particle flag: atomic numbers, see also ionizer.param

- only reasonable for atoms / ions / nuclei
- is required when boundElectrons is set

alias(effectiveNuclearCharge)

alias for particle flag: effective nuclear charge,

- see also ionizer.param
- only reasonable for atoms / ions / nuclei

alias(massRatio)

alias for particle mass ratio

mass ratio between base particle, see also speciesConstants.param `SI::BASE_MASS_SI` and a user defined species

default value: 1.0 if unset

alias(chargeRatio)

alias for particle charge ratio

charge ratio between base particle, see also speciesConstants.param `SI::BASE_CHARGE_SI` and a user defined species

default value: 1.0 if unset

alias(densityRatio)

alias for particle density ratio

density ratio between default density, see also density.param `SI::BASE_DENSITY_SI` and a user defined species

default value: 1.0 if unset

alias(exchangeMemCfg)

alias to reserved bytes for each communication direction

This is an optional flag and overwrites the default species configuration in memory.param.

A memory config must be of the following form:

```
struct ExampleExchangeMemCfg
{
    static constexpr uint32_t BYTES_EXCHANGE_X = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_EXCHANGE_Y = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_EXCHANGE_Z = 5 * 1024 * 1024;
    static constexpr uint32_t BYTES_CORNER = 16 * 1024;
    static constexpr uint32_t BYTES_EDGES = 16 * 1024;
    using REF_LOCAL_DOM_SIZE = mCT::Int<0, 0, 0>;
    const std::array<float_X, 3> DIR_SCALING_FACTOR = {{0.0, 0.0, 0.0}};
};
```

alias(boundaryCondition)

alias to specify the internal pmacc boundary treatment for particles

It controls the internal behavior and intented for special cases only. To set physical boundary conditions for a species, instead use <species>_boundary command-line option.

The default behavior if this alias is not given to a species is to do nothing. The existing boundary implementations already take care of the particles leaving the global simulation volume.

The following species attributes are defined by PMacc and always stored with a particle:

namespace **pmacc**

this specifies how an object of the ConfigNumber class can be written to an external file for storage.

Functions

`value_identifier (lcellId_t, localCellIdx, 0)`

cell of a particle inside a supercell

Value is a linear cell index inside the supercell

`value_identifier (uint8_t, multiMask, 0)`

state of a particle

Particle might be valid or invalid in a particle frame. Valid particles can further be marked as candidates to leave a supercell. Possible multiMask values are:

- 0 (zero): no particle (invalid)
- 1: particle (valid)
- 2 to 27: (valid) particle that is about to leave its supercell but is still stored in the current particle frame. Directions to leave the supercell are defined as follows. An ExchangeType = value - 1 (e.g. 27 - 1 = 26) means particle leaves supercell in the direction of FRONT(value=18) && TOP(value=6) && LEFT(value=2) which defines a diagonal movement over a supercell corner (18+6+2=26).

speciesConstants.param

Constants and thresholds for particle species.

Defines the reference mass and reference charge to express species with (default: electrons with negative charge).

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Variables

constexpr float_X **GAMMA_THRESH** = 1.005_X

Threshold between relativistic and non-relativistic regime.

Threshold used for calculations that want to separate between high-precision formulas for relativistic and non-relativistic use-cases, e.g. energy-binning algorithms.

constexpr float_X **GAMMA_INV_SQUARE_RAD_THRESH** = 0.18_X

Threshold in radiation plugin between relativistic and non-relativistic regime.

This limit is used to decide between a pure 1-sqrt(1-x) calculation and a 5th order Taylor approximation of 1-sqrt(1-x) to avoid halving of significant digits due to the sqrt() evaluation at x = 1/gamma^2 near 0.0. With 0.18 the relative error between Taylor approximation and real value will be below 0.001% = 1e-5 * for x=1/gamma^2 < 0.18

namespace **SI**

Variables

constexpr float_64 **BASE_MASS_SI** = *ELECTRON_MASS_SI*

base particle mass

reference for massRatio in speciesDefinition.param

unit: kg

constexpr float_64 **BASE_CHARGE_SI** = *ELECTRON_CHARGE_SI*

base particle charge

reference for chargeRatio in speciesDefinition.param

unit: C

species.param

Particle shape, field to particle interpolation, current solver, and particle pusher can be declared here for usage in `speciesDefinition.param`.

See also:

MODELS / Hierarchy of Charge Assignment Schemes in the online documentation for information on particle shapes.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  -
    R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  -
    H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Typedefs

using `UsedParticleShape` = *particles*::shapes::TSC

select macroparticle shape

WARNING the shape names are redefined and diverge from PIConGPU versions before 0.6.0.

- particles::shapes::CIC : Assignment function is a piecewise linear spline
- particles::shapes::TSC : Assignment function is a piecewise quadratic spline
- particles::shapes::PQS : Assignment function is a piecewise cubic spline
- particles::shapes::PCS : Assignment function is a piecewise quartic spline

using `UsedField2Particle` = FieldToParticleInterpolation<*UsedParticleShape*, AssignedTrilinearInterpolation>

select interpolation method to be used for interpolation of grid-based field values to particle positions

using `UsedParticleCurrentSolver` = currentSolver::Esirkepov<*UsedParticleShape*>

select current solver method

- currentSolver::Esirkepov< SHAPE, STRATEGY > : particle shapes - CIC, TSC, PQS, PCS (1st to 4th order)
- currentSolver::EmZ< SHAPE, STRATEGY > : particle shapes - CIC, TSC, PQS, PCS (1st to 4th order)
- currentSolver::EZ< SHAPE, STRATEGY > : same as EmZ (just an alias to match the currently used naming)

STRATEGY (optional):

- currentSolver::strategy::StridedCachedSupercells
- currentSolver::strategy::StridedCachedSupercellsScaled<N> with N >= 1
- currentSolver::strategy::CachedSupercells
- currentSolver::strategy::CachedSupercellsScaled<N> with N >= 1
- currentSolver::strategy::NonCachedSupercells
- currentSolver::strategy::NonCachedSupercellsScaled<N> with N >= 1

using `UsedParticlePusher` = *particles*::*pusher*::Boris

particle pusher configuration

Defining a pusher is optional for particles

- particles::pusher::HigueraCary : Higuera & Cary's relativistic pusher preserving both volume and ExB velocity

- particles::pusher::Vay : Vay's relativistic pusher preserving ExB velocity
- particles::pusher::Boris : Boris' relativistic pusher preserving volume
- particles::pusher::ReducedLandauLifshitz : 4th order RungeKutta pusher with classical radiation reaction
- particles::pusher::Composite : composite of two given pushers, switches between using one (or none) of those

For diagnostics & modeling: ——————————————————&#8212;

- particles::pusher::Acceleration : Accelerate particles by applying a constant electric field
- particles::pusher::Free : free propagation, ignore fields (= free stream model)
- particles::pusher::Photon : propagate with c in direction of normalized mom.
- particles::pusher::Probe : Probe particles that interpolate E & B For development purposes: ——————————————————&#8212;
- particles::pusher::Axel : a pusher developed at HZDR during 2011 (testing)

Current solver details.

speciesDefinition.param

Define particle species.

This file collects all previous declarations of base (reference) quantities and configured solvers for species and defines particle species. This includes "attributes" (lvalues to store with each species) and "flags" (rvalues & aliases for solvers to perform with the species for each timestep and ratios to base quantities). With those information, a `Particles` class is defined for each species and then collected in the list `VectorAllSpecies`.

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Typedefs

using **DefaultParticleAttributes** = MakeSeq_t<position<position_pic>, momentum, weighting>
    describe attributes of a particle

using **ParticleFlagsElectrons** = MakeSeq_t<particlePusher<*UsedParticlePusher*>, shape<*UsedParticleShape*>, interpolation<*UsedField2Particle*>, current<*UsedParticleCurrentSolver*>, massRatio<MassRatioElectrons>, chargeRatio<ChargeRatioElectrons>>

using **PIC_Electrons** = *Particles*<PMACC_CSTRING("e"), *ParticleFlagsElectrons*, *DefaultParticleAttributes*>

using **ParticleFlagsIons** = MakeSeq_t<particlePusher<*UsedParticlePusher*>, shape<*UsedParticleShape*>, interpolation<*UsedField2Particle*>, current<*UsedParticleCurrentSolver*>, massRatio<MassRatioIons>, chargeRatio<ChargeRatioIons>, densityRatio<DensityRatioIons>, atomicNumbers<*ionization*::*atomicNumbers*::*Hydrogen_t*>>

using **PIC_Ions** = *Particles*<PMACC_CSTRING("i"), *ParticleFlagsIons*, *DefaultParticleAttributes*>

using **VectorAllSpecies** = MakeSeq_t<*PIC_Electrons*, *PIC_Ions*>
    All known particle species of the simulation.

    List all defined particle species from above in this list to make them available to the PIC algorithm.

Functions

value_identifier (float_X, MassRatioElectrons, 1.0)

value_identifier (float_X, ChargeRatioElectrons, 1.0)

value_identifier (float_X, MassRatioIons, 1836.152672)

value_identifier (float_X, ChargeRatioIons, -1.0)

value_identifier (float_X, DensityRatioIons, 1.0)

particle.param

Configurations for particle manipulators.

Set up and declare functors that can be used in speciesInitialization.param for particle species initialization and manipulation, such as temperature distributions, drifts, pre-ionization and in-cell position.

namespace **picongpu**
    rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **particles**

Variables

constexpr float_X **MIN_WEIGHTING** = 10.0

a particle with a weighting below MIN_WEIGHTING will not be created / will be deleted

unit: none

constexpr uint32_t **TYPICAL_PARTICLES_PER_CELL** = 2u

(Approximate) Number of maximum macro-particles per cell.

Used internally for unit normalization. And used in startPosition functors further below to set real maximum number of macro-particles per cell.

namespace **manipulators**

Typedefs

using **AssignXDrift** = unary::*Drift*<*DriftParam*, *pmacc*::math::operation::Assign>

Definition of manipulator that assigns a drift in X using parameters from struct DriftParam.

using **AddTemperature** = unary::*Temperature*<*TemperatureParam*>

Definition of manipulator assigning a temperature using parameters from struct TemperatureParam.

using **AddFreeTemperature** = unary::*FreeTemperature*<*TemperatureFunctor*>

Definition of manipulator that assigns a temperature according to the functor defined by struct TemperaturFunctor.

using **DoubleWeighting** = generic::*Free*<*DoubleWeightingFunctor*>

Definition of the free particle manipulator which doubles each weighting.

using **RandomEnabledRadiation** = generic::*FreeRng*<*RandomEnabledRadiationFunctor*, *pmacc*::random::distributions::Uniform<float_X>>

Definition of manipulator that selects macro-particles for Radiation plugin.

using **RandomPosition** = unary::*RandomPosition*

Definition of manipulator that changes the in-cell position of each particle of a species.

using **SetNeutral** = unary::ChargeState<0u>

definition of manipulator that sets the boundElectron attribute(charge state) of each particle of an ion of a species to neutral

struct **DriftParam**

Define Lorentz factor of initial particle drift.

Public Static Attributes

static constexpr float_64 **gamma** = 1.0

static constexpr auto **driftDirection** = float3_X(1.0, 0.0, 0.0)

Define initial particle drift direction vector.

struct **TemperatureParam**

Define initial particle temperature.

Public Static Attributes

static constexpr float_64 **temperature** = 0.0

Initial temperature unit: keV.

struct **TemperatureFunctor**

Define initial particle temperature as a function of position.

This is a functor which needs to follow the requirements of param::TemperatureFunctor.

Public Functions

inline **TemperatureFunctor**()

Constructor, can take currentStep or no parameters (can also be auto-generated by a compiler)

Parameters

currentStep – current time iteration

inline HDINLINE float_X operator() (const DataSpace< simDim > &totalCellOffset)

Return the temperature in keV for the given position.

Return type may be float_X or float_64.

Parameters

totalCellOffset – total offset including all slides [in cells]

struct **DoubleWeightingFunctor**

Unary particle manipulator: double each weighting.

Public Functions

template<typename T_Particle> inline DINLINE void operator() (T_Particle &particle

struct **RandomEnabledRadiationFunctor**

Define mask which randomly marks macro-particles used by the radiation plugin to calculate far field radiation.

Public Functions

template<typename T_Rng,
typename T_Particle> inline DINLINE void operator() (T_Rng &rng,
T_Particle &particle)

namespace **startPosition**

Typedefs

using **Random** = RandomImpl<*RandomParameter*>

Definition of start position functor that randomly distributes macro-particles within a cell.

using **RandomPositionAndWeighting** = RandomPositionAndWeightingImpl<*RandomParameter*>

Definition of start position functor that randomly distributes macro-particles within a cell and randomly distributes weightings across macro-particles within a cell.

using **Quiet** = QuietImpl<*QuietParam*>

Definition of Quiet start position functor that positions macro-particles regularly on the grid.

No random number generator used.

using **OnePosition** = OnePositionImpl<*OnePositionParameter*>

Definition of OnePosition start position functor that places macro-particles at the initial in-cell position defined above.

struct **RandomParameter**

Define target number for marco-particles per cell to be used in Random start position functor.

Public Static Attributes

static constexpr uint32_t **numParticlesPerCell** = *TYPICAL_PARTICLES_PER_CELL*

Maximum number of macro-particles per cell during density profile evaluation.

Determines the weighting of a macro particle as well as the number of macro-particles which sample the evolution of the particle distribution function in phase space.

unit: none

struct **QuietParam**

Define target number for marco-particles per cell along a direction.

To be used in Quiet start position functor.

Here, one macro-particle per cell along x, one macro-particle per cell along z, and TYPICAL_PARTICLES_PER_CELL macro-particles per cell along y.

Vector is automatically reduced to two dimensions for 2D (x,y) simulations.

Public Types

using **numParticlesPerDimension** = mCT::shrinkTo<mCT::Int<1, *TYPICAL_PARTICLES_PER_CELL*, 1>, *simDim*>::type

Count of macro-particles per cell per direction at initial state.

unit: none

struct **OnePositionParameter**

Configuration of initial in-cell particle position.

Here, macro-particles sit directly in lower corner of the cell.

Public Static Attributes

static constexpr uint32_t **numParticlesPerCell** = *TYPICAL_PARTICLES_PER_CELL*

Maximum number of macro-particles per cell during density profile evaluation.

Determines the weighting of a macro particle as well as the number of macro-particles which sample the evolution of the particle distribution function in phase space.

unit: none

static constexpr auto **inCellOffset** = float3_X(0., 0., 0.)

each x, y, z in-cell position component in range [0.0, 1.0)

in 2D the last component is ignored

More details on the order of initialization of particles inside a particle species *can be found here*.

List of all pre-defined particle manipulators.

unit.param

In this file we define typical scales for normalization of physical quantities aka "units".

Usually, a user would not change this file but might use the defined constants in other input files.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Variables

constexpr float_64 **UNIT_TIME** = *SI*::*DELTA_T_SI*

Unit of time.

constexpr float_64 **UNIT_LENGTH** = *UNIT_TIME* * *UNIT_SPEED*

Unit of length.

constexpr float_64 **UNIT_MASS** = *SI*::*BASE_MASS_SI* * *particles*::*TYPICAL_NUM_PARTICLES_PER_MACROPARTICLE*

Unit of mass.

constexpr float_64 **UNIT_CHARGE** = -1.0 * *SI*::*BASE_CHARGE_SI* * *particles*::*TYPICAL_NUM_PARTICLES_PER_MACROPARTICLE*

Unit of charge.

constexpr float_64 **UNIT_ENERGY** = (*UNIT_MASS* * *UNIT_LENGTH* * *UNIT_LENGTH* / (*UNIT_TIME* * *UNIT_TIME*))

Unit of energy.

constexpr float_64 **UNIT_EFIELD** = 1.0 / (*UNIT_TIME* * *UNIT_TIME* / *UNIT_MASS* / *UNIT_LENGTH* * *UNIT_CHARGE*)

Unit of EField: V/m.

constexpr float_64 **UNIT_BFIELD** = (*UNIT_MASS* / (*UNIT_TIME* * *UNIT_CHARGE*))

namespace **particles**

Variables

constexpr float_64 **TYPICAL_NUM_PARTICLES_PER_MACROPARTICLE** = (*SI*::*BASE_DENSITY_SI* * *SI*::*CELL_WIDTH_SI* * *SI*::*CELL_HEIGHT_SI* * *SI*::*CELL_DEPTH_SI*) / float_64(*particles*::*TYPICAL_PARTICLES_PER_CELL*)

Typical number of particles per macro particle (= typical macro particle weighting) unit: none.

particleFilters.param

A common task in both modeling and in situ processing (output) is the selection of particles of a particle species by attributes.

Users can define such selections as particle filters in this file.

Particle filters are simple mappings assigning each particle of a species either `true` or `false` (ignore / filter out).

All active filters need to be listed in `AllParticleFilters`. They are then combined with `VectorAllSpecies` at compile-time, e.g. for plugins.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  -

    R. Mewe. “Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences.” Astronomy and Astrophysics 20, 215 (1972)

–

H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **`particles`**

namespace **`filter`**

Typedefs

using **`AllParticleFilters`** = MakeSeq_t<*All*>

Plugins: collection of all available particle filters.

Create a list of all filters here that you want to use in plugins.

Note: filter *All* is defined in picongpu/particles/filter/filter.def

namespace **`traits`**

List of all pre-defined particle filters.

speciesInitialization.param

Initialize particles inside particle species.

This is the final step in setting up particles (defined in `speciesDefinition.param`) via density profiles (defined in `density.param`). One can then further derive particles from one species to another and manipulate attributes with "manipulators" and "filters" (defined in `particle.param` and `particleFilters.param`).

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published

  –

  R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

  the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)

  –

  H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

  highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **particles**

Typedefs

using **InitPipeline** = *pmacc*::mp_list<>

InitPipeline defines in which order species are initialized.

the functors are called in order (from first to last functor). The functors must be default-constructible and take the current time iteration as the only parameter.

List of all initialization methods for particle species.

Particles

Particles are defined in modular steps. First, species need to be generally defined in *speciesDefinition.param*. Second, species are initialized with particles in *speciesInitialization.param*.

The following operations can be applied in the `picongpu::particles::InitPipeline` of the latter:

Initialization

CreateDensity

template<typename **T_DensityFunctor**, typename **T_PositionFunctor**, typename **T_SpeciesType** = boost::mpl::_1>
struct **CreateDensity**

Sample macroparticles according to the given spatial density profile.

Create macroparticles inside a species.

This function only concerns the number of macroparticles, positions, and weighting. So it basically performs sampling in the coordinate space, while not initializing other attributes. When needed, those should be set (for then-existing macroparticles) by subsequently calling *Manipulate*.

User input to this functor is two-fold. T_DensityFunctor represents spatial density of real particles, normalized according to our requirements. It describes the physical setup being simulated and only deals with real, not macro-, particles. T_PositionFunctor is more of a PIC simulation parameter. It defines how real particles in each cell will be represented with macroparticles. This concerns the count, weighting, and in-cell positions of the created macroparticles.

The sampling process operates independently for each cell, as follows:

- Evaluate the amount of real particles in the cell, Nr, using T_DensityFunctor.
- If Nr > 0, decide how to represent it with macroparticles using T_PositionFunctor:
  - (For simplicity we describe how all currently used functors but RandomPositionAndWeightingImpl and RandomBinomialImpl operate, see below for customization)

  - Try to have exactly T_PositionFunctor::numParticlesPerCell macroparticles with same weighting w = Nr / T_PositionFunctor::numParticlesPerCell.
  - If such w < MIN_WEIGHTING, instead use fewer macroparticles and higher weighting.
  - In any case the combined weighting of all new macroparticles will match Nr.
- Create the selected number of macroparticles with selected weighting.
- Set in-cell positions according to T_PositionFunctor.

In principle, one could override the logic inside the (If Nr > 0) block by implementing a custom functor. Then one could have an arbitrary number of macroparticles and weight distribution between them. The only requirement is that together it matches Nr. For an example of non-uniform weight distribution

See also:

startPosition::RandomPositionAndWeightingImpl. Note that in this scheme almost all non-vacuum cells will start with the same number of macroparticles. Having a higher density in a cell would mean larger weighting, but not more macroparticles.

Note: *FillAllGaps* is automatically called after creation.

Template Parameters

- **`T_DensityFunctor`** – unary lambda functor with profile description, see density.param, example: picongpu::particles::densityProfiles::Homogenous
- **`T_PositionFunctor`** – unary lambda functor with position description and number of macroparticles per cell, see particle.param, examples: picongpu::particles::startPosition::Quiet, picongpu::particles::startPosition::Random
- **`T_SpeciesType`** – type or name as PMACC_CSTRING of the used species, see speciesDefinition.param

Derive

template<typename **`T_SrcSpeciesType`**, typename **`T_DestSpeciesType`** = boost::mpl::_1, typename **`T_Filter`** = *filter*::*All*>
struct **`Derive`** : public *picongpu*::*particles*::*ManipulateDerive*<*manipulators*::generic::None, *T_SrcSpeciesType*, boost::mpl::_1, *filter*::*All*>

Generate particles in a species by deriving from another species' particles.

Create particles in `T_DestSpeciesType` by deriving (copying) all particles and their matching attributes (except `particleId`) from `T_SrcSpeciesType`.

Note: *FillAllGaps* is called on on `T_DestSpeciesType` after the derivation is finished.

Template Parameters

- **`T_SrcSpeciesType`** – type or name as PMACC_CSTRING of the source species
- **`T_DestSpeciesType`** – type or name as PMACC_CSTRING of the destination species
- **`T_Filter`** – picongpu::particles::filter, particle filter type to select source particles to derive

Manipulate

template<typename **`T_Manipulator`**, typename **`T_Species`** = boost::mpl::_1, typename **`T_Filter`** = *filter*::*All*, typename **`T_Area`** = std::integral_constant<uint32_t, CORE + BORDER>>
struct **`Manipulate`** : public *pmacc*::particles::algorithm::CallForEach<*pmacc*::particles::meta::FindByNameOrType<*VectorAllSpecies*, boost::mpl::_1>, detail::MakeUnaryFilteredFunctor<*T_Manipulator*, boost::mpl::_1, *filter*::*All*>, *T_Area*::value>

Run a user defined manipulation for each particle of a species in an area.

Allows to manipulate attributes of existing particles in a species with arbitrary unary functors ("manipulators").

Provides two versions of operator() to either operate on T_Area or a custom area,

See also:

pmacc::particles::algorithm::CallForEach.

See also:

picongpu::particles::manipulators

Warning: Does NOT call *FillAllGaps* after manipulation! If the manipulation deactivates particles or creates "gaps" in any other way, *FillAllGaps* needs to be called for the `T_Species` manually in the next step!

Template Parameters

- **`T_Manipulator`** – unary lambda functor accepting one particle species,
- **`T_Species`** – type or name as PMACC_CSTRING of the used species
- **`T_Filter`** – picongpu::particles::filter, particle filter type to select particles in `T_Species` to manipulate
- **`T_Area`** – area to process particles in operator()(currentStep), wrapped into std::integral_constant for boost::mpl::apply to work; does not affect operator()(currentStep, areaMapperFactory)

ManipulateDerive

template<typename **`T_Manipulator`**, typename **`T_SrcSpeciesType`**, typename **`T_DestSpeciesType`** = boost::mpl::_1, typename **`T_SrcFilter`** = *filter*::*All*>
struct **`ManipulateDerive`**

Generate particles in a species by deriving and manipulating from another species' particles.

Create particles in `T_DestSpeciesType` by deriving (copying) all particles and their matching attributes (except `particleId`) from `T_SrcSpeciesType`. During the derivation, the particle attributes in can be manipulated with `T_ManipulateFunctor`.

See also:

picongpu::particles::manipulators

Note: *FillAllGaps* is called on on T_DestSpeciesType after the derivation is finished. If the derivation also manipulates the T_SrcSpeciesType, e.g. in order to deactivate some particles for a move, *FillAllGaps* needs to be called for the T_SrcSpeciesType manually in the next step!

Template Parameters

- **T_Manipulator** – a pseudo-binary functor accepting two particle species: destination and source,
- **T_SrcSpeciesType** – type or name as PMACC_CSTRING of the source species
- **T_DestSpeciesType** – type or name as PMACC_CSTRING of the destination species
- **T_SrcFilter** – picongpu::particles::filter, particle filter type to select particles in T_SrcSpeciesType to derive into T_DestSpeciesType

FillAllGaps

template<typename **T_SpeciesType** = boost::mpl::_1>

struct **FillAllGaps**

Generate a valid, contiguous list of particle frames.

Some operations, such as deactivating or adding particles to a particle species can generate "gaps" in our internal particle storage, a list of frames.

This operation copies all particles from the end of the frame list to "gaps" in the beginning of the frame list. After execution, the requirement that all particle frames must be filled contiguously with valid particles and that all frames but the last are full is fulfilled.

Template Parameters

T_SpeciesType – type or name as PMACC_CSTRING of the particle species to fill gaps in memory

Manipulation Functors

Some of the particle operations above can take the following functors as arguments to manipulate attributes of particle species. A particle filter (see following section) is used to only manipulated selected particles of a species with a functor.

Free

template<typename **T_Functor**>

struct **Free** : protected *picongpu*::*particles*::functor::User<*T_Functor*>

call simple free user defined manipulators

example for `particle.param`: set in cell position to zero

```
struct FunctorInCellPositionZero
{
    template< typename T_Particle >
    HDINLINE void operator()( T_Particle & particle )
    {
        particle[ position_ ] = floatD_X::create( 0.0 );
    }
    static constexpr char const * name = "inCellPositionZero";
};

using InCellPositionZero = generic::Free<
   FunctorInCellPositionZero
>;
```

Template Parameters

T_Functor – user defined manipulators **optional**: can implement **one** host side constructor `T_Functor()` or `T_Functor(uint32_t currentTimeStep)`

FreeRng

template<typename **T_Functor**, typename **T_Distribution**>

struct **FreeRng** : protected *picongpu*::*particles*::functor::User<*T_Functor*>, private *picongpu*::*particles*::functor::misc::Rng<*T_Distribution*>

call simple free user defined functor and provide a random number generator

example for `particle.param`: add

```
#include <pmacc/random/distributions/Uniform.hpp>

struct FunctorRandomX
{
    template< typename T_Rng, typename T_Particle >
    HDINLINE void operator()( T_Rng& rng, T_Particle& particle )
    {
        particle[ position_ ].x() = rng();
    }
    static constexpr char const * name = "randomXPos";
};

using RandomXPos = generic::FreeRng<
   FunctorRandomX,
   pmacc::random::distributions::Uniform< float_X >
>;
```

and to `InitPipeline` in `speciesInitialization.param`:

```
Manipulate< manipulators::RandomXPos, SPECIES_NAME >
```

Template Parameters

- **T_Functor** – user defined unary functor
- **T_Distribution** – pmacc::random::distributions, random number distribution

FreeTotalCellOffset

template<typename **T_Functor**>

struct **FreeTotalCellOffset** : protected *picongpu*::*particles*::functor::User<*T_Functor*>, private *picongpu*::*particles*::functor::misc::TotalCellOffset

call simple free user defined manipulators and provide the cell information

The functor passes the cell offset of the particle relative to the total domain origin into the functor.

example for `particle.param`: set a user-defined species attribute y0 (type: uint32_t) to the current total y-cell index

```
struct FunctorSaveYcell
{
    template< typename T_Particle >
    HDINLINE void operator()(
       DataSpace< simDim > const & particleOffsetToTotalOrigin,
       T_Particle & particle
    )
    {
        particle[ y0_ ] = particleOffsetToTotalOrigin.y();
    }
    static constexpr char const * name = "saveYcell";
};

using SaveYcell = unary::FreeTotalCellOffset<
   FunctorSaveYcell
>;
```

Template Parameters

`T_Functor` – user defined unary functor

CopyAttribute

template<typename **`T_DestAttribute`**, typename **`T_SrcAttribute`**>

using *`picongpu`*`::`*`particles`*`::`*`manipulators`*`::unary::`**`CopyAttribute`** = generic::*Free*<acc::CopyAttribute<*T_DestAttribute*, *T_SrcAttribute*>>

copy a particle source attribute to a destination attribute

This is an unary functor and operates on one particle.

Template Parameters

- **`T_DestAttribute`** – type of the destination attribute e.g. `momentumPrev1`
- **`T_SrcAttribute`** – type of the source attribute e.g. `momentum`

Drift

template<typename **`T_ParamClass`** = param::DriftCfg, typename **`T_ValueFunctor`** = *pmacc*::math::operation::Add>
using *`picongpu`*`::`*`particles`*`::`*`manipulators`*`::unary::`**`Drift`** = generic::*Free*<acc::Drift<*T_ParamClass*, *T_ValueFunctor*>>

change particle's momentum based on speed

allow to manipulate a speed to a particle

Template Parameters

- **`T_ParamClass`** – param::DriftCfg, configuration parameter
- **`T_ValueFunctor`** – pmacc::math::operation::*, binary functor type to manipulate the momentum attribute

RandomPosition

using *picongpu*::*particles*::*manipulators*::unary::**RandomPosition** = generic::*FreeRng*<acc::RandomPosition, *pmacc*::random::distributions::Uniform<float_X>>

Change the in cell position.

This functor changes the in-cell position of a particle. The new in-cell position is uniformly distributed position between [0.0;1.0).

example: add

```
particles::Manipulate<RandomPosition,SPECIES_NAME>
```

to `InitPipeline` in `speciesInitialization.param`

Temperature

template<typename **T_ParamClass** = param::TemperatureCfg, typename **T_ValueFunctor** = *pmacc*::math::operation::Add>
using *picongpu*::*particles*::*manipulators*::unary::**Temperature** = generic::*FreeRng*<acc::Temperature<*T_ParamClass*, *T_ValueFunctor*>, *pmacc*::random::distributions::Normal<float_X>>

Modify particle momentum based on temperature.

Sample a random momentum value distributed according to the given temperature and add it to the existing particle momentum.

Note: initial electron temperature should generally be chosen so that the resulting Debye length is resolved by the grid.

Template Parameters

- **T_ValueFunctor** – pmacc::math::operation::*, binary functor type to add a new momentum to an old one Version with fixed temperature given via parameter struct
- **T_ParamClass** – configuration parameter, follows requirements of param::TemperatureCfg

template<typename **T_TemperatureFunctor** = param::TemperatureFunctor, typename **T_ValueFunctor** = *pmacc*::math::operation::Add>
using *picongpu*::*particles*::*manipulators*::unary::**FreeTemperature** = FreeTotalCellOffsetRng<acc::FreeTemperature<*T_TemperatureFunctor*, *T_ValueFunctor*>, *pmacc*::random::distributions::Normal<float_X>>

Version with user-provided temperature functor.

Template Parameters

T_TemperatureFunctor – temperature functor, follows requirements of param::TemperatureFunctor

Assign

using *picongpu*::*particles*::*manipulators*::binary::**Assign** = generic::*Free*<acc::Assign>

assign attributes of one particle to another

Can be used as binary and higher order operator but only the first two particles are used for the assign operation.

Assign all matching attributes of a source particle to the destination particle. Attributes that only exist in the destination species are initialized with the default value. Attributes that only exists in the source particle will be ignored.

DensityWeighting

using *picongpu*::*particles*::*manipulators*::binary::**DensityWeighting** = generic::*Free*<acc::DensityWeighting>

Re-scale the weighting of a cloned species by densityRatio.

When deriving species from each other, the new species "inherits" the macro-particle weighting of the first one. This functor can be used to manipulate the weighting of the new species' macro particles to satisfy the input densityRatio of it.

note: needs the densityRatio flag on both species, used by the GetDensityRatio trait.

ProtonTimesWeighting

using *picongpu*::*particles*::*manipulators*::binary::**ProtonTimesWeighting** = generic::*Free*<acc::ProtonTimesWeighting>

Re-scale the weighting of a cloned species by numberOfProtons.

When deriving species from each other, the new species "inherits" the macro-particle weighting of the first one. This functor can be used to manipulate the weighting of the new species' macro particles to be a multiplied by the number of protons of the initial species.

As an example, this is useful when initializing a quasi-neutral, pre-ionized plasma of ions and electrons. Electrons can be created from ions via deriving and increasing their weight to avoid simulating multiple macro electrons per macro ion (with Z>1).

note: needs the atomicNumbers flag on the initial species, used by the GetAtomicNumbers trait.

Manipulation Filters

Most of the particle functors shall operate on all valid particles, where `filter::All` is the default assumption. One can limit the domain or subset of particles with filters such as the ones below (or define new ones).

All

struct **All**

RelativeGlobalDomainPosition

template<typename **T_Params**>

struct **RelativeGlobalDomainPosition**

filter particle dependent on the global position

Check if a particle is within a relative area in one direction of the global domain.

Template Parameters

T_Params – picongpu::particles::filter::param::RelativeGlobalDomainPosition, parameter to configure the functor

Free

template<typename **T_Functor**>

struct **Free** : protected *picongpu*::*particles*::functor::User<*T_Functor*>
call simple free user defined filter

example for `particleFilters.param`: each particle with in-cell position greater than 0.5

```
struct FunctorEachParticleAboveMiddleOfTheCell
{
    template< typename T_Particle >
    HDINLINE bool operator()( T_Particle const & particle )
    {
        bool result = false;
        if( particle[ position_ ].y() >= float_X( 0.5 ) )
            result = true;
        return result;
    }
    static constexpr char const * name = "eachParticleAboveMiddleOfTheCell";

    static constexpr bool isDeterministic = true;
};

using EachParticleAboveMiddleOfTheCell = generic::Free<
   FunctorEachParticleAboveMiddleOfTheCell
>;
```

Template Parameters
T_Functor – user defined filter **optional**: can implement **one** host side constructor `T_Functor()` or `T_Functor(uint32_t currentTimeStep)`

FreeRng

template<typename **T_Functor**, typename **T_Distribution**>

struct **FreeRng** : protected *picongpu*::*particles*::functor::User<*T_Functor*>, private *picongpu*::*particles*::functor::misc::Rng<*T_Distribution*>
call simple free user defined functor and provide a random number generator

example for `particleFilters.param`: get every second particle (random sample of 50%)

```
struct FunctorEachSecondParticle
{
    template< typename T_Rng, typename T_Particle >
    HDINLINE bool operator()(
        T_Rng & rng,
        T_Particle const & particle
    )
    {
        bool result = false;
        if( rng() >= float_X( 0.5 ) )
            result = true;
        return result;
    }
```

(continues on next page)

(continued from previous page)

```
    static constexpr char const * name = "eachSecondParticle";

    static constexpr bool isDeterministic = false;
};

using EachSecondParticle = generic::FreeRng<
   FunctorEachSecondParticle,
   pmacc::random::distributions::Uniform< float_X >
>;
```

Template Parameters

- **T_Functor** – user defined unary functor
- **T_Distribution** – pmacc::random::distributions, random number distribution

FreeTotalCellOffset

template<typename **T_Functor**>

struct **FreeTotalCellOffset** : protected *picongpu*::*particles*::functor::User<*T_Functor*>, private *picongpu*::*particles*::functor::misc::TotalCellOffset

call simple free user defined functor and provide the cell information

The functor passes the cell offset of the particle relative to the total domain origin into the functor.

example for `particleFilters.param`: each particle with a cell offset of 5 in X direction

```
struct FunctorEachParticleInXCell5
{
    template< typename T_Particle >
    HDINLINE bool operator()(
        DataSpace< simDim > const & particleOffsetToTotalOrigin,
        T_Particle const & particle
    )
    {
        bool result = false;
        if( particleOffsetToTotalOrigin.x() == 5 )
            result = true;
        return result;
    }
    static constexpr char const * name = "eachParticleInXCell5";

    static constexpr bool isDeterministic = true;
};

using EachParticleInXCell5 = generic::FreeTotalCellOffset<
   FunctorEachParticleInXCell5
>;
```

Template Parameters

T_Functor – user defined unary functor

9.4.2 Memory

memory.param

Define low-level memory settings for compute devices.

Settings for memory layout for supercells and particle frame-lists, data exchanges in multi-device domain-decomposition and reserved fields for temporarily derived quantities are defined here.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Typedefs

using **SuperCellSize** = typename mCT::shrinkTo<mCT::Int<8, 8, 4>, *simDim*>::type

size of a superCell

volume of a superCell must be <= 1024

using **MappingDesc** = MappingDescription<*simDim*, *SuperCellSize*>

define mapper which is used for kernel call mappings

using **GuardSize** = typename mCT::shrinkTo<mCT::Int<1, 1, 1>, *simDim*>::type

define the size of the core, border and guard area

PIConGPU uses spatial domain-decomposition for parallelization over multiple devices with non-shared memory architecture. The global spatial domain is organized per device in three sections: the GUARD area contains copies of neighboring devices (also known as "halo"/"ghost"). The BORDER area is the outermost layer of cells of a device, equally to what neighboring devices see as GUARD area. The CORE area is the innermost area of a device. In union with the BORDER area it defines the "active" spatial domain on a device.

GuardSize is defined in units of SuperCellSize per dimension.

Variables

constexpr size_t **reservedGpuMemorySize** = 350 * 1024 * 1024

static constexpr uint32_t **numFrameSlots** = *pmacc*::math::CT::volume<*SuperCellSize*>::type::value

number of slots for particles within a frame

constexpr uint32_t **fieldTmpNumSlots** = 1

number of scalar fields that are reserved as temporary fields

constexpr bool **fieldTmpSupportGatherCommunication** = true

can *FieldTmp* gather neighbor information

If `true` it is possible to call the method `asyncCommunicationGather()` to copy data from the border of neighboring GPU into the local guard. This is also known as building up a "ghost" or "halo" region in domain decomposition and only necessary for specific algorithms that extend the basic PIC cycle, e.g. with dependence on derived density or energy fields.

struct **DefaultExchangeMemCfg**

bytes reserved for species exchange buffer

This is the default configuration for species exchanges buffer sizes when performing a simulation with 32bit precision (default for PIConGPU). For double precision the amount of memory used for exchanges will be automatically doubled. The default exchange buffer sizes can be changed per species by adding the alias exchangeMemCfg with similar members like in DefaultExchangeMemCfg to its flag list.

Public Types

using **REF_LOCAL_DOM_SIZE** = mCT::Int<0, 0, 0>

Reference local domain size.

The size of the local domain for which the exchange sizes `BYTES_*` are configured for. The required size of each exchange will be calculated at runtime based on the local domain size and the reference size. The exchange size will be scaled only up and not down. Zero means that there is no reference domain size, exchanges will not be scaled.

Public Members

const std::array<float_X, 3> **DIR_SCALING_FACTOR** = {{0.0, 0.0, 0.0}}

Scaling rate per direction.

1.0 means it scales linear with the ratio between the local domain size at runtime and the reference local domain size.

Public Static Attributes

static constexpr uint32_t **BYTES_EXCHANGE_X** = 1 * 1024 * 1024

static constexpr uint32_t **BYTES_EXCHANGE_Y** = 3 * 1024 * 1024

static constexpr uint32_t **BYTES_EXCHANGE_Z** = 1 * 1024 * 1024

static constexpr uint32_t **BYTES_EDGES** = 32 * 1024

static constexpr uint32_t **BYTES_CORNER** = 8 * 1024

precision.param

Define the precision of typically used floating point types in the simulation.

PIConGPU normalizes input automatically, allowing to use single-precision by default for the core algorithms. Note that implementations of various algorithms (usually plugins or non-core components) might still decide to hard-code a different (mixed) precision for some critical operations.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

mallocMC.param

Fine-tuning of the particle heap for GPUs: When running on GPUs, we use a high-performance parallel "new" allocator (mallocMC) which can be parametrized here.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Typedefs

using **DeviceHeap** = mallocMC::Allocator<*pmacc*::Acc<DIM1>, mallocMC::CreationPolicies::Scatter<*DeviceHeapConfig*>, mallocMC::DistributionPolicies::Noop, mallocMC::OOMPolicies::ReturnNull, mallocMC::ReservePoolPolicies::AlpakaBuf<*pmacc*::Acc<DIM1>>, mallocMC::AlignmentPolicies::Shrink<>>

Define a new allocator.

This is an allocator resembling the behaviour of the ScatterAlloc algorithm.

struct **DeviceHeapConfig**

configure the CreationPolicy "Scatter"

Public Static Attributes

static constexpr uint32_t **pagesize** = 2u * 1024u * 1024u

2MiB page can hold around 256 particle frames

static constexpr uint32_t **accessblocksize** = 2u * 1024u * 1024u * 1024u

accessblocksize, regionsize and wastefactor are not conclusively investigated and might be performance sensitive for multiple particle species with heavily varying attributes (frame sizes)

static constexpr uint32_t **regionsize** = 16u

static constexpr uint32_t **wastefactor** = 2u

static constexpr bool **resetfreedpages** = true

resetfreedpages is used to minimize memory fragmentation with varying frame sizes

9.4.3 PIC Extensions

fieldBackground.param

Load external background fields.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published

  –

  R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

  the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)

–

H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

class `FieldBackgroundE`

Public Functions

`PMACC_ALIGN`(m_unitField, const float3_64)

inline HINLINE `FieldBackgroundE`(const float3_64 unitField)

HDINLINE `FieldBackgroundE`(const *FieldBackgroundE*&) = default

`inline HDINLINE float3_X operator() (const DataSpace< simDim > &cellIdx, const uint32_t currentStep) const`

Specify your background field E(r,t) here.

Parameters

- `cellIdx` – The total cell id counted from the start at t = 0
- `currentStep` – The current time step

Public Static Attributes

static constexpr bool `InfluenceParticlePusher` = false

class `FieldBackgroundB`

Public Functions

`PMACC_ALIGN`(m_unitField, const float3_64)

inline HINLINE `FieldBackgroundB`(const float3_64 unitField)

HDINLINE `FieldBackgroundB`(const *FieldBackgroundB*&) = default

`inline HDINLINE float3_X operator() (const DataSpace< simDim > &cellIdx, const uint32_t currentStep) const`

Specify your background field B(r,t) here.

Parameters

- `cellIdx` – The total cell id counted from the start at t=0
- `currentStep` – The current time step

Public Static Attributes

static constexpr bool **InfluenceParticlePusher** = false

class **FieldBackgroundJ**

Public Functions

PMACC_ALIGN(m_unitField, const float3_64)

inline HINLINE **FieldBackgroundJ**(const float3_64 unitField)

HDINLINE **FieldBackgroundJ**(const *FieldBackgroundJ*&) = default

inline HDINLINE float3_X operator() (const DataSpace< simDim > &cellIdx, const uint32_t currentStep) const

Specify your background field J(r,t) here.

Parameters

- **cellIdx** – The total cell id counted from the start at t=0
- **currentStep** – The current time step

Public Static Attributes

static constexpr bool **activated** = false

ionizer.param

This file contains the proton and neutron numbers of commonly used elements of the periodic table.

The elements here should have a matching list of ionization energies in

Furthermore there are parameters for specific ionization models to be found here. That includes lists of screened nuclear charges as seen by bound electrons for the aforementioned elements as well as fitting parameters of the Thomas-Fermi ionization model.

See also:

ionizationEnergies.param. Moreover this file contains a description of how to configure an ionization model for a species.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  -
    R. Mewe. “Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences.” Astronomy and Astrophysics 20, 215 (1972)
  -
    H.-K. Chung, R.W. Lee, M.H. Chen. “A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter” High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **ionization**

Ionization Model Configuration.

- None : no particle is ionized
- BSI : simple barrier suppression ionization
- BSIEffectiveZ : BSI taking electron shielding into account via an effective atomic number Z_eff
- ADKLinPol : Ammosov-Delone-Krainov tunneling ionization (H-like) -> linearly polarized lasers
- ADKCircPol : Ammosov-Delone-Krainov tunneling ionization (H-like) -> circularly polarized lasers
- Keldysh : Keldysh ionization model
- ThomasFermi : statistical impact ionization based on Thomas-Fermi atomic model Attention: requires 2 *FieldTmp* slots

  Research and development:

  See also:

  memory.param
- BSIStarkShifted : BSI for hydrogen-like atoms and ions considering the Stark upshift of ionization potentials

Usage: Add flags to the list of particle flags that has the following structure

```
  ionizers< MakeSeq_t< particles::ionization::IonizationModel<␣
↪Species2BCreated > > >,
  atomicNumbers< ionization::atomicNumbers::Element_t >,
  effectiveNuclearCharge< ionization::effectiveNuclearCharge::Element_t >,
  ionizationEnergies< ionization::energies::AU::Element_t >
```

namespace **atomicNumbers**

Specify (chemical) element

Proton and neutron numbers define the chemical element that the ion species is based on. This value can be non-integer for physical models taking charge shielding effects into account.

It is wrapped into a struct because of C++ restricting floats from being template arguments.

See also:

http://en.wikipedia.org/wiki/Effective_nuclear_charge

Do not forget to set the correct mass and charge via `massRatio<>` and `chargeRatio<>`!

struct **Hydrogen_t**

H-1 99.98% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 1.0

static constexpr float_X **numberOfNeutrons** = 0.0

struct **Deuterium_t**

H-2 0.02% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 1.0

static constexpr float_X **numberOfNeutrons** = 1.0

struct **Helium_t**

He-4 ~100% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 2.0

static constexpr float_X **numberOfNeutrons** = 2.0

struct **Carbon_t**

C-12 98.9% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 6.0

static constexpr float_X **numberOfNeutrons** = 6.0

struct **Nitrogen_t**

N-14 99.6% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 7.0

static constexpr float_X **numberOfNeutrons** = 7.0

struct **Oxygen_t**

O-16 99.76% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 8.0

static constexpr float_X **numberOfNeutrons** = 8.0

struct **Aluminium_t**

Al-27 ~100% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 13.0

static constexpr float_X **numberOfNeutrons** = 14.0

struct **Silicon_t**

Si-28 ~92.23% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 14.0

static constexpr float_X **numberOfNeutrons** = 14.0

struct **Copper_t**

Cu-63 69.15% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 29.0

static constexpr float_X **numberOfNeutrons** = 34.0

struct **Gold_t**

Au-197 ~100% NA.

Public Static Attributes

static constexpr float_X **numberOfProtons** = 79.0

static constexpr float_X **numberOfNeutrons** = 118.0

namespace **effectiveNuclearCharge**

Effective Nuclear Charge.

Due to the shielding effect of inner electron shells in an atom / ion which makes the core charge seem smaller to valence electrons new, effective, atomic core charge numbers can be defined to make the crude barrier suppression ionization (BSI) model less inaccurate.

References: Clementi, E.; Raimondi, D. L. (1963) "Atomic Screening Constants from SCF Functions" J. Chem. Phys. 38 (11): 2686-2689. doi:10.1063/1.1733573 Clementi, E.; Raimondi, D. L.; Reinhardt, W. P. (1967) "Atomic Screening Constants from SCF Functions. II. Atoms with 37 to 86 Electrons" Journal of Chemical Physics. 47: 1300-1307. doi:10.1063/1.1712084

See also:

https://en.wikipedia.org/wiki/Effective_nuclear_charge or refer directly to the calculations by Slater or Clementi and Raimondi

IMPORTANT NOTE: You have to insert the values in REVERSE order since the lowest shell corresponds to the last ionization process!

Functions

`PMACC_CONST_VECTOR (float_X, 1, Hydrogen, 1.)`

`PMACC_CONST_VECTOR (float_X, 1, Deuterium, 1.)`

`PMACC_CONST_VECTOR (float_X, 2, Helium, 1.688, 1.688)`

`PMACC_CONST_VECTOR (float_X, 6, Carbon, 3.136, 3.136, 3.217, 3.217, 5.673, 5.673)`

`PMACC_CONST_VECTOR (float_X, 7, Nitrogen, 3.834, 3.834, 3.834, 3.874, 3.874, 6.665, 6.665)`

`PMACC_CONST_VECTOR (float_X, 8, Oxygen, 4.453, 4.453, 4.453, 4.453, 4.492, 4.492, 7.658, 7.658)`

`PMACC_CONST_VECTOR (float_X, 13, Aluminium, 4.066, 4.117, 4.117, 8.963, 8.963, 8.963, 8.963, 8.963, 8.963, 8.214, 8.214, 12.591, 12.591)`

`PMACC_CONST_VECTOR (float_X, 14, Silicon, 4.285, 4.285, 4.903, 4.903, 9.945, 9.945, 9.945, 9.945, 9.945, 9.945, 9.020, 9.020, 13.575, 13.575)`

`PMACC_CONST_VECTOR (float_X, 29, Copper, 13.201, 13.201, 13.201, 13.201, 13.201, 13.201, 13.201, 13.201, 13.201, 13.201, 5.842, 14.731, 14.731, 14.731, 14.731, 14.731, 14.731, 15.594, 15.594, 25.097, 25.097, 25.097, 25.097, 25.097, 25.097, 21.020, 21.020, 28.339, 28.339)`

`PMACC_CONST_VECTOR (float_X, 79, Gold, 20.126, 20.126, 20.126, 20.126, 20.126, 20.126, 20.126, 20.126, 20.126, 20.126, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 40.650, 10.938, 25.170, 25.170, 25.170, 25.170, 25.170, 25.170, 41.528, 41.528, 41.528, 41.528, 41.528, 41.528, 41.528, 41.528, 41.528, 41.528, 27.327, 27.327, 43.547, 43.547, 43.547, 43.547, 43.547, 43.547, 65.508, 65.508, 65.508, 65.508, 65.508, 65.508, 65.508, 65.508, 65.508, 65.508, 44.413, 44.413, 56.703, 56.703, 56.703, 56.703, 56.703, 56.703, 55.763, 55.763, 74.513, 74.513, 74.513, 74.513, 74.513, 74.513, 58.370, 58.370, 77.476, 77.476)`

namespace **particles**

namespace **ionization**

namespace **thomasFermi**

Variables

constexpr float_X **TFAlpha** = 14.3139

Fitting parameters to average ionization degree Z* = 4/3*pi*R_0^3 * n(R_0) as an extension towards arbitrary atoms and temperatures.

See table IV of \url http://www.sciencedirect.com/science/article/pii/S0065219908601451 doi:10.1016/S0065-2199(08)60145-1

constexpr float_X **TFBeta** = 0.6624

constexpr float_X **TFA1** = 3.323e-3

constexpr float_X **TFA2** = 9.718e-1

constexpr float_X **TFA3** = 9.26148e-5

constexpr float_X **TFA4** = 3.10165

constexpr float_X **TFB0** = -1.7630

constexpr float_X **TFB1** = 1.43175

constexpr float_X **TFB2** = 0.31546

constexpr float_X **TFC1** = -0.366667

constexpr float_X **TFC2** = 0.983333

constexpr float_X **CUTOFF_MAX_ENERGY_KEV** = 50.0

cutoff energy for electron "temperature" calculation

In laser produced plasmas we can have different, well-separable groups of electrons. For the Thomas-Fermi ionization model we only want the thermalized "bulk" electrons. Including the high-energy "prompt" electrons is physically questionable since they do not have a large cross section for collisional ionization.

unit: keV

constexpr float_X **CUTOFF_MAX_ENERGY** = *CUTOFF_MAX_ENERGY_KEV * UNITCONV_keV_to_Joule*

cutoff energy for electron "temperature" calculation in SI units

constexpr float_X **CUTOFF_LOW_DENSITY** = 1.7422e27

lower ion density cutoff

The Thomas-Fermi model yields unphysical artifacts for low ion densities. Low ion densities imply lower collision frequency and thus less collisional ionization. The Thomas-Fermi model yields an increasing charge state for decreasing densities and electron temperatures of 10eV and above. This cutoff will be used to set the lower application threshold for charge state calculation.

unit: 1 / m^3

example: 1.7422e27 as a hydrogen ion number density equal to the corresponding critical electron number density for an 800nm laser

The choice of the default is motivated by by the following: In laser-driven plasmas all dynamics in density regions below the critical electron density will be laser-dominated. Once ions of that density are ionized once the laser will not penetrate fully anymore and the as electrons are heated the dynamics will be collision-dominated.

Note: This cutoff value should be set in accordance to FLYCHK calculations, for instance! It is not a universal value and requires some preliminary approximations!

constexpr float_X **CUTOFF_LOW_TEMPERATURE_EV** = 1.0

lower electron temperature cutoff

The Thomas-Fermi model predicts initial ionization for many materials of solid density even when the electron temperature is 0.

ionizationEnergies.param

This file contains the ionization energies of commonly used elements of the periodic table.

Each atomic species in PIConGPU can represent exactly one element. The ionization energies of that element are stored in a vector which contains the *name* and *proton number* as well as a list of *energy values*. The number of ionization levels must be equal to the proton number of the element.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
    - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

      the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
    - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

      highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **ionization**

Ionization Model Configuration.

- None : no particle is ionized
- BSI : simple barrier suppression ionization
- BSIEffectiveZ : BSI taking electron shielding into account via an effective atomic number Z_eff
- ADKLinPol : Ammosov-Delone-Krainov tunneling ionization (H-like) -> linearly polarized lasers
- ADKCircPol : Ammosov-Delone-Krainov tunneling ionization (H-like) -> circularly polarized lasers
- Keldysh : Keldysh ionization model
- ThomasFermi : statistical impact ionization based on Thomas-Fermi atomic model Attention: requires 2 *FieldTmp* slots

  Research and development:

  See also:

  memory.param
- BSIStarkShifted : BSI for hydrogen-like atoms and ions considering the Stark upshift of ionization potentials

Usage: Add flags to the list of particle flags that has the following structure

```
  ionizers< MakeSeq_t< particles::ionization::IonizationModel<␣
↪Species2BCreated > > >,
  atomicNumbers< ionization::atomicNumbers::Element_t >,
  effectiveNuclearCharge< ionization::effectiveNuclearCharge::Element_t >,
  ionizationEnergies< ionization::energies::AU::Element_t >
```

namespace **energies**

Ionization potentials.

Please follow these rules for defining ionization energies of atomic species, unless your chosen ionization model requires a different unit system than `AU::`

- input of values in either atomic units or converting eV or Joule to them -> use either UNITCONV_eV_to_AU or SI::ATOMIC_UNIT_ENERGY for that purpose
- use `float_X` as the preferred data type

example: ionization energy for ground state hydrogen: 13.6 eV 1 Joule = 1 kg * m^2 / s^2 1 eV = 1.602e-19 J

1 AU (energy) = 27.2 eV = 1 Hartree = 4.36e-18 J = 2 Rydberg = 2 x Hydrogen ground state binding energy

Atomic units are useful for ionization models because they simplify the formulae greatly and provide intuitively understandable relations to a well-known system, i.e. the Hydrogen atom.

for PMACC_CONST_VECTOR usage,

Reference: Kramida, A., Ralchenko, Yu., Reader, J., and NIST ASD Team (2014) NIST Atomic Spectra Database (ver. 5.2), [Online] Available: http://physics.nist.gov/asd [2017, February 8] National Institute of Standards and Technology, Gaithersburg, MD

See also:

include/pmacc/math/ConstVector.hpp for finding ionization energies, @url http://physics.nist.gov/PhysRefData/ASD/ionEnergy.html

namespace **AU**

Functions

PMACC_CONST_VECTOR (float_X, 1, Hydrogen, 13.59843 *UNITCONV_eV_to_AU)

PMACC_CONST_VECTOR (float_X, 1, Deuterium, 13.60213 *UNITCONV_eV_to_AU)

PMACC_CONST_VECTOR (float_X, 2, Helium, 24.58739 *UNITCONV_eV_to_AU, 54.41776 *UNITCONV_eV_to_AU)

PMACC_CONST_VECTOR (float_X, 6, Carbon, 11.2603 *UNITCONV_eV_to_AU, 24.3845 *UNITCONV_eV_to_AU, 47.88778 *UNITCONV_eV_to_AU, 64.49351 *UNITCONV_eV_to_AU, 392.0905 *UNITCONV_eV_to_AU, 489.993177 *UNITCONV_eV_to_AU)

PMACC_CONST_VECTOR (float_X, 7, Nitrogen, 14.53413 *UNITCONV_eV_to_AU, 29.60125 *UNITCONV_eV_to_AU, 47.4453 *UNITCONV_eV_to_AU, 77.4735 *UNITCONV_eV_to_AU, 97.89013 *UNITCONV_eV_to_AU, 552.06731 *UNITCONV_eV_to_AU, 667.04609 *UNITCONV_eV_to_AU)

```
PMACC_CONST_VECTOR (float_X, 8, Oxygen, 13.
61805 *UNITCONV_eV_to_AU, 35.12112 *UNITCONV_eV_to_AU, 54.
93554 *UNITCONV_eV_to_AU, 77.41350 *UNITCONV_eV_to_AU, 113.
8989 *UNITCONV_eV_to_AU, 138.1189 *UNITCONV_eV_to_AU, 739.
3268 *UNITCONV_eV_to_AU, 871.4098 *UNITCONV_eV_to_AU)

PMACC_CONST_VECTOR (float_X, 13, Aluminium, 5.
98577 *UNITCONV_eV_to_AU, 18.8285 *UNITCONV_eV_to_AU, 28.
4476 *UNITCONV_eV_to_AU, 119.992 *UNITCONV_eV_to_AU, 153.
825 *UNITCONV_eV_to_AU, 190.495 *UNITCONV_eV_to_AU, 241.
769 *UNITCONV_eV_to_AU, 284.647 *UNITCONV_eV_to_AU, 330.
214 *UNITCONV_eV_to_AU, 398.656 *UNITCONV_eV_to_AU, 442.
006 *UNITCONV_eV_to_AU, 2085.97 *UNITCONV_eV_to_AU, 2304.
14 *UNITCONV_eV_to_AU)

PMACC_CONST_VECTOR (float_X, 14, Silicon, 8.
151683 *UNITCONV_eV_to_AU, 16.345845 *UNITCONV_eV_to_AU, 33.
493 *UNITCONV_eV_to_AU, 45.14179 *UNITCONV_eV_to_AU, 166.
767 *UNITCONV_eV_to_AU, 205.267 *UNITCONV_eV_to_AU, 246.
32 *UNITCONV_eV_to_AU, 303.66 *UNITCONV_eV_to_AU, 351.
1 *UNITCONV_eV_to_AU, 401.38 *UNITCONV_eV_to_AU, 476.
18 *UNITCONV_eV_to_AU, 523.415 *UNITCONV_eV_to_AU, 2437.
65804 *UNITCONV_eV_to_AU, 2673.1774 *UNITCONV_eV_to_AU)

PMACC_CONST_VECTOR (float_X, 29, Copper, 7.
72638 *UNITCONV_eV_to_AU, 20.2924 *UNITCONV_eV_to_AU, 36.
8411 *UNITCONV_eV_to_AU, 57.385 *UNITCONV_eV_to_AU, 79.
87 *UNITCONV_eV_to_AU, 103.010 *UNITCONV_eV_to_AU, 139.
012 *UNITCONV_eV_to_AU, 166.021 *UNITCONV_eV_to_AU, 198.
022 *UNITCONV_eV_to_AU, 232.25 *UNITCONV_eV_to_AU, 265.
332 *UNITCONV_eV_to_AU, 367.09 *UNITCONV_eV_to_AU, 401.
03 *UNITCONV_eV_to_AU, 436.06 *UNITCONV_eV_to_AU, 483.
19 *UNITCONV_eV_to_AU, 518.712 *UNITCONV_eV_to_AU, 552.
821 *UNITCONV_eV_to_AU, 632.56 *UNITCONV_eV_to_AU, 670.
608 *UNITCONV_eV_to_AU, 1690.59 *UNITCONV_eV_to_AU, 1800.
3 *UNITCONV_eV_to_AU, 1918.4 *UNITCONV_eV_to_AU, 2044.
6 *UNITCONV_eV_to_AU, 2179.4 *UNITCONV_eV_to_AU, 2307.
32 *UNITCONV_eV_to_AU, 2479.12 *UNITCONV_eV_to_AU, 2586.
95 *UNITCONV_eV_to_AU, 11062.4 *UNITCONV_eV_to_AU, 11567.
6 *UNITCONV_eV_to_AU)
```

```
PMACC_CONST_VECTOR (float_X, 79, Gold, 9.2256 *UNITCONV_eV_to_AU,
20.203 *UNITCONV_eV_to_AU, 30.016 *UNITCONV_eV_to_AU, 45.
017 *UNITCONV_eV_to_AU, 60.019 *UNITCONV_eV_to_AU, 74.
020 *UNITCONV_eV_to_AU, 94.020 *UNITCONV_eV_to_AU, 112.
02 *UNITCONV_eV_to_AU, 130.12 *UNITCONV_eV_to_AU, 149.
02 *UNITCONV_eV_to_AU, 168.21 *UNITCONV_eV_to_AU, 248.
01 *UNITCONV_eV_to_AU, 275.14 *UNITCONV_eV_to_AU, 299.
15 *UNITCONV_eV_to_AU, 324.16 *UNITCONV_eV_to_AU, 365.
19 *UNITCONV_eV_to_AU, 392.20 *UNITCONV_eV_to_AU, 433.
21 *UNITCONV_eV_to_AU, 487.25 *UNITCONV_eV_to_AU, 517.
30 *UNITCONV_eV_to_AU, 546.30 *UNITCONV_eV_to_AU, 600.
30 *UNITCONV_eV_to_AU, 650.40 *UNITCONV_eV_to_AU, 710.
40 *UNITCONV_eV_to_AU, 760.40 *UNITCONV_eV_to_AU, 820.
40 *UNITCONV_eV_to_AU, 870.40 *UNITCONV_eV_to_AU, 930.
50 *UNITCONV_eV_to_AU, 990.50 *UNITCONV_eV_to_AU, 1040.
5 *UNITCONV_eV_to_AU, 1100.5 *UNITCONV_eV_to_AU, 1150.
6 *UNITCONV_eV_to_AU, 1210.6 *UNITCONV_eV_to_AU, 1475.
5 *UNITCONV_eV_to_AU, 1527.5 *UNITCONV_eV_to_AU, 1584.
5 *UNITCONV_eV_to_AU, 1644.5 *UNITCONV_eV_to_AU, 1702.
4 *UNITCONV_eV_to_AU, 1758.4 *UNITCONV_eV_to_AU, 1845.
4 *UNITCONV_eV_to_AU, 1904.4 *UNITCONV_eV_to_AU, 1967.
4 *UNITCONV_eV_to_AU, 2026.4 *UNITCONV_eV_to_AU, 2261.
4 *UNITCONV_eV_to_AU, 2320.4 *UNITCONV_eV_to_AU, 2383.
4 *UNITCONV_eV_to_AU, 2443.4 *UNITCONV_eV_to_AU, 2640.
4 *UNITCONV_eV_to_AU, 2708.4 *UNITCONV_eV_to_AU, 2870.
4 *UNITCONV_eV_to_AU, 2941.0 *UNITCONV_eV_to_AU, 4888.
4 *UNITCONV_eV_to_AU, 5013.4 *UNITCONV_eV_to_AU, 5156.
5 *UNITCONV_eV_to_AU, 5307.5 *UNITCONV_eV_to_AU, 5452.
5 *UNITCONV_eV_to_AU, 5594.5 *UNITCONV_eV_to_AU, 5846.
6 *UNITCONV_eV_to_AU, 5994.6 *UNITCONV_eV_to_AU, 6156.
7 *UNITCONV_eV_to_AU, 6305.1 *UNITCONV_eV_to_AU, 6724.
1 *UNITCONV_eV_to_AU, 6854.1 *UNITCONV_eV_to_AU, 6997.
2 *UNITCONV_eV_to_AU, 7130.2 *UNITCONV_eV_to_AU, 7756.
3 *UNITCONV_eV_to_AU, 7910.4 *UNITCONV_eV_to_AU, 8210.
4 *UNITCONV_eV_to_AU, 8360.5 *UNITCONV_eV_to_AU, 18040.
*UNITCONV_eV_to_AU, 18401. *UNITCONV_eV_to_AU, 18791.
*UNITCONV_eV_to_AU, 19151. *UNITCONV_eV_to_AU, 21471.
*UNITCONV_eV_to_AU, 21921. *UNITCONV_eV_to_AU, 22500.
*UNITCONV_eV_to_AU, 22868. *UNITCONV_eV_to_AU, 91516.
*UNITCONV_eV_to_AU, 93254. *UNITCONV_eV_to_AU)
```

collision.param

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  -

H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **particles**

namespace **collision**

Typedefs

using **CollisionScreeningSpecies** = MakeSeq_t<>

List of (filtered) species contributing to the screening (Debye) length calculation.

Elements must be either a species types or a misc::SpeciesFilter specialization.

using **CollisionPipeline** = *pmacc*::mp_list<>

CollisionPipeline defines in which order species interact with each other.

the functors are called in order (from first to last functor)

Variables

constexpr bool **debugScreeningLength** = false

constexpr uint32_t **cellListChunkSize** = std::min(*particles*::*TYPICAL_PARTICLES_PER_CELL*, 4u)

Chunk size used for cell list allocations.

To reduce the fragmentation of the heap memory on accelerators the collision algorithm is allocating a multiple of this value to store a cell list of particle IDs. The value must be non zero.

Our experience shows that 4 or 8 are working quite well. Higher numbers lead to more inefficient memory usage, so we cap this chunk size by default at the TYPICAL_PARTICLES_PER_CELL value.

namespace **precision**

Typedefs

using **float_COLL** = float_64

More information on collision.param

9.4.4 Plugins

fileOutput.param

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Typedefs

using **ChargeDensity_Seq** = deriveField::CreateEligible_t<*VectorAllSpecies*, deriveField::derivedAttributes::ChargeDensity>

FieldTmp output (calculated at runtime) *******************************.

Those operations derive scalar field quantities from particle species at runtime. Each value is mapped per cell. Some operations are identical up to a constant, so avoid writing those twice to save storage.

you can choose any of these particle to grid projections:

- Density: particle position + shape on the grid
- BoundElectronDensity: density of bound electrons note: only makes sense for partially ionized ions
- ChargeDensity: density * charge note: for species that do not change their charge state, this is the same as the density times a constant for the charge
- Energy: sum of kinetic particle energy per cell with respect to shape

- EnergyDensity: average kinetic particle energy per cell times the particle density note: this is the same as the sum of kinetic particle energy divided by a constant for the cell volume
- Momentum: sum of chosen component of momentum per cell with respect to shape
- MomentumDensity: average chosen component of momentum per cell times the particle density note: this is the same as the sum of the chosen momentum component divided by a constant for the cell volume
- LarmorPower: radiated Larmor power (species must contain the attribute `momentumPrev1`)

for debugging:

- MidCurrentDensityComponent: density * charge * velocity_component
- Counter: counts point like particles per cell
- MacroCounter: counts point like macro particles per cell

combined attributes: These attributes are defined as a function of two primary derived attributes.

- AverageAttribute: template to compute a per particle average of any derived value
- RelativisticDensity: equals to 1/gamma^2 * n. Where gamma is the Lorentz factor for the mean kinetic energy in the cell and n ist the usual number density. Useful for Faraday Rotation calculation.

  Example use:

```
using AverageEnergy_Seq = deriveField::CreateEligible_t<
    VectorAllSpecies,
    deriveField::combinedAttributes::AverageAttribute
↪<deriveField::derivedAttributes::Energy>>;
using RelativisticDensity_Seq
    = deriveField::CreateEligible_t<PIC_Electrons,␣
↪deriveField::combinedAttributes::RelativisticDensity>;
```

Filtering: You can create derived fields from filtered particles. Only particles passing the filter will contribute to the field quantity. For that you need to define your filters in `particleFilters.param` and pass a filter as the 3rd (optional) template argument in `CreateEligible_t`.

Example: This will create charge density field for all species that are eligible for the this attribute and the chosen filter.

```
using ChargeDensity_Seq
= deriveField::CreateEligible_t< VectorAllSpecies,
deriveField::derivedAttributes::ChargeDensity, filter::FilterOfYourChoice>;
```

using **`EnergyDensity_Seq`** = deriveField::CreateEligible_t<*VectorAllSpecies*, deriveField::derivedAttributes::EnergyDensity>

using **`FieldTmpSolvers`** = MakeSeq_t<*ChargeDensity_Seq*, *EnergyDensity_Seq*>

FieldTmpSolvers groups all solvers that create data for *FieldTmp* ******.

FieldTmpSolvers is used in

See also:

FieldTmp to calculate the exchange size

using **`NativeFileOutputFields`** = MakeSeq_t<*FieldE*, *FieldB*>

FileOutputFields: Groups all Fields that shall be dumped.

Possible native fields: *FieldE*, *FieldB*, *FieldJ* using **FileOutputFields** = MakeSeq_t<*NativeFileOutputFields*, *FieldTmpSolvers*>

using **FileOutputParticles** = *VectorAllSpecies*

FileOutputParticles: Groups all Species that shall be dumped **********.

hint: to disable particle output set to using FileOutputParticles = MakeSeq_t< >;

isaac.param

Definition which native fields and density fields of (filtered) particles will be visualizable with ISAAC.

ISAAC is an in-situ visualization library with which the PIC simulation can be observed while it is running avoiding the time consuming writing and reading of simulation data for the classical post processing of data.

ISAAC can directly visualize natives fields like the E or B field, but density fields of particles need to be calculated from PIConGPU on the fly which slightly increases the runtime and the memory consumption. Every particle density field will reduce the amount of memory left for PIConGPUs particles and fields.

To get best performance, ISAAC defines an exponential amount of different visualization kernels for every combination of (at runtime) activated fields. So furthermore a lot of fields will increase the compilation time.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **isaacP**

Typedefs

using **Particle_Seq** = *VectorAllSpecies*

Intermediate list of native particle species of PIConGPU which shall be visualized.

using **Native_Seq** = MakeSeq_t<*FieldE*, *FieldB*, *FieldJ*>

Intermediate list of native fields of PIConGPU which shall be visualized.

using **Density_Seq** = deriveField::CreateEligible_t<*Particle_Seq*, deriveField::derivedAttributes::Density>

Intermediate list of particle species, from which density fields shall be created at runtime to visualize them.

You can create such densities from filtered particles by passing a particle filter as the third template argument (`filter::All` by default). Don't forget to add your filtered densities to the `Fields_Seq` below.

```
using Density_Seq_Filtered = deriveField::CreateEligible_t<Particle_
↪Seq,
    deriveField::derivedAttributes::Density, filter::All>;
```

using **Fields_Seq** = MakeSeq_t<*Native_Seq*, *Density_Seq*>

Compile time sequence of all fields which shall be visualized.

Basically joining Native_Seq and Density_Seq.

using **VectorFields_Seq** = *Native_Seq*

Compile time sequence of all fields which shall be visualized.

Basically joining Native_Seq and Density_Seq.

particleCalorimeter.param

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published

  –

  R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

  the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)

  –

  H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

  highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace `particleCalorimeter`

Functions

`HDINLINE float2_X mapYawPitchToNormedRange (const float_X yaw, const float_X pitch, const float_X maxYaw, const float_X maxPitch)`

Map yaw and pitch into [0,1] respectively.

These ranges correspond to the normalized histogram range of the calorimeter (0: first bin, 1: last bin). Out-of-range values are mapped to the first or the last bin.

Useful for fine tuning the spatial calorimeter resolution.

Parameters

- `yaw` – -maxYaw. . . maxYaw
- `pitch` – -maxPitch. . . maxPitch
- `maxYaw` – maximum value of angle yaw
- `maxPitch` – maximum value of angle pitch

Returns

Two values within [-1,1]

radiation.param

Definition of frequency space, number of observers, filters, form factors and window functions of the radiation plugin.

All values set here determine what the radiation plugin will compute. The observation direction is defined in a seperate file `radiationObserver.param`. On the comand line the plugin still needs to be called for each species the radiation should be computed for.

Defines

`PIC_VERBOSE_RADIATION`

radiation verbose level: 0=nothing, 1=physics, 2=simulation_state, 4=memory, 8=critical

namespace `picongpu`

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. “Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences.” Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. “A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter” High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **plugins**

namespace **radiation**

Typedefs

using **RadiationParticleFilter** = *picongpu*::*particles*::*manipulators*::generic::*Free*<*GammaFilterFunctor*>

filter to (de)select particles for the radiation calculation

to activate the filter:

- goto file `speciesDefinition.param`
- add the attribute `radiationMask` to the particle species

struct **GammaFilterFunctor**

select particles for radiation example of a filter for the relativistic Lorentz factor gamma

Public Functions

`template<typename T_Particle> inline HDINLINE void operator() (T_Particle &particl`

Public Static Attributes

static constexpr float_X **radiationGamma** = 5.0

Gamma value above which the radiation is calculated.

namespace **frequencies_from_list**

Variables

constexpr const char ***listLocation** = "/path/to/frequency_list"

path to text file with frequencies

constexpr unsigned int **N_omega** = 2048

number of frequency values to compute if frequencies are given in a file [unitless]

namespace **linear_frequencies**

Variables

constexpr unsigned int **`N_omega`** = 2048
number of frequency values to compute in the linear frequency [unitless]

namespace **`SI`**

Variables

constexpr float_64 **`omega_min`** = 0.0
mimimum frequency of the linear frequency scale in units of [1/s]

constexpr float_64 **`omega_max`** = 1.06e16
maximum frequency of the linear frequency scale in units of [1/s]

namespace **`log_frequencies`**

Variables

constexpr unsigned int **`N_omega`** = 2048
number of frequency values to compute in the logarithmic frequency [unitless]

namespace **`SI`**

Variables

constexpr float_64 **`omega_min`** = 1.0e14
mimimum frequency of the logarithmic frequency scale in units of [1/s]

constexpr float_64 **`omega_max`** = 1.0e17
maximum frequency of the logarithmic frequency scale in units of [1/s]

namespace **`parameters`**

Variables

constexpr unsigned int **`N_observer`** = 256
number of observation directions

namespace **`radFormFactor_CIC_1Dy`**

namespace **`radFormFactor_CIC_3D`**
correct treatment of coherent and incoherent radiation from macro particles

Choose different form factors in order to consider different particle shapes for radiation

- radFormFactor_CIC_3D … CIC charge distribution
- radFormFactor_TSC_3D … TSC charge distribution
- radFormFactor_PCS_3D … PCS charge distribution
- radFormFactor_CIC_1Dy … only CIC charge distribution in y
- radFormFactor_Gauss_spherical … symmetric Gauss charge distribution
- radFormFactor_Gauss_cell … Gauss charge distribution according to cell size
- radFormFactor_incoherent … only incoherent radiation
- radFormFactor_coherent … only coherent radiation

namespace **radFormFactor_coherent**

namespace **radFormFactor_Gauss_cell**

namespace **radFormFactor_Gauss_spherical**

namespace **radFormFactor_incoherent**

namespace **radFormFactor_PCS_3D**

namespace **radFormFactor_TSC_3D**

namespace **radiationNyquist**

selected mode of frequency scaling:

options:

- linear_frequencies
- log_frequencies
- frequencies_from_list

Variables

constexpr float_32 **NyquistFactor** = 0.5

Nyquist factor: fraction of the local Nyquist frequency above which the spectra is set to zero should be in (0, 1).

namespace **radWindowFunctionGauss**

namespace **radWindowFunctionHamming**

namespace **radWindowFunctionNone**

namespace **radWindowFunctionTriangle**

add a window function weighting to the radiation in order to avoid ringing effects from sharpe boundaries default: no window function via `radWindowFunctionNone`

Choose different window function in order to get better ringing reduction radWindowFunctionTriangle radWindowFunctionHamming radWindowFunctionTriplett radWindowFunctionGauss radWindowFunctionNone

namespace **radWindowFunctionTriplett**

radiationObserver.param

This file defines a function describing the observation directions.

It takes an integer index from [0, picongpu::parameters::N_observer) and maps it to a 3D unit vector in R^3 (norm=1) space that describes the observation direction in the PIConGPU cartesian coordinate system.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published

  –

  R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

  the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)

–

H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

---

namespace **`plugins`**

namespace **`radiation`**

namespace **`radiation_observer`**

Functions

`HDINLINE vector_64 observationDirection (const int observation_id_extern)`

Compute observation angles.

This function is used in the Radiation plug-in kernel to compute the observation directions given as a unit vector pointing towards a 'virtual' detector

This default setup is an example of a 2D detector array. It computes observation directions for 2D virtual detector field with its center pointing toward the +y direction (for theta=0, phi=0) with observation angles ranging from theta = [angle_theta_start : angle_theta_end] phi = [angle_phi_start : angle_phi_end] Every observation_id_extern index moves the phi angle from its start value toward its end value until the observation_id_extern reaches N_split. After that the theta angle moves further from its start value towards its end value while phi is reset to its start value.

The unit vector pointing towards the observing virtual detector can be described using theta and phi by: x_value = sin(theta) * cos(phi) y_value = cos(theta) z_value = sin(theta) * sin(phi) These are the standard spherical coordinates.

The example setup describes an detector array of 16x16 detectors ranging from -pi/8= -22.5 degrees to +pi/8= +22.5 degrees for both angles with the center pointing toward the y-axis (laser propagation direction).

Parameters

`observation_id_extern` – int index that identifies each block on the GPU to compute the observation direction

Returns

unit vector pointing in observation direction type: vector_64

png.param

Defines

`EM_FIELD_SCALE_CHANNEL1`

`EM_FIELD_SCALE_CHANNEL2`

`EM_FIELD_SCALE_CHANNEL3`

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Variables

constexpr float_64 **`scale_image`** = 1.0

constexpr bool **`scale_to_cellsize`** = true

constexpr bool **`white_box_per_GPU`** = false

namespace **`visPreview`**

Unnamed Group

DINLINE float_X preChannel1 (const float3_X &field_B, const float3_X &field_E, const float3_X &field_Current)

Calculate values for png channels for given field values.

Parameters

- `field_B` – normalized magnetic field value
- `field_E` – normalized electric field value
- `field_Current` – normalized electric current value (note - not current density)

DINLINE float_X preChannel2 (const float3_X &field_B, const float3_X &field_E, const float3_X &field_Current)

DINLINE float_X preChannel3 (const float3_X &field_B, const float3_X &field_E, const float3_X &field_Current)

Variables

constexpr float_64 **customNormalizationSI**[3] = {5.0e12 / *SI*::*SPEED_OF_LIGHT_SI*, 5.0e12, 15.0}

SI values to be used for Custom normalization.

The order of normalization values is: B, E, current (note - current, not current density). This variable must always be defined, but has no effect for other normalization types.

constexpr float_X **preParticleDens_opacity** = 0.25_X

constexpr float_X **preChannel1_opacity** = 1.0_X

constexpr float_X **preChannel2_opacity** = 1.0_X

constexpr float_X **preChannel3_opacity** = 1.0_X

pngColorScales.param

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace `colorScales`

namespace `blue`

Functions

```
HDINLINE void addRGB (float3_X &img, const float_X value,
const float_X opacity)
```

namespace `gray`

Functions

```
HDINLINE void addRGB (float3_X &img, const float_X value,
const float_X opacity)
```

namespace `grayInv`

Functions

```
HDINLINE void addRGB (float3_X &img, const float_X value,
const float_X opacity)
```

namespace `green`

Functions

```
HDINLINE void addRGB (float3_X &img, const float_X value,
const float_X opacity)
```

namespace `none`

Functions

```
HDINLINE void addRGB (const float3_X &, const float_X, const float_X)
```

namespace `red`

Functions

`HDINLINE void addRGB (float3_X &img, const float_X value, const float_X opacity)`

transitionRadiation.param

Definition of frequency space, number of observers, filters and form factors of the transition radiation plugin.

All values set here determine what the radiation plugin will compute. On the comand line the plugin still needs to be called for each species the transition radiation should be computed for.

Defines

`PIC_VERBOSE_RADIATION`

Uses the same verbose level schemes as the radiation plugin.

radiation verbose level: 0=nothing, 1=physics, 2=simulation_state, 4=memory, 8=critical

namespace **`picongpu`**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **`plugins`**

namespace **`radiation`**

namespace **`radFormFactor_CIC_1Dy`**

namespace **`radFormFactor_CIC_3D`**

correct treatment of coherent and incoherent radiation from macro particles

Choose different form factors in order to consider different particle shapes for radiation

- radFormFactor_CIC_3D … CIC charge distribution

- radFormFactor_TSC_3D … TSC charge distribution
- radFormFactor_PCS_3D … PCS charge distribution
- radFormFactor_CIC_1Dy … only CIC charge distribution in y
- radFormFactor_Gauss_spherical … symmetric Gauss charge distribution
- radFormFactor_Gauss_cell … Gauss charge distribution according to cell size
- radFormFactor_incoherent … only incoherent radiation
- radFormFactor_coherent … only coherent radiation

namespace **`radFormFactor_coherent`**

namespace **`radFormFactor_Gauss_cell`**

namespace **`radFormFactor_Gauss_spherical`**

namespace **`radFormFactor_incoherent`**

namespace **`radFormFactor_PCS_3D`**

namespace **`radFormFactor_TSC_3D`**

namespace **`transitionRadiation`**

Typedefs

using **`GammaFilter`** = *picongpu*::*particles*::*manipulators*::generic::*Free*<*GammaFilterFunctor*>

filter to (de)select particles for the radiation calculation

to activate the filter:

- goto file `speciesDefinition.param`
- add the attribute `transitionRadiationMask` to the particle species

Warning: Do not remove this filter. Otherwise still standing electrons would generate NaNs in the output of the plugin and transition radiation is scaling proportionally to gamma^2.

Functions

`HDINLINE float3_X observationDirection (const int observation_id_extern)`

Compute observation angles.

This function is used in the transition radiation plugin kernel to compute the observation directions given as a unit vector pointing towards a 'virtual' detector

This default setup is an example of a 2D detector array. It computes observation directions for 2D virtual detector field with its center pointing toward the +y direction (for theta=0, phi=0) with observation angles ranging from theta = [angle_theta_start : angle_theta_end] phi = [angle_phi_start : angle_phi_end] Every observation_id_extern index moves the phi angle from its start value toward its end value until the observation_id_extern reaches N_split. After that the theta angle moves further from its start value towards its end value while phi is reset to its start value.

The unit vector pointing towards the observing virtual detector can be described using theta and phi by: x_value = sin(theta) * cos(phi) y_value = cos(theta) z_value = sin(theta) * sin(phi) These are the standard spherical coordinates.

The example setup describes an detector array of 128X128 detectors ranging from 0 to pi for the azimuth angle theta and from 0 to 2 pi for the polar angle phi.

If the calculation is only supposed to be done for a single azimuth or polar angle, it will use the respective minimal angle.

Parameters
`observation_id_extern` – int index that identifies each block on the GPU to compute the observation direction

Returns
unit vector pointing in observation direction type: float3_X

struct **`GammaFilterFunctor`**

example of a filter for the relativistic Lorentz factor gamma

Public Functions

`template<typename T_Particle> inline HDINLINE void operator() (T_Particle &particl`

Public Static Attributes

static constexpr float_X **`filterGamma`** = 5.0

Gamma value above which the radiation is calculated, must be positive.

namespace **`linearFrequencies`**

units for linear frequencies distribution for transition radiation plugin

Variables

constexpr unsigned int **`nOmega`** = 512

number of frequency values to compute in the linear frequency [unitless]

namespace **`SI`**

Variables

constexpr float_64 **`omegaMin`** = 0.0

mimimum frequency of the linear frequency scale in units of [1/s]

constexpr float_64 **`omegaMax`** = 1.06e16

maximum frequency of the linear frequency scale in units of [1/s]

namespace **`listFrequencies`**

units for frequencies from list for transition radiation calculation

Variables

constexpr char **`listLocation`**[] = "/path/to/frequency_list"

path to text file with frequencies

constexpr unsigned int **`nOmega`** = 512

number of frequency values to compute if frequencies are given in a file [unitless]

namespace **`logFrequencies`**

units for logarithmic frequencies distribution for transition radiation plugin

Variables

constexpr unsigned int **nOmega** = 256

number of frequency values to compute in the logarithmic frequency [unitless]

namespace **SI**

Variables

constexpr float_64 **omegaMin** = 1.0e13

mimimum frequency of the logarithmic frequency scale in units of [1/s]

constexpr float_64 **omegaMax** = 1.0e17

maximum frequency of the logarithmic frequency scale in units of [1/s]

namespace **parameters**

selected mode of frequency scaling:

unit for foil position

options:

- linearFrequencies
- logFrequencies
- listFrequencies correct treatment of coherent radiation from macro particles

These formfactors are the same as in the radiation plugin! Choose different form factors in order to consider different particle shapes for radiation

- picongpu::plugins::radiation::radFormFactor_CIC_3D … CIC charge distribution
- ::picongpu::plugins::radiation::radFormFactor_TSC_3D … TSC charge distribution
- ::picongpu::plugins::radiation::radFormFactor_PCS_3D … PCS charge distribution
- ::picongpu::plugins::radiation::radFormFactor_CIC_1Dy … only CIC charge distribution in y
- ::picongpu::plugins::radiation::radFormFactor_Gauss_spherical … symmetric Gauss charge distribution
- ::picongpu::plugins::radiation::radFormFactor_Gauss_cell … Gauss charge distribution according to cell size
- ::picongpu::plugins::radiation::radFormFactor_incoherent … only incoherent radiation
- ::picongpu::plugins::radiation::radFormFactor_coherent … only coherent radiation

Variables

constexpr unsigned int **nPhi** = 128

Number of observation directions.

If nPhi or nTheta is equal to 1, the transition radiation will be calculated for phiMin or thetaMin respectively.

constexpr unsigned int **nTheta** = 128

constexpr unsigned int **nObserver** = *nPhi * nTheta*

constexpr float_64 **thetaMin** = 0.0

constexpr float_64 **thetaMax** = *picongpu::PI*

constexpr float_64 **phiMin** = 0.0

constexpr float_64 **phiMax** = 2 * *picongpu::PI*

namespace **SI**

Variables

constexpr float_64 **foilPosition** = 0.0

9.4.5 Misc

starter.param

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **defaultPIConGPU**

random.param

Configure the pseudorandom number generator (PRNG).

Allows to select method and global seeds in order to vary the initial state of the parallel PRNG.

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

namespace **random**

Typedefs

using **Generator** = *pmacc*::random::methods::XorMin<>

Random number generation methods.

It is not allowed to change the method and restart an already existing checkpoint.

- pmacc::random::methods::XorMin
- pmacc::random::methods::MRG32k3aMin
- pmacc::random::methods::AlpakaRand

using **SeedGenerator** = seed::Value<42>

random number start seed

Generator to create a seed for the random number generator. Depending of the generator the seed is reproducible or or changed with each program execution.

- seed::Value< 42 >
- seed::FromTime
- seed::FromEnvironment

physicalConstants.param

namespace **picongpu**

rate calculation from given atomic data, extracted from flylite, based on FLYCHK

References:

- Axel Huebl flylite, not yet published
  - R. Mewe. "Interpolation formulae for the electron impact excitation of ions in

    the H-, He-, Li-, and Ne-sequences." Astronomy and Astrophysics 20, 215 (1972)
  - H.-K. Chung, R.W. Lee, M.H. Chen. "A fast method to generate collisional excitation cross-sections of

    highly charged ions in a hot dense matter" High Energy Dennsity Physics 3, 342-352 (2007)

Note: this file uses the same naming convention for updated and incident field as Solver.kernel.

Note: In this file we use camelCase "updatedField" in both code and comments to denote field E or B that is being updated (i.e. corrected) in the kernel. The other of the two fields is called "incidentField". And for the incidentField source we explicitly use "functor" to not confuse it with the field itself. Please refer to https://picongpu.readthedocs.io/en/latest/models/total_field_scattered_field.html for theoretical background of this procedure.

Variables

constexpr float_64 **PI** = 3.141592653589793238462643383279502884197169399

constexpr float_64 **UNIT_SPEED** = *SI*::*SPEED_OF_LIGHT_SI*

Unit of speed.

constexpr float_X **SPEED_OF_LIGHT** = float_X(*SI*::*SPEED_OF_LIGHT_SI* / *UNIT_SPEED*)

constexpr float_64 **UNITCONV_keV_to_Joule** = 1.60217646e-16

constexpr float_64 **UNITCONV_eV_to_Joule** = *UNITCONV_keV_to_Joule* * 1e-3

constexpr float_64 **UNITCONV_Joule_to_keV** = (1.0 / *UNITCONV_keV_to_Joule*)

constexpr float_64 **UNITCONV_AU_to_eV** = 27.21139

constexpr float_64 **UNITCONV_eV_to_AU** = (1.0 / *UNITCONV_AU_to_eV*)

namespace **SI**

Variables

constexpr float_64 **SPEED_OF_LIGHT_SI** = 2.99792458e8

unit: m / s

constexpr float_64 **MUE0_SI** = *PI* * 4.e-7

unit: N / A^2

constexpr float_64 **EPS0_SI** = 1.0 / *MUE0_SI* / *SPEED_OF_LIGHT_SI* / *SPEED_OF_LIGHT_SI*

unit: C / (V m)

constexpr float_64 **Z0_SI** = *MUE0_SI* * *SPEED_OF_LIGHT_SI*

impedance of free space unit: ohm

constexpr float_64 **HBAR_SI** = 1.054571800e-34

reduced Planck constant unit: J * s

constexpr float_64 **ELECTRON_MASS_SI** = 9.109382e-31

unit: kg

constexpr float_64 **ELECTRON_CHARGE_SI** = -1.602176e-19

unit: C

constexpr float_64 **ATOMIC_UNIT_ENERGY** = 4.36e-18

constexpr float_64 **BOHR_RADIUS** = 5.292e-7

bohr radius, unit: m

constexpr float_64 **ATOMIC_UNIT_EFIELD** = 5.14e11

constexpr float_64 **ATOMIC_UNIT_TIME** = 2.4189e-17

constexpr float_64 **N_AVOGADRO** = 6.02214076e23

Avogadro number unit: mol^-1.

Y. Azuma et al. Improved measurement results for the Avogadro constant using a 28-Si-enriched crystal, Metrologie 52, 2015, 360-375 doi:10.1088/0026-1394/52/2/360

constexpr float_64 **ELECTRON_RADIUS_SI** = *ELECTRON_CHARGE_SI* * *ELECTRON_CHARGE_SI* / (4.0 * *PI* * *EPS0_SI* * *ELECTRON_MASS_SI* * *SPEED_OF_LIGHT_SI* * *SPEED_OF_LIGHT_SI*)

Classical electron radius in SI units.

CHAPTER

TEN

PLUGINS

Plugin name	short description
binning[2][6]	particle binning plugin to make histograms with user-defined axes and quantity
charge conservation[5]	maximum difference between electron charge density and div E
checkpoint[2]	stores the primary data of the simulation for restarts.
count particles[5]	count total number of macro particles
count per supercell[2]	count macro particles *per supercell*
energy histogram[6]	energy histograms for electrons and ions
energy fields	electromagnetic field energy per time step
energy particles[6]	kinetic and total energies summed over all electrons and/or ions
ISAAC	interactive 3D live visualization [Matthes2016]
openPMD[2][6]	outputs simulation data via the openPMD API
particle calorimeter[2][3][6]	spatially resolved, particle energy detector in infinite distance
phase space[2][5][6]	calculate 2D phase space [Huebl2014]
PNG[6]	pictures of 2D slices
radiation[2]	compute emitted electromagnetic spectra [Pausch2012] [Pausch2014] [Pausch2018]
shadowgraphy[6]	compute synthetic shadowgrams [Carstens2022]
slice emittance	compute emittance and slice emittance of particles
sum currents[5]	compute the total current summed over all cells
transitionRadiation	compute emitted electromagnetic spectra

10.1 Binning

This binning plugin is a flexible binner for particles properties. Users can

- Define their own axes
- Define their own quantity which is binned
- Choose which species which are used for the binning
- Choose how frequently they want the binning to be executed
- Choose if the binning should be time averaging or normalized by bin volume
- Write custom output to file, for example other quantites related to the simulation which the user is interested in
- Execute multiple binnings at the same time

[2] Requires PIConGPU to be compiled with openPMD API.
[6] Multi-Plugin: Can be configured to run multiple times with varying parameters.
[5] Only runs on the *CUDA* backend (GPU).
[3] Can remember particles that left the box at a certain time step.

10.1.1 User Input

Users can set up their binning in the `binningSetup.param` file. After setup, PIConGPU needs to be recompiled.

Attention: Unlike other plugins, the binning plugin doesn't provide any runtime configuration. To set up binning, users need to define it in the param file and then recompile.

A binner is created using the `addBinner()` function, which describes the configuration options available to the user to set up the binning. Multiple binnings can be run at the same time by simply calling `addBinner()` multiple times with different parameters.

class **`BinningCreator`**

An object of this class is provided to the user to add their binning setups.

Public Functions

template<typename **`TAxisTuple`**, typename **`TSpeciesTuple`**, typename **`TDepositionData`**>
inline auto **`addBinner`**`(`std::string binnerOutputName, *TAxisTuple* axisTupleObject, *TSpeciesTuple* speciesTupleObject, *TDepositionData* depositionData, std::function<void(::openPMD::Series &series, ::openPMD::Iteration &iteration, ::openPMD::Mesh &mesh)> writeOpenPMDFunctor = `[](::openPMD::Series &series, ::openPMD::Iteration &iteration, ::openPMD::Mesh &mesh) {})` -> BinningData<*TAxisTuple*, *TSpeciesTuple*, *TDepositionData*>&

Creates a binner from user input and adds it to the vector of all binners.

Parameters

- **`binnerOutputName`** – filename for openPMD output. It must be unique or will cause overwrites during data dumps and undefined behaviour during restarts
- **`axisTupleObject`** – tuple holding the axes
- **`speciesTupleObject`** – tuple holding the species to do the binning with
- **`depositionData`** – functorDescription of the deposited quantity
- **`notifyPeriod`** – The periodicity of the output
- **`dumpPeriod`** – The number of notify steps to accumulate over. Dump at the end. Defaults to 1.
- **`timeAveraging`** – Time average the accumulated data when doing the dump. Defaults to true.
- **`normalizeByBinVolume`** – defaults to true
- **`writeOpenPMDFunctor`** – Functor to write out user specified openPMD data

A most important parts of defining a binning are the axes (the axes of the histogram which define the bins) and the deposited quantity (the quantity to be binned). Both of these are described using the "Functor Description".

Functor Description

The basic building block for the binning plugin is the Functor Description object, and it is used to describe both axes and the deposited quantity. It describes the particle properties which we find interesting and how we can calculate/get this property from the particle. A functor description is created using createFunctorDescription.

template<typename **QuantityType**, typename **FunctorType**>
HINLINE auto *picongpu*::*plugins*::binning::**createFunctorDescription**(*FunctorType* functor, std::string name, std::array<double, 7> units = std::array<double, 7>({0., 0., 0., 0., 0., 0., 0.}))

Describes the functors, units and names for the axes and the deposited quantity.

Todo:

infer T_Quantity from T_Functor, needs particle type also, different particles may have different return types

Template Parameters

- **QuantityType** – The type returned by the functor
- **FunctorType** – Automatically deduced type of the functor

Parameters

- **functor** – Functor which access the particle property
- **name** – Name for the functor/axis written out with the openPMD output.
- **units** – The dimensionality of the quantity returned by the functor in the 7D format. Defaults to unitless.

Returns

FunctorDescription object

Functor

The functor needs to follow the signature shown below. This provides the user access to the particle object and with information about the domain.

```
auto myFunctor = [] ALPAKA_FN_ACC(auto const& domainInfo, auto const& worker, auto␣
↪const& particle) -> returnType
{
    // fn body
    return myParameter;
};
```

The return type is defined by the user.

Domain Info

Enables the user to find the location of the particle (in cells) in the simulation domain. Contains

class **DomainInfo**

Provides knowledge of the simulation domain to the user.

Public Functions

inline DINLINE **DomainInfo**(uint32_t simStep, *pmacc*::*DataSpace*<SIMDIM> gOffset, *pmacc*::*DataSpace*<SIMDIM> lOffset, DataSpace<SIMDIM> physicalSuperCellIdx)

Parameters

physicalSuperCellIdx – supercell index relative to the border origin

Public Members

uint32_t **currentStep**

Current simulation timestep.

pmacc::*DataSpace*<SIMDIM> **globalOffset**

Offset of the global domain on all GPUs.

pmacc::*DataSpace*<SIMDIM> **localOffset**

Offset of the domain simulated on current GPU.

pmacc::*DataSpace*<SIMDIM> **blockCellOffset**

Offset of domain simulated by current block wrt the border.

The global and local offsets can be understood by lookng at the PIConGPU domain definitions.

Dimensionality and units

Users can specify the units of their functor output using a 7 dimensional array. Each element of the array corresponds an SI base unit, and the value stored in that index is the exponent of the unit. The dimensional base quantities are defined as in `SIBaseUnits_t` following the international system of quantities (ISQ). If no units are given, the quantity is assumed to be dimensionless.

```
std::array<double, 7> momentumDimension{};
momentumDimension[SIBaseUnits::length] = 1.0;
momentumDimension[SIBaseUnits::mass] = 1.0;
momentumDimension[SIBaseUnits::time] = -1.0;
```

enum *picongpu*::traits::SIBaseUnits::**SIBaseUnits_t**

Values:

enumerator **length**

enumerator **mass**

enumerator **time**

enumerator **electricCurrent**

enumerator **thermodynamicTemperature**

enumerator **amountOfSubstance**

enumerator **luminousIntensity**

Axis

Axis is a combination of a *functor description* and an *axis splitting* These are brought together by createAxis functions, depending on what kind of an axis you want. The name used in the functor description is used as the name of the axis for openPMD.

Attention: The return type of the functor as specified in the functor description is required to be the same as the type of the range (min, max).

Currently implemented axis types

- Linear Axis

template<typename **T_Attribute**, typename **T_AttrFunctor**>

class **LinearAxis**

Linear axis with contiguous fixed sized bins.

Axis splitting is defined with min, max and n_bins. Bin size = (max-min)/n_bins. Bins are closed open [) intervals [min, min + size), [min + size, min + 2*size) ,. . . , [max-size, max). Allocates 2 extra bins, for under and overflow. These are bin index 0 and (n_bins+2)-1

Binning can be done over an arbitrary number of axes, by creating a tuple of all the axes. Limited by memory depending on number of bins in each axis.

Axis Splitting

Defines the axis range and how it is split into bins. In the future this plugin will support other ways to split the domain, eg. using the binWidth or by auto-selecting the parameters.

template<typename **T_Data**>

class **AxisSplitting**

Holds the axis range in SI units and information on how this range is split into bins.

Public Members

Range<*T_Data*> **m_range**

Range object in SI units.

uint32_t **nBins**

Number of bins in range.

const bool **enableOverflowBins**

Enable or Disable overflow bins.

Number of overflow bis is the responsibility of the axis implementaiton Defaults to true

Range

template<typename **T_Data**>

class **Range**

Holds the range in SI Units in axis space over which the binning will be done.

Public Members

T_Data **min**

Minimum of binning range in SI Units.

T_Data **max**

Maximum of binning range in SI Units.

Species

PIConGPU species which should be used in binning. Species can be instances of a species type or a particle species name as a PMACC_CSTRING. For example,

```
auto electronsObj = PMACC_CSTRING("e"){};
```

Note: Some parameters (axes and species) are given in the form of tuples. These are just a collection of objects and are of arbitrary size. Users can make a tuple by using the `createTuple()` function and passing in the objects as arguments.

Deposited Quantity

Quantity to be deposited is simply a *functor description*.

Notify period

Set the periodicity of the output. Follows the period syntax defined *here*.

Dump Period

Defines the number of notify steps to accumulate over. Note that this is not accumulating over actual PIC iterations, but over the notify periods. If time averaging is enabled, this is also the period to do time averaging over. For example a value of 10 means that after every 10 notifies, an accumulated file will be written out. If PIConGPU exits before executing 10 notifies, then there will be no output. The plugin dumps on every notify if this is set to either 0 or 1.

Time Averaging

When dumping the accumulated output, whether or not to divide by the dump period, i.e. do a time averaging.

Attention: The user needs to set a dump period to enable time averaging.

Normalize by Bin Volume

Since it is possible to have non-uniformly sized axes, it makes sense to normalize the binned quantity by the bin volume to enable a fair comparison between bins.

writeOpenPMDFunctor

Users can also write out custom output to file, for example other quantites related to the simulation which the user is interested in. This is a lambda with the following signature.

```
[=](::openPMD::Series& series, ::openPMD::Iteration& iteration, ::openPMD::Mesh&␣
↪mesh) -> void
```

Note: Make sure to capture by copy only, as the objects defined in the param file are not kept alive

10.1.2 OpenPMD Output

The binning outputs are stored in HDF5 files in `simOutput/binningOpenPMD/` directory.

The files are named as `<binnerOutputName>_<timestep>.h5`.

The OpenPMD mesh is call "Binning".

The outputs in written in SI units.

If normalization is enabled, the output is normalized by the bin volume.

The output histogram has 2 bins more in each dimension than the user-defined `nBins` in that dimension, to deal with under and overflow.

The number of bin edges written out for an axis is one more than the user-defined `nBins`. These represent the bins in [min,max]. Since there are actually `nBins + 2` bins, two edges are not written out. These are the first and last edge, corresponding to the overflow bins, and they have the value of -inf and + inf.

Attribute	Description
`unitSI`	Scaling factor for the deposited quantity to convert to SI
`<axisName>_bin_edges`	The edges of the bins of an axis in SI units
`<axisName>_units`	The units of an axis

10.2 Charge Conservation

First the charge density of all species with respect to their shape function is computed. Then this charge density is compared to the charge density computed from the divergence of the electric field $\nabla\vec{E}$. The maximum deviation value multiplied by the cell's volume is printed.

Attention: This plugin assumes a Yee-like divergence E stencil!

10.2.1 .cfg file

PIConGPU command line argument (for `.cfg` files):

```
--chargeConservation.period <periodOfSteps>
```

10.2.2 Memory Complexity

Accelerator

no extra allocations (needs at least one FieldTmp slot).

Host

negligible.

10.2.3 Output and Analysis Tools

A new file named `chargeConservation.dat` is generated:

```
#timestep max-charge-deviation unit[As]
0 7.59718e-06 5.23234e-17
100 8.99187e-05 5.23234e-17
200 0.000113926 5.23234e-17
300 0.00014836 5.23234e-17
400 0.000154502 5.23234e-17
500 0.000164952 5.23234e-17
```

The charge is normalized to `UNIT_CHARGE` (third column) which is the typical charge of *one* macro-particle.

There is a up 5% difference to a native hdf5 post-processing based implementation of the charge conversation check due to a different order of subtraction. And the zero-th time step (only numerical differences) might differ more then 5% relative due to the close to zero result.

10.3 Checkpoint

Stores the primary data of the simulation for restarts. Primary data includes:

- electro-magnetic fields
- particle attributes
- state of random number generators and particle ID generator
- …

Note: Some plugins have their own internal state. They will be notified on checkpoints to store their state themselves.

10.3.1 What is the format of the created files?

We write our fields and particles in an open markup called *openPMD*.

For further details, see *the openPMD API* section.

10.3.2 External Dependencies

The plugin is available as soon as the *openPMD API library* is compiled in.

10.3.3 .cfg file

You can use `--checkpoint.period` or *–checkpoint.timePeriod* to enable checkpointing.

PIConGPU command line option	Description
`--checkpoint.period <N>`	Create checkpoints every N steps.
`--checkpoint.timePeriod <M>`	Create checkpoints every M minutes.
`--checkpoint.backend <IO-backend>`	IO-backend used to create the checkpoint.
`--checkpoint.directory <string>`	Directory inside `simOutput` for writing checkpoints. Default is `checkpoints`.
`--checkpoint.file <string>`	Relative or absolute fileset prefix for writing checkpoints. If relative, checkpoint files are stored under `simOutput/<checkpoint-directory>`. Default depends on the selected IO-backend.
`--checkpoint.restart`	Restart a simulation from the latest checkpoint (requires a valid checkpoint).
`--checkpoint.tryRestart`	Restart a simulation from the latest checkpoint if available else start from scratch.
`--checkpoint.restart.step <N>`	Select a specific restart checkpoint.
`--checkpoint.restart.backend <IO-backend>`	IO-backend used to load a existent checkpoint.
`--checkpoint.restart.directory <string>`	Directory inside `simOutput` containing checkpoints for a restart. Default is `checkpoints`.
`--checkpoint.restart.file <string>`	Relative or absolute fileset prefix for reading checkpoints. If relative, checkpoint files are searched under `simOutput/<checkpoint-directory>`. Default depends on the selected IO-backend``.
`--checkpoint.restart.chunkSize <N>`	Number of particles processed in one kernel call during restart to prevent frame count blowup.
`--checkpoint.restart.loop <N>`	Number of times to restart the simulation after simulation has finished. This mode is intended for visualization and not all plugins support it.
`--checkpoint.<IO-backend>.*`	Additional options to control the IO-backend

Depending on the available external dependencies (see above), the options for the *<IO-backend>* are:

- *openPMD*

10.3.4 Interacting Manually with Checkpoint Data

Note: Interacting with the *raw data of checkpoints* for manual manipulation is considered an advanced feature for experienced users.

Contrary to regular output, checkpoints contain additional data which might be confusing on the first glance. For example, some comments might be missing, all data from our concept of slides for moving window simulations will be visible, additional data for internal states of helper classes is stored as well and index tables such as openPMD particle patches are essential for parallel restarts.

10.4 Count Particles

This plugin counts the total number of *macro particles associated with a species* and writes them to a file for specified time steps. It is used mainly for debugging purposes. Only in case of constant particle density, where each macro particle describes the same number of real particles (weighting), conclusions on the plasma density can be drawn.

10.4.1 .cfg file

The *CountParticles* plugin is always complied for all species. By specifying the periodicity of the output using the command line argument `--e_macroParticlesCount.period` (here for an electron species called `e`) with picongpu, the plugin is enabled. Setting `--e_macroParticlesCount.period 100` adds the number of all electron macro particles to the file *ElectronsCount.dat* for every 100th time step of the simulation.

10.4.2 Memory Complexity

Accelerator

no extra allocations.

Host

negligible.

10.4.3 Output

In the output file `e_macroParticlesCount.dat` there are three columns. The first is the integer number of the time step. The second is the number of macro particles as integer - useful for exact counts. And the third is the number of macro particles in scientific floating point notation - provides better human readability.

10.4.4 Known Issues

Currently, the file `e_macroParticlesCount.dat` is overwritten when restarting the simulation. Therefore, all previously stored counts are lost.

10.5 Count per Supercell

This plugin counts the total number of *macro particles of a species* for each super cell and stores the result in an hdf5 file. Only in case of constant particle weighting, where each macro particle describes the same number of real particles, conclusions on the plasma density can be drawn.

10.5.1 External Dependencies

The plugin is available as soon as the *openPMD API* is compiled in.

10.5.2 .cfg files

By specifying the periodicity of the output using the command line argument `--e_macroParticlesPerSuperCell.period` (here for an electron species `e`) with picongpu the plugin is enabled. Setting `--e_macroParticlesPerSuperCell.period 100` adds the number of all electron macro particles to the file `e_macroParticlesCount.dat` for every 100th time step of the simulation.

Accelerator

an extra permanent allocation of `size_t` for each local supercell.

Host

negligible.

10.5.3 Output

The output is stored as hdf5 file in a separate directory.

10.6 Energy Fields

This plugin computes the total energy contained in the electric and magnetic field of the entire volume simulated. The energy is computed for user specified time steps.

10.6.1 .cfg file

By setting the PIConGPU command line flag `--fields_energy.period` to a non-zero value the plugin computes the total field energy. The default value is `0`, meaning that the total field energy is not stored. By setting e.g. `--fields_energy.period 100` the total field energy is computed for time steps *0, 100, 200, . . .*.

10.6.2 Memory Complexity

Accelerator

negligible.

Host

negligible.

10.6.3 Output

The data is stored in `fields_energy.dat`. There are two columns. The first gives the time step. The second is the total field energy in **Joule**. The first row is a comment describing the columns:

```
#step total[Joule] Bx[Joule] By[Joule] Bz[Joule] Ex[Joule] Ey[Joule] Ez[Joule]
0     2.5e+18      0         0         0         2.5e+18   0         0
100   2.5e+18      2.45e-22  2.26e-08  2.24e-08  2.5e+18   2.29e-08  2.30e-08
```

Attention: The output of this plugin computes a *sum over all cells* in a very naive implementation. This can lead to significant errors due to the finite precision in floating-point numbers. Do not expect the output to be precise to more than a few percent. Do not expect the output to be deterministic due to the statistical nature of the implemented reduce operation.

Please see this issue for a longer discussion and possible future implementations.

10.6.4 Example Visualization

Python example snippet:

```python
import numpy as np
import matplotlib.pyplot as plt


simDir = "path/to/simOutput/"

# Ekin in Joules (see EnergyParticles)
e_sum_ene = np.loadtxt(simDir + "e_energy_all.dat")[:, 0:2]
p_sum_ene = np.loadtxt(simDir + "p_energy_all.dat")[:, 0:2]
C_sum_ene = np.loadtxt(simDir + "C_energy_all.dat")[:, 0:2]
N_sum_ene = np.loadtxt(simDir + "N_energy_all.dat")[:, 0:2]
# Etotal in Joules
fields_sum_ene = np.loadtxt(simDir + "fields_energy.dat")[:, 0:2]

plt.figure()
plt.plot(e_sum_ene[:,0], e_sum_ene[:,1], label="e")
plt.plot(p_sum_ene[:,0], p_sum_ene[:,1], label="p")
plt.plot(C_sum_ene[:,0], C_sum_ene[:,1], label="C")
plt.plot(N_sum_ene[:,0], N_sum_ene[:,1], label="N")
plt.plot(fields_sum_ene[:,0], fields_sum_ene[:,1], label="fields")
plt.plot(
    e_sum_ene[:,0],
    e_sum_ene[:,1] + p_sum_ene[:,1] + C_sum_ene[:,1] + N_sum_ene[:,1] + fields_sum_
↪ene[:,1],
    label="sum"
)
plt.legend()
plt.show()
```

10.7 Energy Histogram

This plugin computes the energy histogram (spectrum) of a selected particle species and stores it to plain text files. The acceptance of particles for counting in the energy histogram can be adjusted, e.g. to model the limited acceptance of a realistic spectrometer.

10.7.1 .param file

The *particleFilters.param* file allows to define accepted particles for the energy histogram. A typical *filter* could select particles within a specified *opening angle in forward direction*.

10.7.2 .cfg files

There are several command line parameters that can be used to set up this plugin. Replace the prefix `e` for electrons with any other species you have defined, we keep using `e` in the examples below for simplicity. Currently, the plugin can be set *once for each species*.

PIConGPU command line option	description
`--e_energyHistogram.period`	The output periodicity of the **electron** histogram. A value of `100` would mean an output at simulation time step *0, 100, 200, . . .*. If set to a non-zero value, the energy histogram of all **electrons** is computed. By default, the value is `0` and no histogram for the electrons is computed.
`--e_energyHistogram.filter`	Use filtered particles. Available filters are set up in *particleFilters.param*.
`--e_energyHistogram.binCount`	Specifies the number of bins used for the **electron** histogram. Default is `1024`.
`--e_energyHistogram.minEnergy`	Set the minimum energy for the **electron** histogram in *keV*. Default is `0`, meaning *0 keV*.
`--e_energyHistogram.maxEnergy`	Set the maximum energy for the **electron** histogram in *keV*. There is **no default value**. This has to be set by the user if `--e_energyHistogram.period 1` is set.

Note: This plugin is a multi plugin. Command line parameter can be used multiple times to create e.g. dumps with different dumping period. In the case where an optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. For example,

```
--e_energyHistogram.period 128 --e_energyHistogram.filter all --e_energyHistogram.
↪maxEnergy 10
--e_energyHistogram.period 100 --e_energyHistogram.filter all --e_energyHistogram.
↪maxEnergy 20 --e_energyHistogram.binCount 512
```

creates two plugins:

1. create an electron histogram **with 512 bins** each 128th time step.
2. create an electron histogram **with 1024 bins** (this is the default) each 100th time step.

10.7.3 Memory Complexity

Accelerator

an extra array with the number of bins.

Host

negligible.

10.7.4 Output

The histograms are stored in ASCII files in the `simOutput/` directory.

The file for the electron histogram is named `e_energyHistogram.dat` and for all other species `<species>_energyHistogram.dat` likewise. The first line of these files does not contain histogram data and is commented-out using `#`. It describes the energy binning that needed to interpret the following data. It can be seen as the head of the following data table. The first column is an integer value describing the simulation time step. The second column counts the number of real particles below the minimum energy value used for the histogram. The following columns give the real electron count of the particles in the specific bin described by the first line/header. The second last column gives the number of real particles that have a higher energy than the maximum energy used for the histogram. The last column gives the total number of particles. In total there are 4 columns more than the number of bins specified with command line arguments. Each row describes another simulation time step.

10.7.5 Analysis Tools

Data Reader

You can quickly load and interact with the data in Python with:

```python
from picongpu.plugins.data import EnergyHistogramData

eh_data = EnergyHistogramData('path/to/run_dir') # the directory in which simOutput␣
↪is located

# show available iterations
eh_data.get_iterations(species='e')

# show available simulation times
eh_data.get_times(species='e')

# load data for a given iteration
counts, bins_keV, _, _ = eh_data.get(species='e', species_filter='all',␣
↪iteration=2000)

# get data for multiple iterations
counts, bins_keV, iteration, dt = eh_data.get(species='e', iteration=[200, 400, 8000])

# load data for a given time
counts, bins_keV, iteration, dt = eh_data.get(species='e', species_filter='all',␣
↪time=1.3900e-14)
```

Matplotlib Visualizer

You can quickly plot the data in Python with:

```
from picongpu.plugins.plot_mpl import EnergyHistogramMPL
import matplotlib.pyplot as plt


# create a figure and axes
fig, ax = plt.subplots(1, 1)

## create the visualizer
# pass a list of ('identifier', 'run_dir') tuples to
# visualize several simulations at once
eh_vis = EnergyHistogramMPL(('short identifier for plot label', 'path/to/run_dir'),␣
↪ax)

eh_vis.visualize(iteration=200, species='e')

plt.show()

# specifying simulation time is also possible (granted there is a matching iteration␣
↪for that time)
eh_vis.visualize(time=2.6410e-13, species='e')

plt.show()

# plotting histogram data for multiple simulations simultaneously also works:
eh_vis = EnergyHistogramMPL([
     ("sim1", "path/to/sim1"),
     ("sim2", "path/to/sim2"),
     ("sim3", "path/to/sim3")], ax)
 eh_vis.visualize(species="e", iteration=10000)

 plt.show()
```

The visualizer can also be used from the command line (for a single simulation only) by writing

```
python energy_histogram_visualizer.py
```

with the following command line options

Options	Value
-p	Path to the run directory of a simulation.
-i	An iteration number
-s (optional, defaults to 'e')	Particle species abbreviation (e.g. 'e' for electrons)
-f (optional, defaults to 'all')	Species filter string

Alternatively, PIConGPU comes with a command line analysis tool for the energy histograms. It is based on *gnuplot* and requires that gnuplot is available via command line. The tool can be found in `src/tools/bin/` and is called `BinEnergyPlot.sh`. It accesses the gnuplot script `BinEnergyPlot.gnuplot` in `src/tools/share/gnuplot/`. `BinEnergyPlot.sh` requires exactly three command line arguments:

Argument	Value
1st	Path and filename to `e_energyHistogram.dat` file.
2nd	Simulation time step (needs to exist)
3rd	Label for particle count used in the graph that this tool produces.

Jupyter Widget

If you want more interactive visualization, then start a jupyter notebook and make sure that `ipywidgets` and `ipympl` are installed.

After starting the notebook server write the following

```
# this is required!
%matplotlib widget
import matplotlib.pyplot as plt
plt.ioff()

from IPython.display import display
from picongpu.plugins.jupyter_widgets import EnergyHistogramWidget

# provide the paths to the simulations you want to be able to choose from
# together with labels that will be used in the plot legends so you still know
# which data belongs to which simulation
w = EnergyHistogramWidget(run_dir_options=[
        ("scan1/sim4", scan1_sim4),
        ("scan1/sim5", scan1_sim5)])
display(w)
```

and then interact with the displayed widgets.

10.8 Energy Particles

This plugin computes the **kinetic and total energy summed over all particles** of a species for time steps specified.

10.8.1 .cfg file

Only the time steps at which the total kinetic energy of all particles should be specified needs to be set via command line argument.

PIConGPU command line option	Description
`--e_energy.period 100`	Sets the time step period at which the energy of all **electrons** in the simulation should be simulated. If set to e.g. `100`, the energy is computed for time steps *0, 100, 200, ...*. The default value is `0`, meaning that the plugin does not compute the particle energy.
`--<species>_period 42`	Same as above, for any other species available.
`--<species>_filter`	Use filtered particles. All available filters will be shown with `picongpu --help`

10.8.2 Memory Complexity

Accelerator

negligible.

Host

negligible.

10.8.3 Output

The plugin creates files prefixed with the species' name and the filter name as postfix, e.g. *e_energy_<filterName>.dat* for the electron energies and *p_energy_<filterName>.dat* for proton energies. The file contains a header describing the columns.

```
#step Ekin_Joule E_Joule
0.0   0.0        0.0
```

Following the header, each line is the output of one time step. The time step is given as first value. The second value is the kinetic energy of all particles at that time step. And the last value is the total energy (kinetic + rest energy) of all particles at that time step.

Attention: The output of this plugin computes a *sum over all particles* in a very naive implementation. This can lead to significant errors due to the finite precision in floating-point numbers. Do not expect the output to be precise to more than a few percent. Do not expect the output to be deterministic due to the statistical nature of the implemented reduce operation.

Please see this issue for a longer discussion and possible future implementations.

10.8.4 Example Visualization

Python snippet:

```
import numpy as np

simDir = "path/to/simOutput/"

# Ekin in Joules (see EnergyParticles)
e_sum_ene = np.loadtxt(simDir + "e_energy_all.dat")[:, 0:2]
p_sum_ene = np.loadtxt(simDir + "p_energy_all.dat")[:, 0:2]
C_sum_ene = np.loadtxt(simDir + "C_energy_all.dat")[:, 0:2]
N_sum_ene = np.loadtxt(simDir + "N_energy_all.dat")[:, 0:2]
# Etotal in Joules
fields_sum_ene = np.loadtxt(simDir + "fields_energy.dat")[:, 0:2]

plt.figure()
plt.plot(e_sum_ene[:,0], e_sum_ene[:,1], label="e")
plt.plot(p_sum_ene[:,0], p_sum_ene[:,1], label="p")
plt.plot(C_sum_ene[:,0], C_sum_ene[:,1], label="C")
plt.plot(N_sum_ene[:,0], N_sum_ene[:,1], label="N")
plt.plot(fields_sum_ene[:,0], fields_sum_ene[:,1], label="fields")
plt.plot(
    e_sum_ene[:,0],
```

(continues on next page)

(continued from previous page)

```
    e_sum_ene[:,1] + p_sum_ene[:,1] + C_sum_ene[:,1] + N_sum_ene[:,1] + fields_sum_
↪ene[:,1],
    label="sum"
)
plt.legend()
```

10.9 ISAAC

This is a plugin for the in-situ library ISAAC [Matthes2016] for a live rendering and steering of PIConGPU simulations.

10.9.1 External Dependencies

The plugin is available as soon as the *ISAAC library* is compiled in.

10.9.2 .cfg file

Command line option	Description
`--isaac.period N`	Sets up, that every *N* th timestep an image will be rendered. This parameter can be changed later with the controlling client.
`--isaac.name NAME`	Sets the *NAME* of the simulation, which is shown at the client.
`--isaac.url URL`	*URL* of the required and running isaac server. Host names and IPs are supported.
`--isaac.port PORT`	*PORT* of the isaac server. The default value is `2458` (for the in-situ plugins), but needs to be changed for tunneling reasons or if more than one server shall run on the very same hardware.
`--isaac.width WIDTH`	Setups the *WIDTH* and *HEIGHT* of the created image(s).
`--isaac.height HEIGHT`	Default is `1024x768`.
`--isaac.direct_pause BOOL`	If activated (argument set to `1`) ISAAC will pause directly after the simulation started. Useful for presentations or if you don't want to miss the beginning of the simulation.
`--isaac.quality QUALITY`	Sets the *QUALITY* of the images, which are compressed right after creation. Values between `1` and `100` are possible. The default is `90`, but `70` does also still produce decent results.
`--isaac.timingsFilename NAME`	Enable and write benchmark results into the given file.

The most important settings for ISAAC are `--isaac.period`, `--isaac.name` and `--isaac.url`. A possible addition for your submission `tbg` file could be `--isaac.period 1 --isaac.name !TBG_jobName --isaac.url YOUR_SERVER`, where the tbg variables `!TBG_jobName` is used as name and `YOUR_SERVER` needs to be set up by yourself.

10.9.3 .param file

The ISAAC Plugin has an *isaac.param*, which specifies which fields and particles are rendered. This can be edited (in your local paramSet), but at runtime also an arbitrary amount of fields (in ISAAC called *sources*) can be deactivated. At default every field and every known species are rendered.

10.9.4 Running and steering a simulation

First of all you need to build and run the isaac server somewhere. On HPC systems, simply start the server on the login or head node since it can be reached by all compute nodes (on which the PIConGPU clients will be running).

10.9.5 Functor Chains

One of the most important features of ISAAC are the **Functor Chains**. As most sources (including fields and species) may not be suited for a direct rendering or even full negative (like the electron density field), the functor chains enable you to change the domain of your field source-wise. A date will be read from the field, the functor chain applied and then **only the x-component** used for the classification and later rendering of the scene. Multiply functors can be applied successive with the Pipe symbol `|`. The possible functors are at default:

- **mul** for a multiplication with a constant value. For vector fields you can choose different value per component, e.g. `mul(1,2,0)`, which will multiply the x-component with 1, the y-component with 2 and the z-component with 0. If less parameters are given than components exists, the last parameter will be used for all components without an own parameter.
- **add** for adding a constant value, which works the same as `mul(...)`.
- **sum** for summarizing all available components. Unlike `mul(...)` and `add(...)` this decreases the dimension of the data to `1`, which is a scalar field. You can exploit this functor to use a different component than the x-component for the classification, e.g. with `mul(0,1,0) | sum`. This will first multiply the x- and z-component with 0, but keep the y-component and then merge this to the x-component.
- **length** for calculating the length of a vector field. Like *sum* this functor reduces the dimension to a scalar field, too. However `mul(0,1,0) | sum` and `mul(0,1,0) | length` do not do the same. As `length` does not know, that the x- and z-component are 0 an expensive square root operation is performed, which is slower than just adding the components up.
- **idem** does nothing, it just returns the input data. This is the default functor chain.

Beside the functor chains the client allows to setup the weights per source (values greater than 6 are more useful for PIConGPU than the default weights of 1), the classification via transfer functions, clipping, camera steering and to switch the render mode to iso surface rendering. Furthermore interpolation can be activated. However this is quite slow and most of the time not needed for non-iso-surface rendering.

10.9.6 Memory Complexity

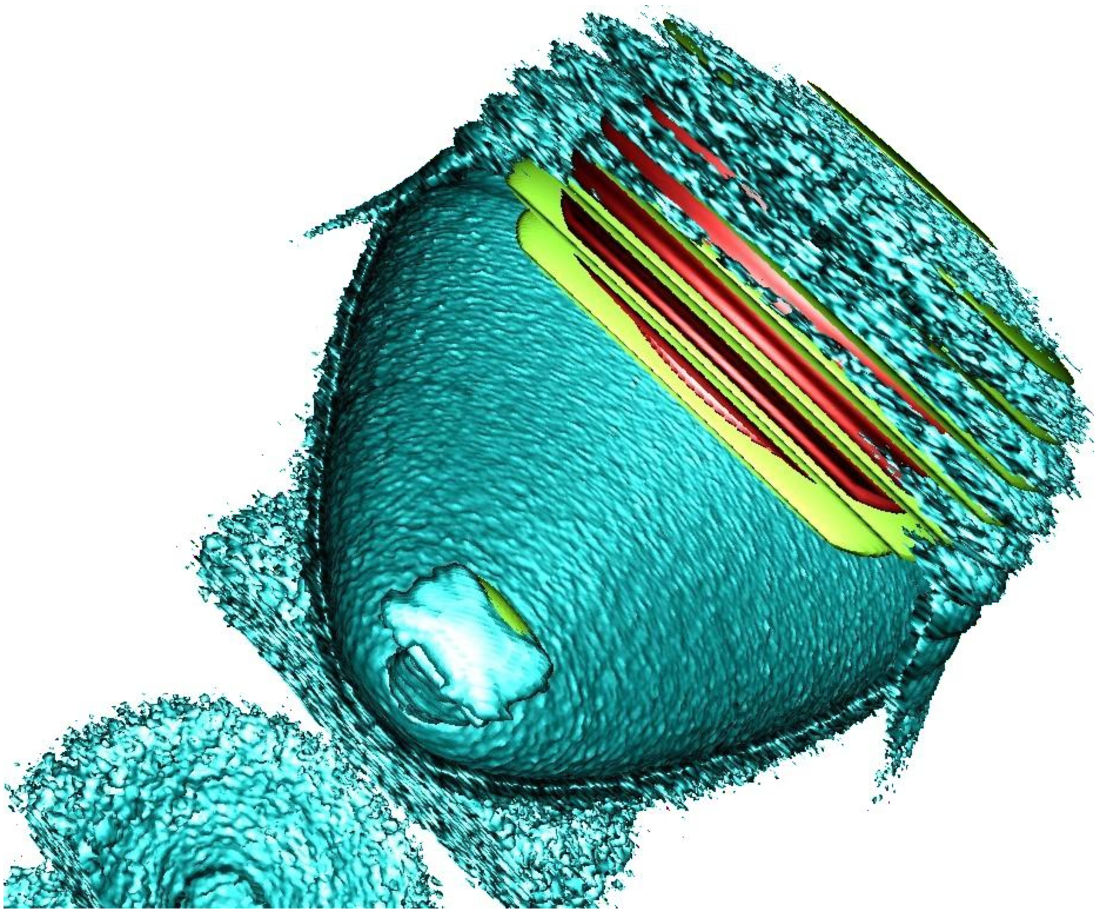
Accelerator

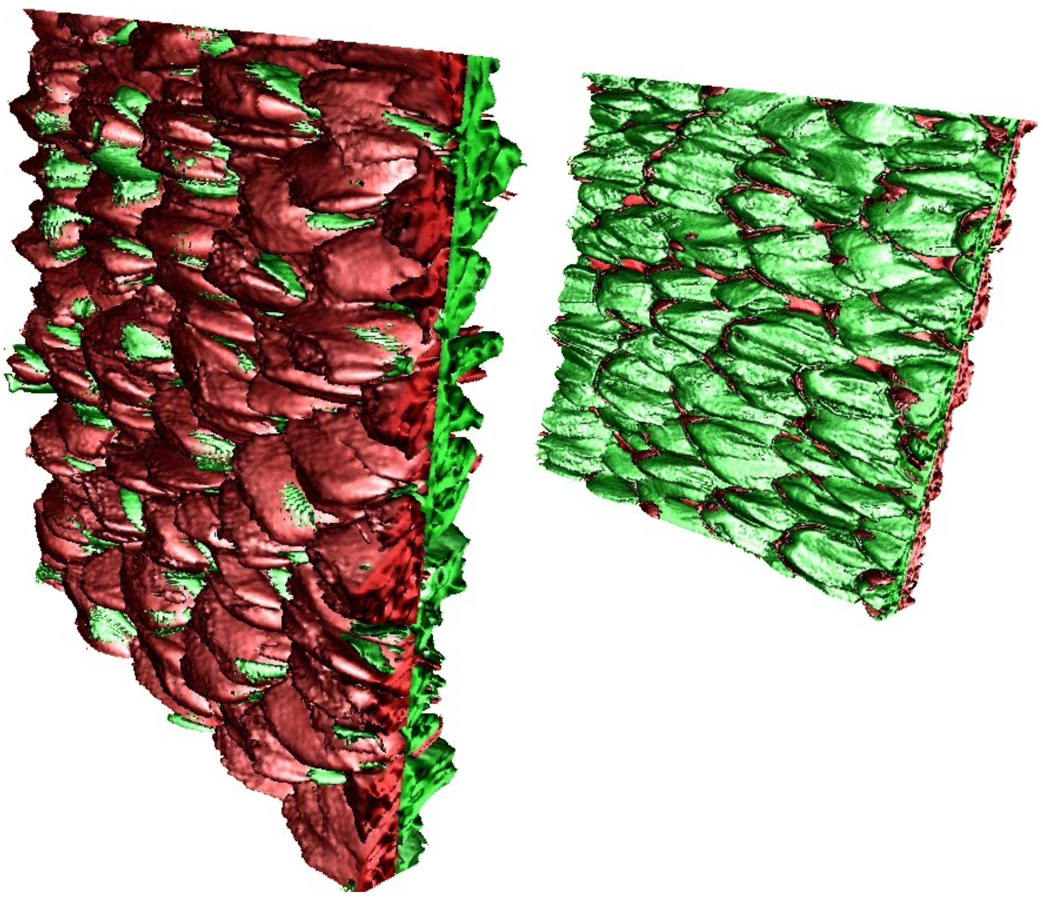
locally, a framebuffer with full resolution and 4 byte per pixel is allocated. For each `FieldTmp` derived field and `FieldJ` a copy is allocated, depending on the input in the *isaac.param* file.

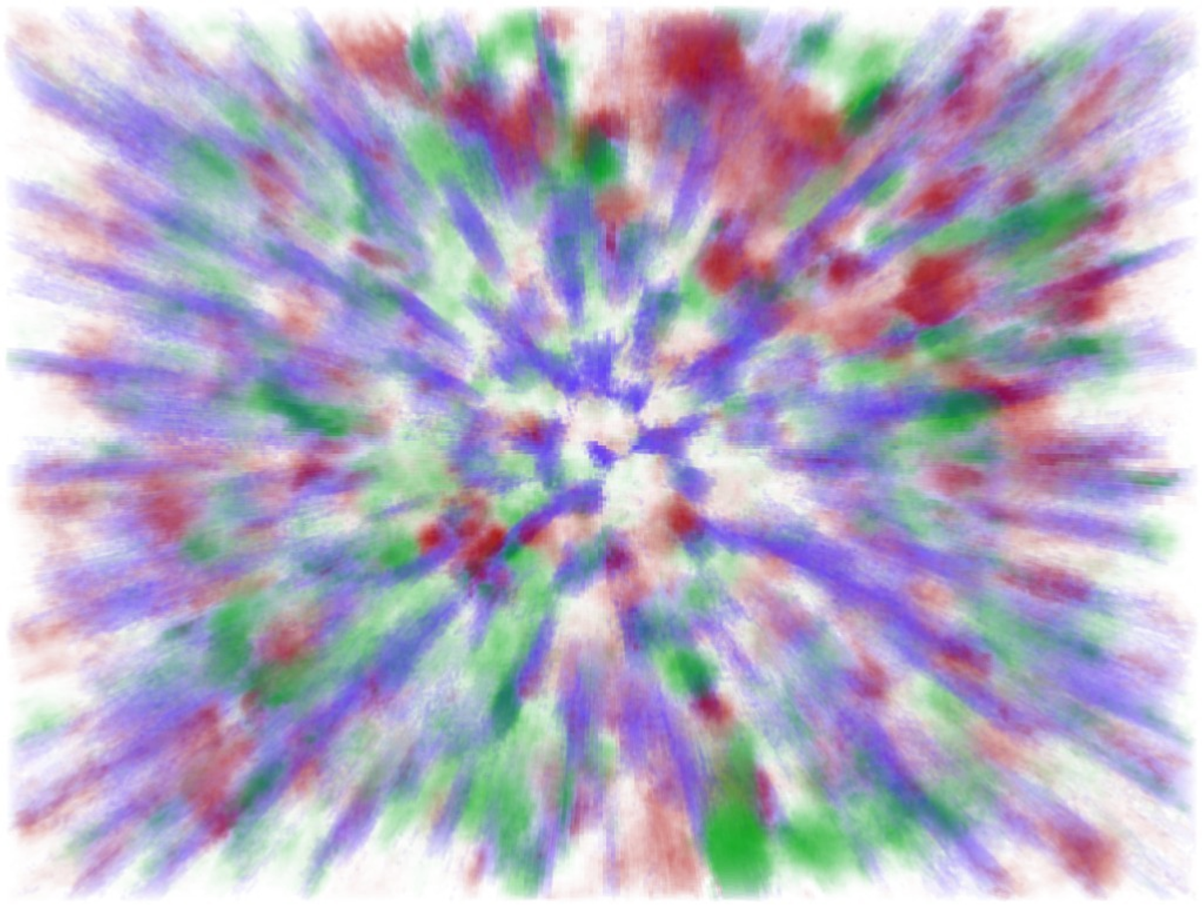
Host

negligible.

10.9.7 Example renderings







10.9.8 References

10.10 openPMD

Stores simulation data such as fields and particles according to the openPMD standard using the openPMD API.

10.10.1 External Dependencies

The plugin is available as soon as the *openPMD API* is compiled in. If the openPMD API is found in version 0.13.0 or greater, PIConGPU will support streaming IO via openPMD.

10.10.2 .param file

The corresponding `.param` file is *fileOutput.param*.

One can e.g. disable the output of particles by setting:

```
/* output all species */
using FileOutputParticles = VectorAllSpecies;
/* disable */
using FileOutputParticles = MakeSeq_t< >;
```

Particle filters used for output plugins, including this one, are defined in *particleFilters.param*. Also see *common patterns of defining particle filters*.

10.10.3 .cfg file

Note that all the following command line parameters can *alternatively* be specified in a `.toml` configuration file. See the next section for further information: *Configuring the openPMD plugin with a TOML configuration file>*

You can use `--openPMD.period` to specify the output period. The base filename is specified via `--openPMD.file`. The openPMD API will parse the file name to decide the chosen backend and iteration layout:

- The filename extension will determine the backend.
- The openPMD will either create one file encompassing all iterations (group-based iteration layout) or one file per iteration (file-based iteration layout). The filename will be searched for a pattern describing how to derive a concrete iteration's filename. If no such pattern is found, the group-based iteration layout will be chosen. Please refer to the documentation of the openPMD API for further information.

In order to set defaults for these value, two further options control the filename:

- `--openPMD.ext` sets the filename extension. Possible extensions include `bp` for the ADIOS2 backend (default), `h5` for HDF5 and `sst` for Streaming via ADIOS2/SST. In case your openPMD API supports both ADIOS1 and ADIOS2, make sure that environment variable `OPENPMD_BP_BACKEND` is not set to ADIOS1.
- `--openPMD.infix` sets the filename pattern that controls the iteration layout, default is "_%06T" for a six-digit number specifying the iteration. Leave empty to pick group-based iteration layout. Since passing an empty string may be tricky in some workflows, specifying `--openPMD.infix=NULL` is also possible.

  Note that streaming IO requires group-based iteration layout in openPMD, i.e. `--openPMD.infix=NULL` is mandatory. If PIConGPU detects a streaming backend (e.g. by `--openPMD.ext=sst`), it will automatically set `--openPMD.infix=NULL`, overriding the user's choice. Note however that the ADIOS2 backend can also be selected via `--openPMD.json` and via environment variables which PIConGPU does not check. It is hence recommended to set `--openPMD.infix=NULL` explicitly.

Option `--openPMD.source` controls which data is output. Its value is a comma-separated list of combinations of a data set name and a filter name. A user can see all possible combinations for the current setup in the command-line help for this option. Note that addding species and particle filters to `.param` files will automatically extend the number of combinations available. By default all particles and fields are output.

For example, `--openPMD.period 128 --openPMD.file simData --openPMD.source 'species_all'` will write only the particle species data to files of the form `simData_000000.bp`, `simData_000128.bp` in the default simulation output directory every 128 steps. Note that this plugin will only be available if the openPMD API is found during compile configuration.

openPMD backend-specific settings may be controlled via two mechanisms:

- Environment variables. Please refer to the backends' documentations for information on environment variables understood by the backends.
- Backend-specific runtime parameters may be set via JSON in the openPMD API. PIConGPU exposes this via the command line option `--openPMD.json`. Please refer to the openPMD API's documentation for further information.

The JSON parameter may be passed directly as a string, or by filename. The latter case is distinguished by prepending the filename with an at-sign `@`. Specifying a JSON-formatted string from within a `.cfg` file can be tricky due to colliding escape mechanisms. An example for a well-escaped JSON string as part of a `.cfg` file is found below.

```
TBG_openPMD="--openPMD.period 100   \
             --openPMD.file simOutput \
             --openPMD.ext bp \
             --openPMD.json '{ \
                 \"adios2\": { \
                   \"dataset\": { \
                     \"operators\": [ \
                       { \
                         \"type\": \"bzip2\" \
                       } \
                     ] \
                   }, \
                   \"engine\": { \
                     \"type\": \"file\", \
                     \"parameters\": { \
                       \"BufferGrowthFactor\": \"1.2\", \
                       \"InitialBufferSize\": \"2GB\" \
                     } \
                   } \
                 } \
               }'"
```

PIConGPU further defines an **extended format for JSON options** that may alternatively used in order to pass dataset-specific configurations. For each backend `<backend>`, the backend-specific dataset configuration found under `config["<backend>"]["dataset"]` may take the form of a JSON list of patterns: `[<pattern_1>, <pattern_2>, ...]`.

Each such pattern `<pattern_i>` is a JSON object with key `cfg` and optional key `select`: `{"select": <pattern>, "cfg": <cfg>}`.

In here, `<pattern>` is a regex or a list of regexes, as used by POSIX `grep -E`. `<cfg>` is a configuration that will be forwarded as-is to openPMD.

The single patterns will be processed in top-down manner, selecting the first matching pattern found in the list. The regexes will be matched against the openPMD dataset path within the iteration (e.g. `E/x` or `particles/.*/position/.*`), considering full matches only.

The **default configuration** is specified by omitting the `select` key. Specifying more than one default is an error. If no pattern matches a dataset, the default configuration is chosen if specified, or an empty JSON object `{}` otherwise.

A full example:

```
{
  "adios2": {
    "engine": {
      "usesteps": true,
      "parameters": {
        "InitialBufferSize": "2Gb",
        "Profile": "On"
      }
    },
    "dataset": [
      {
        "cfg": {
          "operators": [
            {
              "type": "blosc",
              "parameters": {
                "clevel": "1",
                "doshuffle": "BLOSC_BITSHUFFLE"
              }
            }
          ]
        }
      },
      {
        "select": [
          ".*positionOffset.*",
          ".*particlePatches.*"
        ],
        "cfg": {
          "operators": []
        }
      }
    ]
  }
}
```

The extended format is only available for configuration of the writing procedures. The reading procedures (i.e. for restarting from a checkpoint) can be configured via `--checkpoint.openPMD.jsonRestart`, e.g. see the example below for configuring the number of blosc decompression threads in ADIOS2. Note that most sensible use cases for this command line option (including this example) require openPMD-api >= 0.15 (or a recent dev version until the 0.15 release).

```
{
  "adios2": {
    "dataset": {
      "operators": [
        {
          "type": "blosc",
          "parameters": {
            "nthreads": 8
          }
        }
      ]
    }
  }
}
```

Two data preparation strategies are available for downloading particle data off compute devices.

- Set `--openPMD.dataPreparationStrategy doubleBuffer` for use of the strategy that has been optimized for use with ADIOS-based backends. The alias `openPMD.dataPreparationStrategy adios` may be used. This strategy requires at least 2x the GPU main memory on the host side. This is the default.
- Set `--openPMD.dataPreparationStrategy mappedMemory` for use of the strategy that has been optimized for use with HDF5-based backends. This strategy has a small host-side memory footprint (<< GPU main memory). The alias `openPMD.dataPreparationStrategy hdf5` may be used.

PIConGPU command line option	description
`--openPMD.period`	Period after which simulation data should be stored on disk.
`--openPMD.source`	Select data sources and filters to dump. Default is `species_all,fields_all`, which dumps all fields and particle species. Only deterministic filters will be shown.
`--openPMD.range`	Define a contiguous range of cells per dimension to dump. Each dimension is separated by a comma. A range is defined as `[BEGIN:END)` where out of range indices will be clipped. A single value e.g. `10,:,:` for a dimension will only dump a slice. Default is `:,:,:`, which dumps all cells.
`--openPMD.file`	Relative or absolute openPMD file prefix for simulation data. If relative, files are stored under `simOutput`.
`--openPMD.ext`	openPMD filename extension (this controls thebackend picked by the openPMD API).
`--openPMD.infix`	openPMD filename infix (use to pick file- or group-based layout in openPMD). Set to NULL to keep empty (e.g. to pick group-based iteration layout).
`--openPMD.json`	Set backend-specific parameters for openPMD backends in JSON format. Used in writing procedures.
`--checkpoint openPMD.jsonRestart`	Set backend-specific parameters for openPMD backends in JSON format for restarting from a checkpoint.
`--openPMD.dataPreparat`	Strategy for preparation of particle data ('doubleBuffer' or 'mappedMemory'). Aliases 'adios' and 'hdf5' may be used respectively.
`--openPMD.toml`	Alternatively configure the openPMD plugin via a TOML file (see below).

Note: This plugin is a multi plugin. Command line parameter can be used multiple times to create e.g. dumps with different dumping period. Each plugin instance requires that either `--openPMD.period` XOR `--openPMD.toml` is defined. If `--openPMD.toml` is defined, the rest of the configuration for this instance is done via the specified TOML file, no other command line parameters may be passed.

In the case where an optional parameter with a default value is explicitly defined, the parameter will always be passed to the instance of the multi plugin where the parameter is not set. e.g.

```
--openPMD.period 128 --openPMD.file simData1 --openPMD.source 'species_all'
--openPMD.period 1000 --openPMD.file simData2 --openPMD.source 'fields_all' --openPMD.
↪ext h5
```

creates two plugins:

1. dump all species data each 128th time step, use HDF5 backend.
2. dump all field data each 1000th time step, use the default ADIOS backend.

10.10.4 Backend-specific notes

ADIOS2

Streaming with the MPI-based SST implementation

The SST engine (sustainable staging transport) of ADIOS2 has a number of possible backends, including an MPI data transport (still in development at the time of writing this 2023-01-06). It internally uses MPI to set up communcation between data producer (PIConGPU) and consumer via `MPI_Open_Port()` and `MPI_Comm_Accept()`.

Use of this engine requires availability of threaded MPI, which can be activated in PIConGPU via the environment variable `PIC_USE_THREADED_MPI`. Its possible values include `"MPI_THREAD_SINGLE"`, `"MPI_THREAD_FUNNELED"`, `"MPI_THREAD_SERIALIZED"` and `"MPI_THREAD_MULTIPLE"`. The specified value determines the `int required` parameter that will be used in the initialization of MPI via `int MPI_Init_thread( int *argc, char ***argv, int required, int *provided )`. For support of MPI-based SST, the variable must be specified as `PIC_USE_THREADED_MPI=MPI_THREAD_MULTIPLE`.

Additionally, a workaround is required in order to let PIConGPU cleanly terminate after using this engine on ECP hardware. This workaround can be activated on such systems via `PIC_WORKAROUND_CRAY_MPI_FINALIZE=1`.

HDF5

Chunking

By default, the openPMD-api uses a heuristic to automatically set an appropriate dataset chunk size. In combination with some MPI-IO backends (e.g. ROMIO), this has been found to cause crashes. To avoid this, PIConGPU overrides the default choice and deactivates HDF5 chunking in the openPMD plugin.

If you want to use chunking, you can ask for it via the following option passed in `--openPMD.json`:

```
{
  "hdf5": {
    "dataset": {
      "chunks": "auto"
    }
  }
}
```

Performance tuning on Summit

In order to avoid a performance bug for parallel HDF5 on the ORNL Summit compute system, a specific version of ROMIO should be chosen and performance hints should be passed:

```
> export OMPI_MCA_io=romio321
> export ROMIO_HINTS=./my_romio_hints
> cat << EOF > ./my_romio_hints
romio_cb_write enable
romio_ds_write enable
cb_buffer_size 16777216
cb_nodes <number_of_nodes>
EOF
```

Replace `<number_of_nodes>` with the number of nodes that your job uses. These settings are applied automatically in the Summit templates found in `etc/picongpu/summit-ornl`. For further information, see the official Summit documentation and this pull request for WarpX.

10.10.5 Performance

On the Summit compute system, specifying `export IBM_largeblock_io=true` disables data shipping, which leads to reduced overhead for large block write operations. This setting is applied in the Summit templates found in `etc/picongpu/summit-ornl`.

10.10.6 Configuring the openPMD plugin with a TOML configuration file

The openPMD plugin can alternatively be configured by using a `.toml` configuration file. Note the inline comments for a description of the used schema:

```
# The following parameters need not be specified
# If a parameter is left unspecified, it falls back to its default value
file = "simData"   # replaces --openPMD.file,
                   # given value is the default
infix = ""         # replaces --openPMD.infix,
                   # default is "%06T"
ext = "bp4"        # replaces --openPMD.ext,
                   # the default is "bp4" if ADIOS2 is available, else ".h5"
backend_config = "@./adios_config.json"    # replaces --openPMD.json,
                                           # default is "{}"
data_preparation_strategy = "mappedMemory" # replaces --openPMD.
↪dataPreparationStrategy,
                                           # default is "doubleBuffer"
range = "10,:,:"   # replaces --openPMD.range
                   # default is ":,:,:"


# Periods and data sources are specified independently per reading application
# The application names can be arbitrary and are not interpreted, except
# potentially for logging and other messages.
[sink.saxs_scattering.period]
# Each entry here denotes a periodicity combined with data sources requested
# by the reading code from PIConGPU at the specified periodicity
500 = "species_all"

# A second data sink needs other output data
# All reading requests are merged into one single instance of the openPMD plugin
# Overlapping requests are no problem
[sink.some_other_name.period]
# Data sources can be specified as a list if needed
"400:400" = ["E", "B"]
# Time slice syntax is equivalent to that used by --openPMD.period
"100:300:200,444:444" = "fields_all"
```

The location of the `.toml` file on the filesystem is specified via `--openPMD.toml`. If using this parameter, no other parameters must be specified. If another parameter is specified, the openPMD plugin will notice and abort.

10.10.7 Memory Complexity

no extra allocations.

As soon as the openPMD plugin is compiled in, one extra `mallocMC` heap for the particle buffer is permanently reserved. During I/O, particle attributes are allocated one after another. Using `--openPMD.dataPreparationStrategy doubleBuffer` (default) will require at least 2x the GPU memory on the host side. For a smaller host side memory footprint (<< GPU main memory) pick `--openPMD.dataPreparationStrategy mappedMemory`.

10.10.8 Additional Tools

See our *openPMD* chapter.

10.10.9 Experimental: Asynchronous writing

This implements (part of) the workflow described in section 2 of this paper. Rather than writing data to disk directly from PIConGPU, data is streamed via ADIOS2 SST (sustainable streaming transport) to a separate process running asynchronously to PIConGPU. This separate process (`openpmd-pipe`) captures the stream and writes it to disk. `openpmd-pipe` is a Python-based script that comes with recent development versions of openPMD-api (commit bf5174da20e2aeb60ed4c8575da70809d07835ed or newer). A template is provided under `etc/picongpu/summit-ornl/gpu_batch_pipe.tpl` for running such a workflow on the Summit supercompute system. A corresponding single-node runtime configuration is provided for the KelvinHelmholtz example under `share/picongpu/examples/KelvinHelmholtz/etc/picongpu/6_pipe.cfg` (can be scaled up to multi-node). It puts six instances of PIConGPU on one node (one per GPU) and one instance of `openpmd-pipe`.

Advantages:

- Checkpointing and heavy-weight output writing can happen asynchronously, blocking the simulation less.
- Only one file per node is written, implicit node-level aggregation of data from multiple instances of PIConGPU to one instance of `openpmd-pipe` per node. ADIOS2 otherwise also performs explicit node-level data aggregation via MPI; with this setup, ADIOS2 can (and does) skip that step.
- This setup can serve as orientation for the configuration of other loosely-coupled workflows.

Drawbacks:

- Moving all data to another process means that enough memory must be available per node to tolerate that. One way to interpret such a setup is to use idle host memory as a buffer for asynchronous writing in the background.
- Streaming data distribution strategies are not yet mainlined in openPMD-api, meaning that `openpmd-pipe` currently implements a simple ad-hoc data distribution: Data is distributed by simply dividing each dataset into n equal-sized chunks where n is the number of reading processes. In consequence, communication is currently not strictly intra-node. ADIOS2 SST currently relies solely on inter-node communication infrastructure anyway, and performance differences are hence expected to be low.

Notes on the implementation of a proper template file:

- An asynchronous job can be launched by using ordinary Bash syntax for asynchronous launching of processes (`this_is_launched_asynchronously & and_this_is_not;`). Jobs must be waited upon using `wait`.
- Most batch systems will forward all resource allocations of a batch script to launched parallel processes inside the batch script. When launching several processes asynchronously, resources must be allocated explicitly. This includes GPUs, CPU cores and often memory.
- This setup is currently impossible to implement on the HZDR Hemera cluster due to a wrong configuration of the Batch system.

10.11 Particle Calorimeter

A binned calorimeter of the amount of kinetic energy per solid angle and energy-per-particle.

The solid angle bin is solely determined by the particle's momentum vector and not by its position, so we are emulating a calorimeter at infinite distance.

The calorimeter takes into account all existing particles as well as optionally all particles which have already left the global simulation volume.

10.11.1 External Dependencies

The plugin is available as soon as the *openPMD API* is compiled in.

10.11.2 .param file

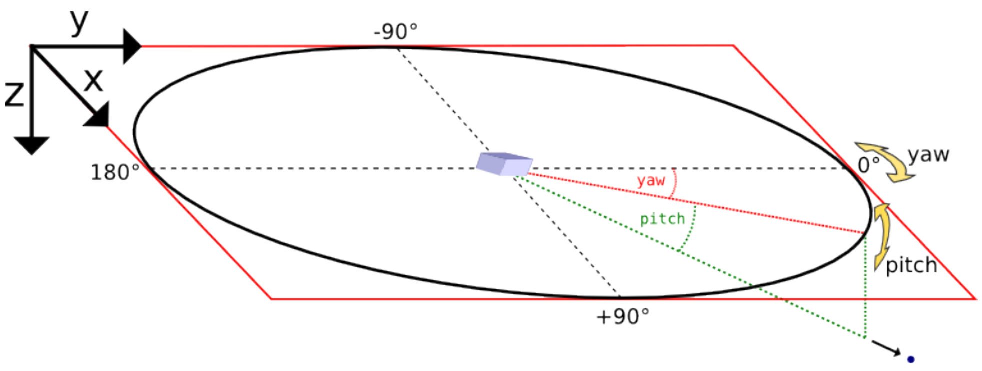
The spatial calorimeter resolution can be customized and in *speciesDefinition.param*. Therein, a species can be also be marked for detecting particles leaving the simulation box.

10.11.3 .cfg file

All options are denoted exemplarily for the photon (`ph`) particle species here.

PIConGPU command line option	Description
`--ph_calorimeter.period`	The ouput periodicity of the plugin. A value of `100` would mean an output at simulation time step *0, 100, 200, . . .*.
`--ph_calorimeter.file`	Output file suffix. Put unique name if same `species + filter` is used multiple times.
`--ph_calorimeter.ext`	openPMD filename extension. This determines the backend to be used by openPMD. Default is `.h5` for HDF5 output.
`--ph_calorimeter.filter`	Use filtered particles. All available filters will be shown with `picongpu --help`
`--ph_calorimeter.numBinsYaw`	Specifies the number of bins used for the yaw axis of the calorimeter. Defaults to `64`.
`--ph_calorimeter.numBinsPitch`	Specifies the number of bins used for the pitch axis of the calorimeter. Defaults to `64`.
`--ph_calorimeter.numBinsEnergy`	Specifies the number of bins used for the energy axis of the calorimeter. Defaults to `1`, i.e. there is no energy binning.
`--ph_calorimeter.minEnergy`	Minimum detectable energy in keV. Ignored if `numBinsEnergy` is 1. Defaults to `0`.
`--ph_calorimeter.maxEnergy`	Maximum detectable energy in keV. Ignored if `numBinsEnergy` is 1. Defaults to `1000`.
`--ph_calorimeter.logScale`	En-/Disable logarithmic energy binning. Allowed values: `0` for disable, `1` enable.
`--ph_calorimeter.openingYaw`	opening angle yaw of the calorimeter in degrees. Defaults to the maximum value: `360`.
`--ph_calorimeter.openingPitch`	opening angle pitch of the calorimeter in degrees. Defaults to the maximum value: `180`.
`--ph_calorimeter.posYaw`	yaw coordinate of the calorimeter position in degrees. Defaults to the +y direction: `0`.
`--ph_calorimeter.posPitch`	pitch coordinate of the calorimeter position in degrees. Defaults to the +y direction: `0`.

10.11.4 Coordinate System



Yaw and pitch are Euler angles defining a point on a sphere's surface, where `(0,0)` points to the `+y` direction here. In the vicinity of `(0,0)`, yaw points to `+x` and pitch to `+z`.

Orientation detail: Since the calorimeters' three-dimensional orientation is given by just two parameters (`posYaw` and `posPitch`) there is one degree of freedom left which has to be fixed. Here, this is achieved by eliminating the Euler angle roll. However, when `posPitch` is exactly `+90` or `-90` degrees, the choice of roll is ambiguous, depending on the yaw angle one approaches the singularity. Here we assume an approach from `yaw = 0`.

10.11.5 Tuning the spatial resolution

By default, the spatial bin size is chosen by dividing the opening angle by the number of bins for yaw and pitch respectively. The bin size can be tuned by customizing the mapping function in `particleCalorimeter.param`.

10.11.6 Memory Complexity

Accelerator

each energy bin times each coordinate bin allocates two counter (`float_X`) permanently and on each accelerator for active and outgoing particles.

Host

as on accelerator.

10.11.7 Output

The calorimeters are stored in openPMD-files in the `simOutput/<species>_calorimeter/` directory. The file names are `<species>_calorimeter_<sfilter>_<timestep>.<file_ending>`.

Depending on whether energy binning is enabled the dataset is two or three dimensional. The dataset has the following attributes:

Attribute	Description
`unitSI`	scaling factor for energy in calorimeter bins
`maxYaw[deg]`	half of the opening angle yaw.
`maxPitch[deg]`	half of the opening angle pitch.
`posYaw[deg]`	yaw coordinate of the calorimeter.
`posPitch[deg]`	pitch coordinate of the calorimeter. If energy binning is enabled:
`minEnergy[keV]`	minimal detectable energy.
`maxEnergy[keV]`	maximal detectable energy.
`logScale`	boolean indicating logarithmic scale.

The output in each bin is given in Joule. Divide by energy value of the bin for a unitless count per bin.

The output uses a custom geometry. Since the openPMD API does currently not (yet) support reading from datasets with a custom-name geometry, this plugin leaves the default geometry "cartesian" instead of specifying something like "calorimeter". If the output is 2D, cells are defined by *[pitch, yaw]* in degrees. If the output is 3D, cells are defined by *[energy bin, pitch, yaw]* where the energy bin is given in keV. Additionally, if *logScale==1*, then the energy bins are on a logarithmic scale whose start and end can be read from the custom attributes *minEnergy[keV]* and *maxEnergy[keV]* respectively.

Note: This plugin is a multi plugin. Command line parameters can be used multiple times to create e.g. dumps with different dumping period. In the case where an optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. e.g.

```
--ph_calorimeter.period 128 --ph_calorimeter.file calo1 --ph_calorimeter.filter all
--ph_calorimeter.period 1000 --ph_calorimeter.file calo2 --ph_calorimeter.filter all -
↪-ph_calorimeter.logScale 1 --ph_calorimeter.minEnergy 1
```

creates two plugins:

1. calorimeter for species ph each 128th time step **with** logarithmic energy binning.
2. calorimeter for species ph each 1000th time step **without** (this is the default) logarithmic energy binning.

Attention: When using the plugin multiple times for the same combination of `species` and `filter`, you *must* provide a unique `file` suffix. Otherwise output files will overwrite each other, since only `species`, `filter` and `file` suffix are encoded in it.

An example use case would be two (or more) calorimeters for the same species and filter but with differing position in space or different binning, range, linear and log scaling, etc.

10.11.8 Analysis Tools

The first bin of the energy axis of the calorimeter contains all particle energy less than the minimal detectable energy whereas the last bin contains all particle energy greater than the maximal detectable energy. The inner bins map to the actual energy range of the calorimeter.

To easily access the data, you can use our python module located in `lib/python/picongpu/extra/plugins/data/calorimeter.py`

```python
import numpy as np
import matplotlib.pyplot as plt

from calorimeter import particleCalorimeter

# setup access to data
calObj = particleCalorimeter("./simOutput/e_calorimeter/e_calorimeter_all_%T.bp")

# last bin contains overflow
selected_energy_bin = -1

plt.title("selected energy: >{:.1f} keV".format(calObj.getEnergy()[selected_energy_
↪bin]), fontsize=18)

plt.pcolormesh(calObj.getYaw(), calObj.getPitch(), calObj.getData(2000)[selected_
↪energy_bin, :, :])
```

(continues on next page)

(continued from previous page)

```
plt.xlabel(calObj.detector_params["axisLabels"][-1] + r" $[^\circ]$", fontsize=18)
plt.ylabel(calObj.detector_params["axisLabels"][-2] + r" $[^\circ]$", fontsize=18)

cb = plt.colorbar()
cb.set_label("energy [keV]", fontsize=18)

plt.show()
```

10.12 Phase Space

This plugin creates a 2D phase space image for a user-given spatial and momentum coordinate.

10.12.1 External Dependencies

The plugin is available as soon as the *openPMD API* is compiled in.

10.12.2 .cfg file

Example for *y-pz* phase space for the *electron* species (`.cfg` file macro):

```
# Calculate a 2D phase space
# - momentum range in m_e c
TGB_ePSypz="--e_phaseSpace.period 10 --e_phaseSpace.filter all --e_phaseSpace.space y␣
↪--e_phaseSpace.momentum pz --e_phaseSpace.min -1.0 --e_phaseSpace.max 1.0 --e_
↪phaseSpace.ext h5"
```

The distinct options are (assuming a species `e` for electrons):

Option	Usage	Unit
`--e_phaseSpace.period <N>`	calculate each N steps	*none*
`--e_phaseSpace.filter`	Use filtered particles. Available filters are set up in *particle-Filters.param*.	*none*
`--e_phaseSpace.space <x/y/z>`	spatial coordinate of the 2D phase space	*none*
`--e_phaseSpace.momentum <px/py/pz>`	momentum coordinate of the 2D phase space	*none*
`--e_phaseSpace.min <ValL>`	minimum of the momentum range	$m_{\text{species}}c$
`--e_phaseSpace.max <ValR>`	maximum of the momentum range	$m_{\text{species}}c$
`--e_phaseSpace.ext <ext>`	filename extension for openPMD backend	*none*

10.12.3 Memory Complexity

Accelerator

locally, a counter matrix of the size local-cells of `space` direction times `1024` (for momentum bins) is permanently allocated.

Host

negligible.

10.12.4 Output

The 2D histograms are stored in the `simOutput/phaseSpace/` directory, by default in `.h5` files. A file is created per species, phasespace selection and time step.

Values are given as *charge density* per phase space bin. In order to scale to a simpler *charge of particles* per $\mathrm{d}r_i$ and $\mathrm{d}p_i$ -bin multiply by the cell volume `dV` (written as an attribute of the openPMD Mesh).

The output writes a number of non-standard custom openPMD attributes:

- `p_min` and `p_max`: The lower and upper bounds for the momentum axis, respectively.
- `dr`: The spacing of the spatial axis in PIConGPU units.
- `dV`: The volume of a cell of the spatial grid. That is, `dV = cell_depth * cell_height * cell_width` in PIConGPU units.
- `dr_unit`: The SI scaling for the spatial axis. Use this instead of `gridUnitSI`.
- `p_unit`: The SI scaling for the momentum axis. Use this instead of `gridUnitSI`.
- `globalDomainOffset`, `globalDomainSize` and `globalDomainAxisLabels`: Information on the global domain.
- `totalDomainOffset`, `totalDomainSize` and `totalDomainAxisLabels`: Information on the total domain. Please consult the PIConGPU wiki for explanations on the meaning of global and total domain.
- `sim_unit`: SI scaling for the charge density values. Alias for `unitSI`.

10.12.5 Analysis Tools

Data Reader

You can quickly load and interact with the data in Python with:

```python
from picongpu.extra.plugins.data import PhaseSpaceData
import numpy as np


ps_data = PhaseSpaceData('/home/axel/runs/lwfa_001')
# show available iterations
ps_data.get_iterations(ps="xpx", species="e", species_filter='all')

# show available simulation times
ps_data.get_times(ps="xpx", species="e", species_filter='all')

# load data for a given iteration
ps, meta = ps_data.get(ps='ypy', species='e', species_filter='all', iteration=2000)

# unit conversion from SI
mu = 1.e6  # meters to microns
e_mc_r = 1. / (9.109e-31 * 2.9979e8)  # electrons: kg * m / s to beta * gamma

Q_dr_dp = np.abs(ps) * meta.dV  # unit: Coulomb
extent = meta.extent * [mu, mu, e_mc_r, e_mc_r]  # spatial: microns, momentum:␣
↪beta*gamma
```

(continues on next page)

(continued from previous page)

```
# load data for a given time
ps, ps_meta = ps_data.get(ps="xpx", species="e", species_filter='all', time=1.3900e-
↪14)

# load data for multiple iterations
ret = ps_data.get(ps="xpx", species="e", species_filter='all', iteration=[2000, 4000])

# data and metadata for iteration 2000
# (data is in same order as the value passed to the 'iteration' parameter)
ps, meta = ret[0]
```

Note that the spatial extent of the output over time might change when running a moving window simulation.

Matplotlib Visualizer

You can quickly plot the data in Python with:

```
from picongpu.extra.plugins.plot_mpl import PhaseSpaceMPL
import matplotlib.pyplot as plt


# create a figure and axes
fig, ax = plt.subplots(1, 1)

# create the visualizer
ps_vis = PhaseSpaceMPL('path/to/run_dir', ax)

# plot
ps_vis.visualize(ps="xpx", iteration=200, species='e', species_filter='all')

plt.show()

# specifying simulation time is also possible (granted there is a matching iteration␣
↪for that time)
ps_vis.visualize(ps="xpx", time=2.6410e-13, species='e', species_filter='all')

plt.show()

# plotting data for multiple simulations simultaneously also works:
ps_vis = PhaseSpaceMPL([
     ("sim1", "path/to/sim1"),
     ("sim2", "path/to/sim2"),
     ("sim3", "path/to/sim3")], ax)
ps_vis.visualize(ps="xpx", iteration=10000, species="e", species_filter='all')

plt.show()
```

The visualizer can also be used from the command line (for a single simulation only) by writing

```
python phase_space_visualizer.py
```

with the following command line options

Options	Value
-p	Path and filename to the run directory of a simulation.
-i	An iteration number
-s (optional, defaults to 'e')	Particle species abbreviation (e.g. 'e' for electrons)
-f (optional, defaults to 'all')	Species filter string
-m (optional, defaults to 'ypy')	Momentum string to specify the phase space

Jupyter Widget

If you want more interactive visualization, then start a jupyter notebook and make sure that `ipywidgets` and `ipympl` are installed.

After starting the notebook server write the following

```
# this is required!
%matplotlib widget
import matplotlib.pyplot as plt
plt.ioff()

from IPython.display import display
from picongpu.extra.plugins.jupyter_widgets import PhaseSpaceWidget

# provide the paths to the simulations you want to be able to choose from
# together with labels that will be used in the plot legends so you still know
# which data belongs to which simulation
w = PhaseSpaceWidget(run_dir_options=[
        ("scan1/sim4", "/path/to/scan1/sim4"),
        ("scan1/sim5", "/path/to/scan1/sim5")])
display(w)
```

and then interact with the displayed widgets.

Plase note that per default the widget allows selection only of the `ypy` phase space slice for particles labelled by `e`. To visualize, for instance the `ypy`, `xpx` and `ypz` slices for particles labelled by `e` (as a rule background electrons) and by `b` (here electrons of a particle bunch) the above has to be augmented by setting `w.ps.options` and `w.species.options`. The final script snippet then reads:

```
# this is required!
%matplotlib widget
import matplotlib.pyplot as plt
plt.ioff()

from IPython.display import display
from picongpu.plugins.jupyter_widgets import PhaseSpaceWidget

# provide the paths to the simulations you want to be able to choose from
# together with labels that will be used in the plot legends so you still know
# which data belongs to which simulation
w = PhaseSpaceWidget(run_dir_options=[
        ("scan1/sim4", "/path/to/scan1/sim4"),
        ("scan1/sim5", "/path/to/scan1/sim5")])
w.ps.set_trait('options', ('ypy', 'xpx', 'ypz'))
w.species.set_trait('options', ('e', 'b'))
display(w)
```

10.12.6 Out-of-Range Behavior

Particles that are *not* in the range of `<ValL>`/`<ValR>` get automatically mapped to the lowest/highest bin respectively. Take care about that when setting your range and during analysis of the results.

10.12.7 Known Limitations

- only one range per selected space-momentum-pair possible right now (naming collisions)
- charge deposition uses the counter shape for now (would need one more write to neighbors to evaluate it correctly according to the shape)
- the user has to define the momentum range in advance
- the resolution is fixed to `1024 bins` in momentum and the number of cells in the selected spatial dimension
- While the openPMD standard has already been updated to support phase space data, the openPMD API does not yet implement this part. The openPMD attribute `gridUnitSI` and `gridUnitDimension` can hence not be correctly written yet and should be ignored in favor of the custom attributes written by this plugin.

10.12.8 References

The internal algorithm is explained in pull request #347 and in [Huebl2014].

10.13 PNG

This plugin generates **images in the png format** for slices through the simulated volume. It allows to draw a **species density** together with electric, magnetic and/or current field values. The exact field values, their coloring and their normalization can be set using `*.param` files. It is a very rudimentary and useful tool to get a first impression on what happens in the simulation and to verify that the parameter set chosen leads to the desired physics.

Note: In the near future, this plugin might be replaced by the ISAAC interactive 3D visualization.

10.13.1 External Dependencies

The plugin is available as soon as the *PNGwriter library* is compiled in.

10.13.2 .cfg file

For **electrons** (`e`) the following table describes the command line arguments used for the visualization.

Command line option	Description
`--e_png.period`	This flag requires an integer value that specifies at what periodicity the png pictures should be created. E.g. setting `--e_png.period 100` generates images for the *0th, 100th, 200th, …* time step. There is no default. If flags are not set, no pngs are created.
`--e_png.axis`	Set 2D slice through 3D volume that will be drawn. Combine two of the three dimensions `x`, `y``and``z`, the define a slice. E.g. setting `--e_png.axis yz` draws both the `y` and `z` dimension and performs a slice in x-direction.
`--e_png.slicePoi`	Specifies at what ratio of the total depth of the remaining dimension, the slice should be performed. The value given should lie between `0.0` and `1.0`.
`--e_png.folder`	Name of the folder, where all pngs for the above setup should be stored.

These flags use `boost::program_options`'s `multitoken()`. Therefore, **several setups** can be specified e.g. to draw different slices. The order of the flags is important in this case. E.g. in the following example, two different slices are visualized and stored in different directories:

```
picongpu [more args]
  # first
  --e_png.period 100
  --e_png.axis xy
  --e_png.slicePoint 0.5
  --e_png.folder pngElectronsXY
  # second
  --e_png.period 100
  --e_png.axis xz
  --e_png.slicePoint 0.5
  --e_png.folder pngElectronsXZ
```

10.13.3 .param files

The two param files *png.param* and *pngColorScales.param* are used to specify the desired output.

Specifying the field values using `png.param`

Depending on the used prefix in the command line flags, electron and/or ion density is drawn. Additionally to that, three field values can be visualized together with the particle density. In order to set up the visualized field values, the `png.param` needs to be changed. In this file, a variety of other parameters used for the PngModule can be specified.

The ratio of the image can be set.

```
/* scale image before write to file, only scale if value is not 1.0 */
const double scale_image = 1.0;

/* if true image is scaled if cellsize is not quadratic, else no scale */
const bool scale_to_cellsize = true;
```

In order to scale the image, `scale_to_cellsize` needs to be set to `true` and `scale_image` needs to specify the reduction ratio of the image.

Note: For a 2D simulation, even a 2D image can be a quite heavy output. Make sure to reduce the preview size!

It is possible to draw the borders between the GPUs used as white lines. This can be done by setting the parameter `white_box_per_GPU` in `png.param` to `true`

```
const bool white_box_per_GPU = true;
```

There are three field values that can be drawn: `CHANNEL1`, `CHANNEL2` and `CHANNEL3`.

Since an adequate color scaling is essential, there several option the user can choose from.

```
// normalize EM fields to typical laser or plasma quantities
//-1: Auto: enable adaptive scaling for each output
// 1: Laser: [outdated]
// 2: Drift: [outdated]
// 3: PlWave: typical fields calculated out of the plasma freq.,
// assuming the wave moves approx. with c
// 4: Thermal: [outdated]
// 5: BlowOut: [outdated]
// 6: Custom: user-provided normalization factors via customNormalizationSI
// 7: Incident: typical fields calculated out of the incident field amplitude,
// uses max amplitude from all enabled incident field profile types ignoring Free
#define EM_FIELD_SCALE_CHANNEL1 -1
#define EM_FIELD_SCALE_CHANNEL2 -1
#define EM_FIELD_SCALE_CHANNEL3 -1

/** SI values to be used for Custom normalization
 *
 * The order of normalization values is: B, E, current (note - current, not current␣
↪density).
 * This variable must always be defined, but has no effect for other normalization␣
↪types.
 */
constexpr float_64 customNormalizationSI[3] = {5.0e12 / SI::SPEED_OF_LIGHT_SI, 5.0e12,
↪ 15.0};
```

In the above example, all channels are set to **auto scale**. **Be careful**, when using a normalization other than auto-scale, depending on your setup, the normalization might fail due to parameters not set by PIConGPU. *Use the other normalization options only in case of the specified scenarios or if you know, how the scaling is computed.*

You can also add opacity to the particle density and the three field values:

```
// multiply highest undisturbed particle density with factor
float_X const preParticleDens_opacity = 0.25;
float_X const preChannel1_opacity = 1.0;
float_X const preChannel2_opacity = 1.0;
float_X const preChannel3_opacity = 1.0;
```

and add different coloring:

```
// specify color scales for each channel
namespace preParticleDensCol = colorScales::red;  /* draw density in red */
namespace preChannel1Col = colorScales::blue;     /* draw channel 1 in blue */
namespace preChannel2Col = colorScales::green;    /* draw channel 2 in green */
namespace preChannel3Col = colorScales::none;     /* do not draw channel 3 */
```

The colors available are defined in `pngColorScales.param` and their usage is described below. If `colorScales::none` is used, the channel is not drawn.

In order to specify what the three channels represent, three functions can be defined in `png.param`. The define the values computed for the png visualization. The data structures used are those available in PIConGPU.

```
/** Calculate values for png channels for given field values
 *
 * @param field_B normalized magnetic field value
 * @param field_E normalized electric field value
 * @param field_Current normalized electric current value (note - not current density)
 *
 * @{
 */
DINLINE float_X preChannel1( float3_X const & field_B, float3_X const & field_E,␣
↪float3_X const & field_Current )
{
    /* Channel1
     * computes the absolute value squared of the electric current */
    return pmacc::math::l2norm2(field_Current);
}

DINLINE float_X preChannel2( float3_X const & field_B, float3_X const & field_E,␣
↪float3_X const & field_Current )
{
    /* Channel2
     * computes the square of the x-component of the electric field */
    return field_E.x() * field_E.x();
}

DINLINE float_X preChannel3( float3_X const & field_B, float3_X const & field_E,␣
↪float3_X const & field_Current )
{
    /* Channel3
     * computes the negative values of the y-component of the electric field
     * positive field_E.y() return as negative values and are NOT drawn */
    return -float_X(1.0) * field_E.y();
}

/** @} */
```

Only positive values are drawn. Negative values are clipped to zero. In the above example, this feature is used for `preChannel3`.

Defining coloring schemes in `pngColorScales.param`

There are several predefined color schemes available:

- none (do not draw anything)
- gray
- grayInv
- red
- green
- blue

But the user can also specify his or her own color scheme by defining a namespace with the color name that provides an `addRGB` function:

```
namespace NameOfColor /* name needs to be unique */
{
    HDINLINE void addRGB( float3_X& img, /* the already existing image */
                          const float_X value, /* the value to draw */
```

(continues on next page)

(continued from previous page)

```
                              const float_X opacity ) /* the opacity specified */
    {
        /* myChannel specifies the color in RGB values (RedGreenBlue) with
         * each value ranging from 0.0 to 1.0 .
         * In this example, the color yellow (RGB=1,1,0) is used. */
        const float3_X myChannel( 1.0, 1.0, 0.0 );

        /* here, the previously calculated image (in case, other channels have already
         * contributed to the png) is changed.
         * First of all, the total image intensity is reduced by the opacity of this
         * channel, but only in the color channels specified by this color
↪"NameOfColor".
         * Then, the actual values are added with the correct color (myChannel) and␣
↪opacity. */
        img = img
              - opacity * float3_X( myChannel.x() * img.x(),
                                    myChannel.y() * img.y(),
                                    myChannel.z() * img.z() )
              + myChannel * value * opacity;
    }
}
```

For most cases, using the predefined colors should be enough.

10.13.4 Memory Complexity

Accelerator

locally, memory for the local 2D slice is allocated with 3 channels in `float_X`.

Host

as on accelerator. Additionally, the master rank has to allocate three channels for the full-resolution image. This is the original size **before** reduction via `scale_image`.

10.13.5 Output

The output of this plugin are pngs stored in the directories specified by `--e_png.folder` or `--i_png.folder`. There can be as many of these folders as the user wants. The pngs follow a naming convention:

```
<species>_png_yx_0.5_002000.png
```

First, either `<species>` names the particle type. Following the 2nd underscore, the drawn dimensions are given. Then the slice ratio, specified by `--e_png.slicePoint` or `--i_png.slicePoint`, is stated in the file name. The last part of the file name is a 6 digit number, specifying the simulation time step, at which the picture was created. This naming convention allows to put all pngs in one directory and still be able to identify them correctly if necessary.

10.13.6 Analysis Tools

Data Reader

You can quickly load and interact with the data in Python with:

```
from picongpu.plugins.data import PNGData


png_data = PNGData('path/to/run_dir')

# get the available iterations for which output exists
iters = png_data.get_iterations(species="e", axis="yx")

# get the available simulation times for which output exists
times = png_data.get_times(species="e", axis="yx")

# pngs as numpy arrays for multiple iterations (times would also work)
pngs = png_data.get(species="e", axis="yx", iteration=iters[:3])

for png in pngs:
    print(png.shape)
```

Matplotlib Visualizer

If you are only interested in visualizing the generated png files it is even easier since you don't have to load the data manually.

```
from picongpu.plugins.plot_mpl import PNGMPL
import matplotlib.pyplot as plt


# create a figure and axes
fig, ax = plt.subplots(1, 1)

# create the visualizer
png_vis = PNGMPL('path/to/run_dir', ax)

# plot
png_vis.visualize(iteration=200, species='e', axis='yx')

plt.show()
```

The visualizer can also be used from the command line by writing

```
python png_visualizer.py
```

with the following command line options

Options	Value
-p	Path and to the run directory of a simulation.
-i	An iteration number
-s	Particle species abbreviation (e.g. 'e' for electrons)
-f (optional, defaults to 'e')	Species filter string
-a (optional, defaults to 'yx')	Axis string (e.g. 'yx' or 'xy')
-o (optional, defaults to 'None')	A float between 0 and 1 for slice offset along the third dimension

Jupyter Widget

If you want more interactive visualization, then start a jupyter notebook and make sure that `ipywidgets` and `ipympl` are installed.

After starting the notebook server write the following

```
# this is required!
%matplotlib widget
import matplotlib.pyplot as plt
# deactivate interactive mode
plt.ioff()

from IPython.display import display
from picongpu.plugins.jupyter_widgets import PNGWidget

# provide the paths to the simulations you want to be able to choose from
# together with labels that will be used in the plot legends so you still know
# which data belongs to which simulation
w = PNGWidget(run_dir_options=[
        ("scan1/sim4", scan1_sim4),
        ("scan1/sim5", scan1_sim5)])
display(w)
```

and then interact with the displayed widgets.

10.14 Radiation

The spectrally resolved far field radiation of charged macro particles.

Our simulation computes the Lienard Wiechert potentials to calculate the emitted electromagnetic spectra for different observation directions using the far field approximation.

$$\frac{\mathrm{d}^2 I}{\mathrm{d}\Omega\,\mathrm{d}\omega}(\omega,\vec{n}) = \frac{q^2}{16\pi^3\varepsilon_0 c}\left|\sum_{k=1}^{N}\int_{-\infty}^{+\infty}\frac{\vec{n}\times\left[\left(\vec{n}-\vec{\beta}_k(t)\right)\times\dot{\vec{\beta}}_k(t)\right]}{\left(1-\vec{\beta}_k(t)\cdot\vec{n}\right)^2}\cdot \mathrm{e}^{\mathrm{i}\,\omega(t-\vec{n}\cdot\vec{r}_k(t)/c)}\,\mathrm{d}\,t\right|^2$$

Details on how radiation is computed with this plugin and how the plugin works can be found in [Pausch2012]. A list of tests can be found in [Pausch2014] and [Pausch2019].

Variable	Meaning
$\vec{r}_k(t)$	The position of particle k at time t.
$\vec{\beta}_k(t)$	The normalized speed of particle k at time t. (Speed divided by the speed of light)
$\dot{\vec{\beta}}_k(t)$	The normalized acceleration of particle k at time t. (Time derivative of the normalized speed.)
t	Time
$\vec{n}$	Unit vector pointing in the direction where the far field radiation is observed.
ω	The circular frequency of the radiation that is observed.
N	Number of all (macro) particles that are used for computing the radiation.
k	Running index of the particles.

Currently this allows to predict the emitted radiation from plasma if it can be described by classical means. Not considered are emissions from ionization, Compton scattering or any bremsstrahlung that originate from scattering on scales smaller than the PIC cell size.

10.14.1 External Dependencies

The plugin is available as soon as the *openPMD API* is compiled in.

10.14.2 .param files

In order to setup the radiation analyzer plugin, both the *radiation.param* and the *radiationObserver.param* have to be configured **and** the radiating particles need to have the attribute `momentumPrev1` which can be added in *speciesDefinition.param*.

In *radiation.param*, the number of frequencies `N_omega` and observation directions `N_theta` is defined.

Frequency range

The frequency range is set up by choosing a specific namespace that defines the frequency setup

```
/* choose linear frequency range */
namespace radiation_frequencies = linear_frequencies;
```

Currently you can choose from the following setups for the frequency range:

namespace	Description
`linear_frequencies`	linear frequency range from `SI::omega_min` to `SI::omega_max` with `N_omega` steps
`log_frequencies`	logarithmic frequency range from `SI::omega_min` to `SI::omega_max` with `N_omega` steps
`frequencies_from_lis`	`N_omega` frequencies taken from a text file with location `listLocation[]`

All three options require variable definitions in the according namespaces as described below:

For the **linear frequency** scale all definitions need to be in the `picongpu::plugins::radiation::linear_frequencies` namespace. The number of total sample frequencies `N_omega` need to be defined as `constexpr unsigned int`. In the sub-namespace `SI`, a minimal frequency `omega_min` and a maximum frequency `omega_max` need to be defined as `constexpr float_64`.

For the **logarithmic frequency** scale all definitions need to be in the `picongpu::plugins::radiation::log_frequencies` namespace. Equivalently to the linear case, three variables need to be defined: The number of total sample frequencies `N_omega` need to be defined as `constexpr unsigned int`. In the sub-namespace `SI`, a minimal frequency `omega_min` and a maximum frequency `omega_max` need to be defined as `constexpr float_64`.

For the **file-based frequency** definition, all definitions need to be in the `picongpu::plugins::radiation::frequencies_from_list` namespace. The number of total frequencies `N_omega` need to be defined as `constexpr unsigned int` and the path to the file containing the frequency values in units of $[\mathrm{s}^{-1}]$ needs to be given as `constexpr const char * listLocation = "/path/to/frequency_list";`. The frequency values in the file can be separated by newlines, spaces, tabs, or any other whitespace. The numbers should be given in such a way, that c++ standard `std::ifstream` can interpret the number e.g., as `2.5344e+16`.

Note: Currently, the variable `listLocation` is required to be defined in the `picongpu::plugins::radiation::frequencies_from_list` namespace, even if `frequencies_from_list` is not used. The string does not need to point to an existing file, as long as the file-based frequency definition is not used.

Observation directions

The number of observation directions N_theta is defined in *radiation.param*, but the distribution of observation directions is given in *radiationObserver.param*) There, the function observationDirection defines the observation directions.

This function returns the x,y and z component of a **unit vector** pointing in the observation direction.

```
DINLINE vector_64
observationDirection( int const observation_id_extern )
{
    /* use the scalar index const int observation_id_extern to compute an
     * observation direction (x,y,y) */
    return vector_64( x , y , z );
}
```

Note: The radiationObserver.param set up will be subject to **further changes**. These might be *namespaces* that describe several preconfigured layouts or a functor if *C++ 11* is included in the *nvcc*.

Nyquist limit

A major limitation of discrete Fourier transform is the limited frequency resolution due to the discrete time steps of the temporal signal. (see Nyquist-Shannon sampling theorem) Due to the consideration of relativistic delays, the sampling of the emitted radiation is not equidistantly sampled. The plugin has the option to ignore any frequency contributions that lies above the frequency resolution given by the Nyquist-Shannon sampling theorem. Because performing this check costs computation time, it can be switched off. This is done via a precompiler pragma:

```
// Nyquist low pass allows only amplitudes for frequencies below Nyquist frequency
// 1 = on (slower and more memory, no Fourier reflections)
// 0 = off (faster but with Fourier reflections)
#define __NYQUISTCHECK__ 0
```

Additionally, the maximally resolvable frequency compared to the Nyquist frequency can be set.

```
namespace radiationNyquist
{
    /* only use frequencies below 1/2*Omega_Nyquist */
    const float NyquistFactor = 0.5;
}
```

This allows to make a save margin to the hard limit of the Nyquist frequency. By using NyquistFactor = 0.5 for periodic boundary conditions, particles that jump from one border to another and back can still be considered.

Form factor

The *form factor* is a method, which considers the shape of the macro particles when computing the radiation. More details can be found in [Pausch2018] and [Pausch2019].

One can select between different macro particle shapes. Currently eight shapes are implemented. A shape can be selected by choosing one of the available namespaces:

```
/* choosing the 3D CIC-like macro particle shape */
namespace radFormFactor = radFormFactor_CIC_3D;
```

Namespace	Description
radFormFactor_CIC_3D	3D Cloud-In-Cell shape
radFormFactor_TSC_3D	3D Triangular shaped density cloud
radFormFactor_PCS_3D	3D Quadratic spline density shape (Piecewise Cubic Spline assignment function)
radFormFactor_CIC_1Dy	Cloud-In-Cell shape in y-direction, dot like in the other directions
radFormFactor_Gauss_sp	symmetric Gauss charge distribution
radFormFactor_Gauss_ce	Gauss charge distribution according to cell size
radFormFactor_incohere	forces a completely incoherent emission by scaling the macro particle charge with the square root of the weighting
radFormFactor_coherent	forces a completely coherent emission by scaling the macro particle charge with the weighting

Reducing the particle sample

In order to save computation time, only a random subset of all macro particles can be used to compute the emitted radiation. In order to do that, the radiating particle species needs the attribute `radiationMask` (which is initialized as `false`) which further needs to be manipulated, to set to true for specific (random) particles.

Note: The reduction of the total intensity is not considered in the output. The intensity will be (in the incoherent case) will be smaller by the fraction of marked to all particles.

Note: The radiation mask is only added to particles, if not all particles should be considered for radiation calculation. Adding the radiation flag costs memory.

Note: In future updates, the radiation will only be computed using an extra particle species. Therefore, this setup will be subject to further changes.

Gamma filter

In order to consider the radiation only of particles with a gamma higher than a specific threshold, the radiating particle species needs the attribute `radiationMask` (which is initialized as `false`). Using a filter functor as:

```
using RadiationParticleFilter = picongpu::particles::manipulators::FreeImpl<
    GammaFilterFunctor
 >;
```

(see Bunch or Kelvin Helmholtz example for details) sets the flag to true is a particle fulfills the gamma condition.

Note: More sophisticated filters might come in the near future. Therefore, this part of the code might be subject to changes.

Window function filter

A window function can be added to the simulation area to reduce ringing artifacts due to sharp transition from radiating regions to non-radiating regions at the boundaries of the simulation box. This should be applied to simulation setups where the entire volume simulated is radiating (e.g. Kelvin-Helmholtz Instability).

In `radiation.param` the precompiler variable `PIC_RADWINDOWFUNCTION` defines if the window function filter should be used or not.

```
// add a window function weighting to the radiation in order
// to avoid ringing effects from sharp boundaries
// 1 = on (slower but with noise/ringing reduction)
// 0 = off (faster but might contain ringing)
#define PIC_RADWINDOWFUNCTION 0
```

If set to 1, the window function filter is used.

There are several different window function available:

```
/* Choose different window function in order to get better ringing reduction
 * radWindowFunctionRectangle
 * radWindowFunctionTriangle
 * radWindowFunctionHamming
 * radWindowFunctionTriplett
 * radWindowFunctionGauss
 */
namespace radWindowFunctionRectangle { }
namespace radWindowFunctionTriangle { }
namespace radWindowFunctionHamming { }
namespace radWindowFunctionTriplett { }
namespace radWindowFunctionGauss { }

namespace radWindowFunction = radWindowFunctionTriangle;
```

By setting `radWindowFunction` a specific window function is selected.

More details can be found in [Pausch2019].

10.14.3 .cfg file

For a specific (charged) species <`species`> e.g. `e`, the radiation can be computed by the following commands.

10.14.4 Memory Complexity

Accelerator

locally, `numJobs` times number of frequencies `N_omega` times number of directions `N_theta` is permanently allocated. Each result element (amplitude) is a double precision complex number.

Host

as on accelerator.

10.14.5 Output

Depending on the command line options used, there are different output files.

Command line flag	Output description
`--<species totalRadia`	Contains *ASCII* files that have the total spectral intensity until the timestep specified by the filename. Each row gives data for one observation direction (same order as specified in the `observer.py`). The values for each frequency are separated by *tabs* and have the same order as specified in `radiation.param`. The spectral intensity is stored in the units [Js].
`--<species lastRadiat`	has the same format as the output of *totalRadiation*. The spectral intensity is only summed over the last radiation `dump` period.
`--<species radPerGPU`	Same output as *totalRadiation* but only summed over each GPU. Because each GPU specifies a spatial region, the origin of radiation signatures can be distinguished.
`--<species distribute`	By default, the amplitudes are summed up over the MPI ranks. Some analyses require the raw distributed output of amplitudes without prior summation. Activating this flag will make the Radiation plugin additionally output the amplitudes before summation. The first dimension in the dataset output corresponds with the parallel distribution of MPI workers. Note that instead of separate datasets for the imaginary and real parts, this output is formatted as datasets of complex number types.
radiationOpenPM	In the folder `radiationOpenPMD`, openPMD files for each radiation dump and species are stored. These are complex amplitudes in units used by *PIConGPU*. These are for restart purposes and for more complex data analysis.

Text-based output

The text-based output of `lastRadiation` and `totalRadiation` contains the intensity values in SI-units [Js]. Intensity values for different frequencies are separated by spaces, while newlines separate values for different observation directions.

In order to read and plot the text-based radiation data, a python script as follows could be used:

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

# frequency definition:
# as defined in the 'radiation.param' file:
N_omega = 1024
omega_min = 0.0 # [1/s]
omega_max = 5.8869e17 # [1/s]
omega = np.linspace(omega_min, omega_max, N_omega)

# observation angle definition:
# as defined in the 'radiation.param' file:
N_observer = 128
# as defined in the 'radiationObserver.param' file:
# this example assumes one used the default Bunch example
# there, the theta values are normalized to the Lorentz factor
theta_min = -1.5 # [rad/gamma]
```

(continues on next page)

(continued from previous page)

```
theta_max = +1.5 # [rad/gamma]
theta = np.linspace(theta_min, theta_max, N_observer)

# load radiation text-based data
rad_data = np.loadtxt('./simOutput/lastRad/e_radiation_2820.dat')

# plot radiation spectrum
plt.figure()
plt.pcolormesh(omega, theta, rad_data, norm=LogNorm())

# add and configure colorbar
cb = plt.colorbar()
cb.set_label(r"$\frac{\mathrm{d}^2 I}{\mathrm{d} \omega \mathrm{d} \Omega} \, \mathrm
↪{[Js]}$", fontsize=18)
for i in cb.ax.get_yticklabels():
    i.set_fontsize(14)

# configure x-axis
plt.xlabel(r"$\omega \, \mathrm{[1/s]}$", fontsize=18)
plt.xticks(fontsize=14)

# configure y-axis
plt.ylabel(r"$\theta / \gamma$", fontsize=18)
plt.yticks(fontsize=14)

# make plot look nice
plt.tight_layout()
plt.show()
```

openPMD output

The openPMD based data contains the following data structure in `/data/{iteration}/DetectorMesh/` according to the openPMD standard:

Amplitude (Group):

Dataset	Description	Dimensions
`x_Re`	real part, x-component of the complex amplitude	(`N_observer`, `N_omega`, 1)
`x_Im`	imaginary part, x-component of the complex amplitude	(`N_observer`, `N_omega`, 1)
`y_Re`	real part, y-component of the complex amplitude	(`N_observer`, `N_omega`, 1)
`y_Im`	imaginary part, y-component of the complex amplitude	(`N_observer`, `N_omega`, 1)
`z_Re`	real part, z-component of the complex amplitude	(`N_observer`, `N_omega`, 1)
`z_Im`	imaginary part, z-component of the complex amplitude	(`N_observer`, `N_omega`, 1)

Note: Please be aware, that despite the fact, that the SI-unit of each amplitude entry is $[\sqrt{\mathrm{Js}}]$, the stored `unitSI` attribute returns $[\mathrm{Js}]$. This inconsistency will be fixed in the future. Until this inconstincy is resolved, please multiply the datasets with the square root of the `unitSI` attribute to convert the amplitudes to SI units.

Amplitude_distributed (Group, opt-in):

Dataset	Description	Dimensions
x	x-component of the complex amplitude	(MPI ranks, N_observer, N_omega)
y	y-component of the complex amplitude	(MPI ranks, N_observer, N_omega)
z	z-component of the complex amplitude	(MPI ranks, N_observer, N_omega)

Note: Please be aware, that despite the fact, that the SI-unit of each amplitude entry is $[\sqrt{\mathrm{Js}}]$, the stored `unitSI` attribute returns $[\mathrm{Js}]$. This inconsistency will be fixed in the future. Until this inconstincy is resolved, please multiply the datasets with the square root of the `unitSI` attribute to convert the amplitudes to SI units.

Additionally, the representation of complex numbers differs from that in the summed Amplitude output. This inconsistency will be fixed by changing the output format of the summed Amplitudes.

DetectorDirection (Group):

Dataset	Description	Dimensions
x	x-component of the observation direction $\vec{n}$	(N_observer, 1, 1)
y	y-component of the observation direction $\vec{n}$	(N_observer, 1, 1)
z	z-component of the observation direction $\vec{n}$	(N_observer, 1, 1)

DetectorFrequency (Group):

Dataset	Description	Dimensions
omega	frequency ω of virtual detector bin	(1, N_omega, 1)

Please be aware that all datasets in the openPMD output are given in the PIConGPU-intrinsic unit system. In order to convert, for example, the frequencies ω to SI-units one has to multiply with the dataset-attribute *unitSI*.

```python
import openpmd_api as io
series = io.Series("e_radAmplitudes_%T.bp", io.Access_Type.read_only)
iteration = series.iterations[2800]

omega_handler = iteration.meshes["DetectorFrequency"]["omega"]
omega = omega_h[0, :, 0]
omega_unitSI = omega_h.unit_SI
series.flush()
omega *= omega_unitSI
```

In order to extract the radiation data from the openPMD datasets, PIConGPU provides a python module to read the data and obtain the result in SI-units. An example python script is given below. This currently assumes adios output but any openPMD output, such as hdf5, will work too.

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

from picongpu.plugins.data import RadiationData

# access openPMD radiation file for specific time step
radData = RadiationData("./simOutput/radiationOpenPMD/e_radAmplitudes_%T.bp", 2820)

# get frequencies
omega = radData.get_omega()
```

(continues on next page)

(continued from previous page)

```python

# get all observation vectors and convert to angle

vec_n = radData.get_vector_n()
gamma = 5.0
theta_norm = np.arctan2(vec_n[:, 0], vec_n[:, 1]) * gamma

# get spectrum over observation angle
spectrum = radData.get_Spectra()

# plot radiation spectrum
plt.figure()
plt.pcolormesh(omega, theta_norm, spectrum, norm=LogNorm())

# add and configure colorbar
cb = plt.colorbar()
cb.set_label(r"$\frac{\mathrm{d}^2 I}{\mathrm{d} \omega \mathrm{d} \Omega} \, \mathrm
→{[Js]}$", fontsize=18)
for i in cb.ax.get_yticklabels():
    i.set_fontsize(14)

# configure x-axis
plt.xlabel(r"$\omega \, \mathrm{[1/s]}$", fontsize=18)
plt.xticks(fontsize=14)

# configure y-axis
plt.ylabel(r"$\theta / \gamma$", fontsize=18)
plt.yticks(fontsize=14)

# make plot look nice
plt.tight_layout()
plt.show()
```

There are various methods besides `get_Spectra()` that are provided by the python module. If a method exists for `_x` (or `_X`) it also exists for `_y` and `_z` (`_Y` and `_Z`) accordingly.

Method	Description
`.get_omega()`	get frequency ω of virtual detector bin in units of $[1/\mathrm{s}]$
`.get_vector_n()`	get observation direction $\vec{n}$
`.get_Spectra()`	get spectrum $\mathrm{d}^2I/\mathrm{d}\omega\mathrm{d}\Omega$ in units of $[\mathrm{Js}]$
`.get_Polarization_X()`	get spectrum but only for polarization in x-direction
`.get_Amplitude_x()`	get x-component of complex amplitude (unit: $[\sqrt{\mathrm{Js}}]$)
`.get_timestep()`	the iteration (timestep) at which the data was produced (unit: PIC-cycles)

Note: Modules for visualizing radiation data and a widget interface to explore the data interactively will be developed in the future.

10.14.6 Analyzing tools

In `picongp/src/tools/bin`, there are tools to analyze the radiation data after the simulation.

Tool	Description
`plotRadiatio py`	Reads *ASCII* radiation data and plots spectra over angles as color plots. This is a python script that has its own help. Run `plotRadiation.py --help` for more information.
`radiationSyn py`	Reads *ASCII* radiation data and statistically analysis the spectra for a user specified region of observation angles and frequencies. This is a python script that has its own help. Run `radiationSyntheticDetector.py --help` for more information.
smooth.py	Python module needed by `plotRadiation.py`.

10.14.7 Known Issues

The plugin supports multiple radiation species but spectra (frequencies and observation directions) are the same for all species.

10.14.8 References

10.15 Shadowgraphy

Computes a 2D image by time-integrating the Poynting-Vectors in a fixed plane in the simulation. This can be used to extract a laser from an simulation, which obtains the full laser-plasma interactions through the PIC code. If the probe laser propagates through plasma structures, the plasma structures lead to modulations in the probe laser's intensity, resulting in a synthetic shadowgram of the plasma structures. The plugin performs the time-integration of the probe laser and the application of various masks on the probe pulse in Fourier space. Thus, one needs to manually add the probe pulse to the simulation with e.g. the *incident field* param files. Since the plugin currently only works in the xy-plane, the probe pulse should propagate in z direction. The integration plane for the plugin must lie within the simulation volume and the position of the field absorber should be considered when placing the plugin plane.

10.15.1 External Dependencies

The plugin is available as soon as the *FFWT3* is compiled in.

10.15.2 Usage

Command line option	Description
`--shadowgrapł start`	Step to start the plugin and with this the time integration.
`--shadowgrapł duration`	Duration of shadowgraphy calculation in simulation time steps. The plugin will be called `duration / params::tRes` times. The final plugin call (and the file output) happens at the time defined in `.period` plus the duration.
`--shadowgrapł slicePoint`	Specifies at what ratio of the total depth of the z dimension, the slice for the field extraction should be set. The value given should lie between `0.0` and `1.0`.
`--shadowgrapł focusPos`	Focus position of lens system relative to slicePoint. The focus position is given in SI units.
`--shadowgrapł fileName`	Output file prefix for openPMD output.
`--shadowgrapł ext`	Backend for openPMD output.
`--shadowgrapł fourierOutpuı`	If enabled, the fields will also be stored on disk in in `(x, y, \omega)` Fourier space in an openPMD file.

Note: Currently the plugin only supports an integration slice in the xy-plane, which means that for probing setups the probe pulse should propagate in z direction. The moving window can't be activated or deactivated during the plugin integration loop.

Reqired param file options	Description
`tRes`	Use the fields at each tRes'th time-step of simulation
`xRes`	Use each xRes'th field value in x direction
`yRes`	Use each yRes'th field value in y direction
`omegaWfMin`	Minimum non-zero value for abs(omega)
`omegaWfMax`	Maximum non-zero value for abs(omega)
`masks::positionWf`	Mask that is multiplied to E and B field when gathering the slice
`masks::timeWf`	Mask that is multiplied to E and B field during the time integration
`masks::maskFourier`	Mask that is multiplied to E and B field in Fourier domain

10.15.3 Output

Plot the first shadowgram that is stored in the simulation output directory `simOutput`.

```python
import matplotlib.pyplot as plt
import numpy as np
import openpmd_api as io


def load_shadowgram(series):
    i = series.iterations[[i for i in series.iterations][0]]

    shadowgram_tmp = i.meshes["shadowgram"][io.Mesh_Record_Component.SCALAR].load_
↪chunk()
    unit = i.meshes["shadowgram"].get_attribute("unitSI")
    series.flush()

```

(continues on next page)

(continued from previous page)

```python
    return shadowgram_tmp * unit


def load_meshgrids(series):
    i = series.iterations[[i for i in series.iterations][0]]

    xspace_tmp = i.meshes["Spatial positions"]["x"].load_chunk()
    xunit = i.meshes["Spatial positions"]["x"].get_attribute("unitSI")
    series.flush()
    xspace = xspace_tmp * xunit

    yspace_tmp = i.meshes["Spatial positions"]["y"].load_chunk()
    yunit = i.meshes["Spatial positions"]["y"].get_attribute("unitSI")
    series.flush()
    yspace = yspace_tmp * yunit

    return np.meshgrid(xspace, yspace)


path = "PATH/TO/simOutput"

series = io.Series(path + "/shadowgraphy_" + "%T." + "bp5", io.Access.read_only)
shadowgram = load_shadowgram(series)
xm, ym = load_meshgrids(series)
series.close()

fig, ax = plt.subplots(figsize=(10, 10))
ax.pcolormesh(xm, ym, shadowgram)
ax.set_aspect("equal")
```

10.15.4 Shadowgram Size and Moving Window

The size of the pixels is the size of the cells in the simulation divided by the resolution in the plugin `CELL_WIDTH_SI * shadowgraphy::params::xRes` and `CELL_HEIGHT_SI * shadowgraphy::params::yRes`. The shadowgram itself does not include cells that lie outside of the field absorber in both x and y direction. When the moving window is activated, the resulting shadowgram is smaller in moving window propagation direction `y`. The size difference is equal to the speed of light times the time it would take for light to propagate from the `-z` border of the simulation box to the plugin integration plane plus the integration duration. This prevents artifacts from the laser being cut off due to the moving window or the laser not fully being propagated through the plasma structures.

10.15.5 References

- ***Modeling ultrafast shadowgraphy in laser-plasma interaction experiments***
  E Siminos et al 2016 Plasma Phys. Control. Fusion 58 065004 https://doi.org/10.1088/0741-3335/58/6/065004
- ***Synthetic few-cycle shadowgraphy diagnostics in particle-in-cell codes for characterizing laser-plasma accelerators***
  Carstens, F.-O., Master Thesis on shadowgraphy plugin https://doi.org/10.5281/zenodo.7755263

10.16 Slice Emittance

The plugin computes the total emittance and the slice emittance (for ten combined cells in the longitudinal direction).

Currently, it outputs only the emittance of the transverse momentum space x-px.

More details on the implementation and tests can be found in the master's thesis [Rudat2019].

10.16.1 External Dependencies

None

10.16.2 .param file

None for now. In the future, adding more compile-time configurations might become necessary (e.g., striding of data output).

10.16.3 .cfg file

All options are denoted for the electron (`e`) particle species here.

PIConGPU command line option	Description
`--e_emittance.period arg`	compute slice emittance [for each n-th step], enable plugin by setting a non-zero value A value of `100` would mean an output at simulation time step *0, 100, 200, ...*
`--e_emittance.filter arg`	Use filtered particles. All available filters will be shown with `picongpu --help`

10.16.4 Memory Complexity

Accelerator

Each `x^2`, `p_x^2` and `x * p_x` summation value as well as the number of real electrons `gCount_e` needs to be stored as `float_64` for each y-cell.

Host

as on accelerator (needed for MPI data transfer)

10.16.5 Output

Note: This plugin is a multi plugin. Command line parameters can be used multiple times to create e.g. dumps with different dumping period. In the case where an optional parameter with a default value is explicitly defined the parameter will be always passed to the instance of the multi plugin where the parameter is not set. e.g.

```
--e_emittance.period 1000 --e_emittance.filter all
--e_emittance.period  100 --e_emittance.filter highEnergy
```

creates two plugins:

1. slice emittance for species e each 1000th time step for **all** particles.

2. slice emittance for species e each 100th time step **only for particles** with high energy (defined by filter).

10.16.6 Analysis Tools

The output is a text file with the first line as a comment describing the content. The first column is the time step. The second column is the total emittance (of all particles defined by the filter). Each following column is the emittance if the slice at ten cells around the position given in the comment line.

```
data = np.loadtxt("<path-to-emittance-file>")
timesteps = data[:, 0]
total_emittance = data[:, 1]
slice_emittance = data[:, 2:]

# time evolution of total emitance
plt.plot(timesteps, total_emittance)
plt.xlabel("time step")
plt.ylabel("emittance")
plt.show()

# plot slice emittance over time and longitudinal (y) position
plt.imshow(slice_emittance)
plt.xlabel("y position [arb.u.]")
plt.ylabel("time [arb.u.]")
cb = plt.colorbar()
cb.set_label("emittance")
plt.show()
```

10.16.7 References

10.17 Sum Currents

This plugin computes the total current integrated/added over the entire volume simulated.

10.17.1 .cfg file

The plugin can be activated by setting a non-zero value with the command line flag `--sumcurr.period`. The value set with `--sumcurr.period` is the periodicity, at which the total current is computed. E.g. `--sumcurr.period 100` computes and prints the total current for time step *0, 100, 200, . . .* .

10.17.2 Memory Complexity

Accelerator

negligible.

Host

negligible.

10.17.3 Output

The result is printed to *standard output*. Therefore, it goes both to `./simOutput/output` as well as to the output file specified by the machine used (usually the `stdout` file in the main directory of the simulation). The output is ASCII-text only. It has the following format:

```
[ANALYSIS] [_rank] [COUNTER] [SumCurrents] [_currentTimeStep] {_current.x _current.y _
↪current.z} Abs:_absCurrent
```

Value	Description	Unit
`_rank`	MPI rank at which prints the particle position	*none*
`_currentTimeStep`	simulation time step = number of PIC cycles	*none*
`_current.x _current.y _current.z`	electric current	Ampere per second
`_absCurrent`	magnitude of current	Ampere per second

In order to extract only the total current information from the output stored in *stdout*, the following command on a bash command line could be used:

```
grep SumCurrents stdout > totalCurrent.dat
```

The plugin data is then stored in `totalCurrent.dat`.

10.17.4 Known Limitations

- this plugin is only available with the CUDA backend

10.17.5 Known Issues

Currently, both `output` and `stdout` are overwritten at restart. All data from the plugin is lost, if these file are not backuped manually.

10.18 Transition Radiation

The spectrally resolved far field radiation created by electrons passing through a metal foil.

Our simulation computes the transition radiation to calculate the emitted electromagnetic spectra for different observation angles.

$$\frac{d^2W}{d\omega d\Omega} = \frac{e^2 N_e}{(4\pi\epsilon_0)\pi^2 c}\left\{\left[\int d^3\vec{p}g(\mathcal{E}_\parallel^2 + \mathcal{E}_\perp^2)\right] + (N_e - 1)\left[\left|\int d^3\vec{p}g\mathcal{E}_\parallel F\right|^2 + \left|\int d^3\vec{p}g\mathcal{E}_\perp F\right|^2\right]\right\}$$

$$\mathcal{E}_\parallel = \frac{u\cos\psi\left[u\sin\psi\cos\phi - (1+u^2)^{1/2}\sin\theta\right]}{\mathcal{N}(\theta, u, \psi, \phi)}$$

$$\mathcal{E}_\perp = \frac{u^2\cos\psi\sin\psi\sin\phi\cos\theta}{\mathcal{N}(\theta, u, \psi, \phi)}$$

$$\mathcal{N}(\theta, u, \psi, \phi) = \left[\left(1+u^2\right)^{1/2} - u\sin\psi\cos\phi\sin\theta\right]^2 - u^2\cos^2\psi\cos^2\theta$$

$$F = \frac{1}{g(\vec{p})} \int d^2\vec{r}_\perp e^{-i\vec{k}_\perp \cdot \vec{r}_\perp} \int dy e^{-iy(\omega - \vec{k}_\perp \cdot \vec{v}_\perp)/v_y} h(\vec{r}, \vec{p})$$

Variable	Meaning
N_e	Amount of real electrons
ψ	Azimuth angle of momentum vector from electrons to y-axis of simulation
θ	Azimuth angle of observation vector
ϕ	Polar angle between momentum vector from electrons and observation vector
ω	The circular frequency of the radiation that is observed.
$h(\vec{r}, \vec{p})$	Normalized phasespace distribution of electrons
$g(\vec{p})$	Normalized momentum distribution of electrons
$g(\vec{p})$	Normalized momentum distribution of electrons
$\vec{k}$	Wavevector of electrons
$\vec{v}$	Velocity vector of electrons
u	Normalized momentum of electrons $\beta\gamma$
$\mathcal{E}$	Normalized energy of electrons
$\mathcal{N}$	Denominator of normalized energies
F	Normalized formfactor of electrons, contains phase informations

This plugin allows to predict the emitted virtual transition radiation, which would be caused by the electrons in the simulation box passing through a virtual metal foil which is set at a specific location. The transition radiation can only be calculated for electrons at the moment.

10.18.1 External Dependencies

There are no external dependencies.

10.18.2 .param files

In order to setup the transition radiation plugin, the *transitionRadiation.param* has to be configured **and** the radiating particles need to have the attributes `weighting`, `momentum`, `location`, and `transitionRadiationMask` (which can be added in *speciesDefinition.param*) as well as the flags `massRatio` and `chargeRatio`.

In *transitionRadiation.param*, the number of frequencies `N_omega` and observation directions `N_theta` and `N_phi` are defined.

Frequency range

The frequency range is set up by choosing a specific namespace that defines the frequency setup

```
/* choose linear frequency range */
namespace radiation_frequencies = linear_frequencies;
```

Currently you can choose from the following setups for the frequency range:

namespace	Description
`linear_frequencies`	linear frequency range from `SI::omega_min` to `SI::omega_max` with `N_omega` steps
`log_frequencies`	logarithmic frequency range from `SI::omega_min` to `SI::omega_max` with `N_omega` steps
`frequencies_from_lis`	`N_omega` frequencies taken from a text file with location `listLocation[]`

All three options require variable definitions in the according namespaces as described below:

For the **linear frequency** scale all definitions need to be in the `picongpu::plugins::transitionRadiation::linear_frequen` namespace. The number of total sample frequencies `N_omega` need to be defined as `constexpr unsigned int`. In the sub-namespace `SI`, a minimal frequency `omega_min` and a maximum frequency `omega_max` need to be defined as `constexpr float_64`.

For the **logarithmic frequency** scale all definitions need to be in the `picongpu::plugins::transitionRadiation::log_frequencies` namespace. Equivalently to the linear case, three variables need to be defined: The number of total sample frequencies `N_omega` need to be defined as `constexpr unsigned int`. In the sub-namespace `SI`, a minimal frequency `omega_min` and a maximum frequency `omega_max` need to be defined as `constexpr float_64`.

For the **file-based frequency** definition, all definitions need to be in the `picongpu::plugins::transitionRadiation::frequencies_from_list` namespace. The number of total frequencies `N_omega` need to be defined as `constexpr unsigned int` and the path to the file containing the frequency values in units of $[s^{-1}]$ needs to be given as `constexpr const char * listLocation = "/path/to/frequency_list";`. The frequency values in the file can be separated by newlines, spaces, tabs, or any other whitespace. The numbers should be given in such a way, that c++ standard `std::ifstream` can interpret the number e.g., as `2.5344e+16`.

Note: Currently, the variable `listLocation` is required to be defined in the `picongpu::plugins::transitionRadiation::frequencies_from_list` namespace, even if `frequencies_from_list` is not used. The string does not need to point to an existing file, as long as the file-based frequency definition is not used.

Observation directions

The number of observation directions `N_theta` and the distribution of observation directions is defined in *transitionRadiation.param*. There, the function `observationDirection` defines the observation directions.

This function returns the x,y and z component of a **unit vector** pointing in the observation direction.

```
DINLINE vector_64
observationDirection( int const observation_id_extern )
{
    /* use the scalar index const int observation_id_extern to compute an
     * observation direction (x,y,y) */
    return vector_64( x , y , z );
}
```

Note: The `transitionRadiation.param` set up will be subject to **further changes**, since the `radiationObserver.param` it is based on is subject to further changes. These might be *namespaces* that describe several preconfigured layouts or a functor if *C++ 11* is included in the *nvcc*.

Foil Position

If one wants to virtually propagate the electron bunch to a foil in a further distance to get a rough estimate of the effect of the divergence on the electron bunch, one can include a foil position. A foil position which is unequal to zero, adds the electrons momentum vectors onto the electron until they reach the given y-coordinate. To contain the longitudinal information of the bunch, the simulation window is actually virtually moved to the foil position and not each single electron.

```
namespace SI
{
    // y position of the foil to calculate transition radiation at
```

(continues on next page)

(continued from previous page)

```
    // leave at 0 for no virtual particle propagation
    constexpr float_64 foilPosition = 0.0;
}
```

Note: This is an experimental feature, which was not verified yet.

Macro-particle form factor

The *macro-particle form factor* is a method, which considers the shape of the macro particles when computing the radiation.

One can select between different macro particle shapes. Currently eight shapes are implemented. A shape can be selected by choosing one of the available namespaces:

```
/* choosing the 3D CIC-like macro particle shape */
namespace radFormFactor = radFormFactor_CIC_3D;
```

Namespace	Description
`radFormFactor_CIC_3D`	3D Cloud-In-Cell shape
`radFormFactor_TSC_3D`	3D Triangular shaped density cloud
`radFormFactor_PCS_3D`	3D Quadratic spline density shape (Piecewise Cubic Spline assignment function)
`radFormFactor_CIC_1Dy`	Cloud-In-Cell shape in y-direction, dot like in the other directions
`radFormFactor_Gauss_sp`	symmetric Gauss charge distribution
`radFormFactor_Gauss_ce`	Gauss charge distribution according to cell size
`radFormFactor_incohere`	forces a completely incoherent emission by scaling the macro particle charge with the square root of the weighting
`radFormFactor_coherent`	forces a completely coherent emission by scaling the macro particle charge with the weighting

Note: One should not confuse this macro-particle form factor with the form factor F, which was previously mentioned. This form factor is equal to the macro-particle shape, while F contains the phase information of the whole electron bunch. Both are necessary for a physically correct transition radiation calculation.

Gamma filter

In order to consider the radiation only of particles with a gamma higher than a specific threshold. In order to do that, the radiating particle species needs the flag `transitionRadiationMask` (which is initialized as `false`) which further needs to be manipulated, to set to true for specific (random) particles.

Using a filter functor as:

```
using GammaFilter = picongpu::particles::manipulators::generic::Free<
    GammaFilterFunctor
>;
```

(see TransitionRadiation example for details) sets the flag to true if a particle fulfills the gamma condition.

Note: More sophisticated filters might come in the near future. Therefore, this part of the code might be subject to changes.

10.18.3 .cfg file

For a specific (charged) species `<species>` e.g. `e`, the radiation can be computed by the following commands.

Command line option	Description
`--<species>_transRad.period`	Gives the number of time steps between which the radiation should be calculated.

10.18.4 Memory Complexity

Accelerator

two counters (`float_X`) and two counters (`complex_X`) are allocated permanently

Host

as on accelerator.

10.18.5 Output

Contains *ASCII* files in `simOutput/transRad` that have the total spectral intensity until the timestep specified by the filename. Each row gives data for one observation direction (same order as specified in the `observer.py`). The values for each frequency are separated by *tabs* and have the same order as specified in `transitionRadiation.param`. The spectral intensity is stored in the units **[J s]**.

10.18.6 Analysing tools

The `transition_radiation_visualizer.py` in `lib/python/picongpu/extra/plugins/plot_mpl` can be used to analyze the radiation data after the simulation. See `transition-radiation_visualizer.py --help` for more information. It only works, if the input frequency are on a divided logarithmically!

10.18.7 Known Issues

The output is currently only physically correct for electron passing through a metal foil.

10.18.8 References

- ***Theory of coherent transition radiation generated at a plasma-vacuum interface***
  Schroeder, C. B. and Esarey, E. and van Tilborg, J. and Leemans, W. P., American Physical Society(2004), https://link.aps.org/doi/10.1103/PhysRevE.69.016501
- ***Diagnostics for plasma-based electron accelerators***
  Downer, M. C. and Zgadzaj, R. and Debus, A. and Schramm, U. and Kaluza, M. C., American Physical Society(2018), https://link.aps.org/doi/10.1103/RevModPhys.90.035002
- ***Synthetic characterization of ultrashort electron bunches using transition radiation***
  Carstens, F.-O., Bachelor thesis on the transition radiation plugin, https://doi.org/10.5281/zenodo.3469663
- ***Quantitatively consistent computation of coherent and incoherent radiation in particle-in-cell codes — A general form factor formalism for macro-particles***
  Pausch, R., Description for the effect of macro-particle shapes in particle-in-cell codes, https://doi.org/10.1016/j.nima.2018.02.020

10.19 Period Syntax

Most plugins allow to define a period on how often a plugin shall be executed (notified). Its simple syntax is: `<period>` with a simple number.

Additionally, the following syntax allows to define intervals for periods:

`<start>:<end>[:<period>]`

- *<start>*: begin of the interval; default: 0
- *<end>*: end of the interval, including the upper bound; default: end of the simulation
- *<period>*: notify period within the interval; default: 1

Multiple intervals can be combined via a comma separated list.

10.19.1 Examples

- `42` every 42th time step
- `::` equal to just writing 1, every time step from start (0) to the end of the simulation
- `11:11` only once at time step 11
- `10:100:2` every second time step between steps 10 and 100 (included)
- `42,30:50:10`: at steps 30 40 42 50 84 126 168 …
- `5,10`: at steps 0 5 10 15 20 25 … (only executed once per step in overlapping intervals)

10.20 Python Postprocessing

In order to further work with the data produced by a plugin during a simulation run, PIConGPU provides python tools that can be used for reading data and visualization. They can be found under `lib/python/picongpu/extra/plugins`.

Note: The python plugin tools have been moved to the *picongpu.extra* submodule.

It is our goal to provide at least three modules for each plugin to make postprocessing as convenient as possible: 1. a data reader (inside the `data` subdirectory) 2. a matplotlib visualizer (inside the `plot_mpl` subdirectory) 3. a jupyter widget visualizer (inside the `jupyter_widgets` subdirectory) for usage in jupyter-notebooks

Further information on how to use these tools can be found at each plugin page.

If you would like to help in developing those classes for a plugin of your choice, please read *python postprocessing*.

References

CHAPTER

ELEVEN

TBG

Section author: Axel Huebl, Klaus Steiniger

Module author: René Widera

Our tool *template batch generator* (tbg) abstracts program runtime options from technical details of supercomputers. On a desktop PC, one can just execute a command interactively and instantaneously. Contrarily on a supercomputer, resources need to be shared between different users efficiently via *job scheduling*. Scheduling on today's supercomputers is usually done via *batch systems* that define various queues of resources.

An unfortunate aspect about batch systems from a user's perspective is, that their usage varies a lot. And naturally, different systems have different resources in queues that need to be described.

PIConGPU runtime options are described in *configuration files* (.cfg). We abstract the description of queues, resource acquisition and job submission via *template files* (.tpl). For example, a .cfg file defines how many *devices* shall be used for computation, but a .tpl file calculates how many *physical nodes* will be requested. Also, .tpl files takes care of how to spawn a process when scheduled, e.g. with mpiexec and which flags for networking details need to be passed. After combining the *machine independent* (portable) .cfg file from user input with the *machine dependent* .tpl file, tbg can submit the requested job to the batch system.

Last but not least, one usually wants to store the input of a simulation with its output. tbg conveniently automates this task before submission. The .tpl and the .cfg files that were used to start the simulation can be found in <tbg destination dir>/tbg/ and can be used together with the .param files from <tbg destination dir>/input/.../param/ to recreate the simulation setup.

In summary, PIConGPU runtime options in .cfg files are portable to any machine. When accessing a machine for the first time, one needs to write template .tpl files, abstractly describing how to run PIConGPU on the specific queue(s) of the batch system. We ship such template files already for a set of supercomputers, interactive execution and many common batch systems. See $PICSRC/etc/picongpu/ and *our list of systems with .profile files* for details.

11.1 Usage

```
TBG (template batch generator)
create a new folder for a batch job and copy in all important files

usage: tbg -c [cfgFile] [-s [submitsystem]] [-t [templateFile]]
          [-o "VARNAME1=10 VARNAME2=5"] [-f] [-h]
          [projectPath] destinationPath

recommended usage when sourcing a PIConGPU config file before:
    tbg -s -t -c cfgFile destinationPath

-c | --cfg      [file]         - Configuration file to set up batch file.
                                 Default: [cfgFile] via export TBG_CFGFILE
-s | --submit   [command]      - Submit command (qsub, "qsub -h", sbatch, ...)
                                 Default: [submitsystem] via export TBG_SUBMIT
```

(continues on next page)

(continued from previous page)

```
-t | --tpl      [file]         - Template to create a batch file from.
                                 tbg will use stdin, if no file is specified.
                                 Default: [templateFile] via export TBG_TPLFILE
                                 Warning: If -t is omitted, stdin will be used as
                                 input for the template.
-o                             - Overwrite any template variable:
                                 spaces within the right side of assign are not␣
↪allowed
                                 e.g. -o "VARNAME1=10 VARNAME2=5"
                                 Overwriting is done after cfg file was executed
-f | --force                   - Override if 'destinationPath' exists.
-h | --help                    - Shows help (this output).

[projectPath]                  - Project directory containing source code and
                                 binaries
                                 Default: current directory
destinationPath                - Directory for simulation output.


TBG exports the following variables, which can be used in cfg and tpl files at
any time:
 TBG_jobName                   - name of the job
 TBG_jobNameShort              - short name of the job, without blanks
 TBG_cfgPath                   - absolute path to cfg file
 TBG_cfgFile                   - full absolute path and name of cfg file
 TBG_projectPath               - absolute project path (see optional parameter
                                 projectPath)
 TBG_dstPath                   - absolute path to destination directory
```

11.2 .cfg File Macros

Feel free to copy & paste sections of the files below into your `.cfg`, e.g. to configure complex plugins:

```
# Copyright 2014-2023 Felix Schmitt, Axel Huebl, Richard Pausch, Heiko Burau,
#                     Franz Poeschel, Sergei Bastrakov, Pawel Ordyna
#
# This file is part of PIConGPU.
#
# PIConGPU is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# PIConGPU is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with PIConGPU.
# If not, see <http://www.gnu.org/licenses/>.

################################################################################
## This file describes sections and variables for PIConGPU's
```

(continues on next page)

(continued from previous page)

```
## TBG batch file generator.
## These variables basically wrap PIConGPU command line flags.
## To see all flags available for your PIConGPU binary, run
## picongpu --help. The avalable flags depend on your configuration flags.
## Note that this is not meant to be a complete and functioning .cfg file.
##
## Flags that target a specific species e.g. electrons (--e_png) or ions
## (--i_png) must only be used if the respective species is activated (configure␣
↪flags).
##
## If not stated otherwise, variables/flags must not be used more than once!
##############################################################################

##############################################################################
## Section: Required Variables
## Variables in this secton are necessary for PIConGPU to work properly and should not
## be removed. However, you are free to adjust them to your needs, e.g. setting
## the number of GPUs in each dimension.
##############################################################################

# Batch system walltime
TBG_wallTime="1:00:00"

# Number of devices in each dimension (x,y,z) to use for the simulation
TBG_devices_x=1
TBG_devices_y=2
TBG_devices_z=1

# Size of the simulation grid in cells as "X Y Z"
# note: the number of cells needs to be an exact multiple of a supercell
#       and has to be at least 3 supercells per device,
#       the size of a supercell (in cells) is defined in `memory.param`
TBG_gridSize="128 256 128"

# Number of simulation steps/iterations as "N"
TBG_steps="100"

# disable grid size auto adjustment
TBG_disableGridAutoAdjustment="--autoAdjustGrid off"

##############################################################################
## Section: Optional Variables
## You are free to add and remove variables here as you like.
## The only exception is TBG_plugins which is used to forward your variables
## to the TBG program. This variable can be modified but should not be removed!
##
## Please add all variables you define in this section to TBG_plugins.
##############################################################################

# Variables which are created by TBG (should be self-descriptive)
TBG_jobName
TBG_jobNameShort
TBG_cfgPath
TBG_cfgFile
TBG_projectPath
TBG_dstPath
```

(continues on next page)

(continued from previous page)

```
# version information on startup
TBG_version="--versionOnce"

# Set progress output period in stdout:
# -p or --progress can be used to set an integer percent value (of the total␣
↪simulation duration).
# It is also possible to set the exact time steps with --progressPeriod and the␣
↪plugin period syntax.
# --progress and --progressPeriod are combined. For setting the period only in␣
↪percent just leave the --progressPeriod option out.
# For setting the period only with the plugin syntax use  -p 0 to disable the default␣
↪-p 5 setting.
# Example for writing every 2 % as well as every 100 steps and every 20 steps from␣
↪the 100th to the 1000th step.
TBG_progress="-p 2 --progressPeriod 100,100:1000:20"


# Regex to describe the static distribution of the cells for each device
# default: equal distribution over all devices
# example for -d 2 4 1 -g 128 192 12
TBG_gridDist="--gridDist '64{2}' '64,32{2},64'"


# Specifies whether the grid is periodic (1) or not (0) in each dimension (X,Y,Z).
# Default: no periodic dimensions
TBG_periodic="--periodic 1 0 1"


# Specifies boundaries for given species with a particle pusher, 'e' in the example.
# The axis order matches --periodic.
# Default: what is set by --periodic, all offsets 0.
# Supported particle boundary kinds: periodic, absorbing, reflecting, thermal.
# Periodic boundaries require 0 offset, thermal require a positive offset, other␣
↪boundary kinds can have non-negative offsets.
# Boundary temperature is set in keV, only affects thermal boundaries.
TBG_particleBoundaries="--e_boundary periodic absorbing thermal --e_boundaryOffset 0␣
↪10 5 --e_boundaryTemperature 0 0 20.0"


# Specify runtime density file for given species, 'e' in the example.
# Only has effect when FromOpenPMDImpl<Param> density is used for the species and its␣
↪Param::filename is empty.
# In case the species has multiple such FromOpenPMDImpl invocations, same runtime␣
↪value is used for all of them.
TBG_runtimeDensityFile="--e_runtimeDensityFile /bigdata/hplsim/production/
↪someDirectory/density.bp"


# Set absorber type of absorbing boundaries.
# Supported values: exponential, pml (default).
# When changing absorber type, one should normally adjust NUM_CELLS in fieldAbsorber.
↪param
TBG_absorber="--fieldAbsorber pml"
```

(continues on next page)

(continued from previous page)

```
# Enables moving window (sliding) in your simulation
TBG_movingWindow="-m"


# Defines when to start sliding the window.
# The window starts sliding at the time required to pass the distance of
# windowMovePoint * (global window size in y) when moving with the speed of light
# Note: beware, there is one "hidden" row of gpus at the y-front, so e.g. when the␣
↪window is enabled
# and this variable is set to 0.75, only 75% of simulation area is used for real␣
↪simulation
TBG_windowMovePoint="--windowMovePoint 0.9"


# stop the moving window after given simulation step
TBG_stopWindow="--stopWindow 1337"


# Set current smoothing.
# Supported values: none (default), binomial
TBG_currentInterpolation="--currentInterpolation binomial"


# Duplicate E and B field storage inside field background to improve performance at␣
↪cost of additional memory
TBG_fieldBackground="--fieldBackground.duplicateFields"


# Allow two MPI ranks to use one compute device.
TBG_ranksPerDevice="--numRanksPerDevice 2"

################################################################################
## Placeholder for multi data plugins:
##
## placeholders must be substituted with the real data name
##
## <species> = species name e.g. e (electrons), i (ions)
## <field>  = field names e.g. FieldE, FieldB, FieldJ
################################################################################

# The following flags are available for the radiation plugin.
# For a full description, see the plugins section in the online wiki.
#--<species>_radiation.period     Radiation is calculated every .period steps.␣
↪Currently 0 or 1
#--<species>_radiation.dump     Period, after which the calculated radiation data␣
↪should be dumped to the file system
#--<species>_radiation.lastRadiation     If flag is set, the spectra summed between␣
↪the last and the current dump-time-step are stored
#--<species>_radiation.folderLastRad     Folder in which the summed spectra are stored
#--<species>_radiation.totalRadiation     If flag is set, store spectra summed from␣
↪simulation start till current time step
#--<species>_radiation.folderTotalRad     Folder in which total radiation spectra are␣
↪stored
#--<species>_radiation.start     Time step to start calculating the radition
#--<species>_radiation.end     Time step to stop calculating the radiation
#--<species>_radiation.radPerGPU     If flag is set, each GPU stores its own spectra␣
```

(continues on next page)

(continued from previous page)

```
↪without summing the entire simulation area
#--<species>_radiation.folderRadPerGPU     Folder where the GPU specific spectras are␣
↪stored
#--<species>_radiation.numJobs     Number of independent jobs used for the radiation␣
↪calculation.
#--<species>_radiation.openPMDSuffix   Suffix for openPMD filename extension and␣
↪iteration expansion pattern.
#--<species>_radiation.openPMDCheckpointExtension    Filename extension for openPMD␣
↪checkpoints.
#--<species>_radiation.openPMDConfig    JSON/TOML configuration for initializing␣
↪openPMD checkpointing.
#--<species>_radiation.openPMDCheckpointConfig    JSON/TOML configuration for␣
↪initializing openPMD checkpointing.
#--<species>_radiation.distributedAmplitude    Additionally output distributed␣
↪amplitudes per MPI rank.
TBG_radiation="--<species>_radiation.period 1 --<species>_radiation.dump 2 --<species>
↪_radiation.totalRadiation \
               --<species>_radiation.lastRadiation --<species>_radiation.start 2800 --
↪<species>_radiation.end 3000"

# The following flags are available for the transition radiation plugin.
# For a full description, see the plugins section in the online documentation.
#--<species>_transRad.period   Gives the number of time steps between which the␣
↪radiation should be calculated.
TBG_transRad="--<species>_transRad.period 1000"

# Create 2D images in PNG format every .period steps.
# The slice plane is defined using .axis [yx,yz] and .slicePoint (offset from origin
# as a float within [0.0,1.0].
# The output folder can be set with .folder.
# Can be used more than once to print different images, e.g. for YZ and YX planes.
TBG_<species>_pngYZ="--<species>_png.period 10 --<species>_png.axis yz --<species>_
↪png.slicePoint 0.5 --<species>_png.folder pngElectronsYZ"
TBG_<species>_pngYX="--<species>_png.period 10 --<species>_png.axis yx --<species>_
↪png.slicePoint 0.5 --<species>_png.folder pngElectronsYX"

# Create a particle-energy histogram [in keV] per species for every .period steps
TBG_<species>_histogram="--<species>_energyHistogram.period 500 --<species>_
↪energyHistogram.binCount 1024     \
                         --<species>_energyHistogram.minEnergy 0 --<species>_
↪energyHistogram.maxEnergy 500000 \
                         --<species>_energyHistogram.filter all"


# Calculate a 2D phase space
# - momentum range in m_<species> c
TBG_<species>_PSxpx="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter␣
↪all --<species>_phaseSpace.space x --<species>_phaseSpace.momentum px --<species>_
↪phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
TBG_<species>_PSxpz="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter␣
↪all --<species>_phaseSpace.space x --<species>_phaseSpace.momentum pz --<species>_
↪phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
TBG_<species>_PSypx="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter␣
↪all --<species>_phaseSpace.space y --<species>_phaseSpace.momentum px --<species>_
↪phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
TBG_<species>_PSypy="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter␣
```

(continues on next page)

(continued from previous page)

```
↪all --<species>_phaseSpace.space y --<species>_phaseSpace.momentum py --<species>_
↪phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"
TBG_<species>_PSypz="--<species>_phaseSpace.period 10 --<species>_phaseSpace.filter␣
↪all --<species>_phaseSpace.space y --<species>_phaseSpace.momentum pz --<species>_
↪phaseSpace.min -1.0 --<species>_phaseSpace.max 1.0"

# Sum up total energy every .period steps for
# - species   (--<species>_energy)
# - fields    (--fields_energy)
TBG_sumEnergy="--fields_energy.period 10 --<species>_energy.period 10 --<species>_
↪energy.filter all"


# Count the number of macro particles per species for every .period steps
TBG_macroCount="--<species>_macroParticlesCount.period 100"


# Count makro particles of a species per super cell
TBG_countPerSuper="--<species>_macroParticlesPerSuperCell.period 100 --<species>_
↪macroParticlesPerSuperCell.period 100"


# Dump simulation data (fields and particles) via the openPMD API.
# Data is dumped every .period steps to the fileset .file.
# To select only parts of data .range can be used.
TBG_openPMD="--openPMD.period 100   \
             --openPMD.file simOutput \
             --openPMD.ext bp \
             --openPMD.json '{ \"adios2\": { \"engine\": { \"type\": \"file\", \
↪"parameters\": { \"BufferGrowthFactor\": \"1.2\", \"InitialBufferSize\": \"2GB\" } }
↪ } }' \
             --openPMD.range 0:100,:,:"
# Further control over the backends used in the openPMD plugins is available
# through the mechanisms exposed by the openPMD API:
# * environment variables
# * JSON-formatted configuration string
# Further information on both is retrieved from the official documentation
# https://openpmd-api.readthedocs.io

# Create a checkpoint that is restartable every --checkpoint.period steps
#   http://git.io/PToFYg
TBG_checkpoint="--checkpoint.period 1000"
# Time periodic checkpoint creation [period in minutes]
TBG_checkpointTime="--checkpoint.timePeriod 2"
# Select the backend for the checkpoint, available are openPMD
#    --checkpoint.backend openPMD
# Available backend options are exactly as in --openPMD.* and can be set
# via:
#   --checkpoint.<IO-backend>.* <value>
# e.g.:
#   --checkpoint.openPMD.dataPreparationStrategy doubleBuffer
# One additional parameter is available for configuring the openPMD-api via JSON
# for restarting procedures:
TBG_openPMD_restart="--checkpoint.openPMD.jsonRestart '{\"adios2\": {\"dataset\": {\
↪"operators\": [{\"type\": \"blosc\",\"parameters\": {\"nthreads\": 8}}]}}}'"
```

(continues on next page)

(continued from previous page)

```
# Restart the simulation from checkpoint created using TBG_checkpoint
TBG_restart="--checkpoint.restart"
# Try to restart if a checkpoint is available else start the simulation from scratch.
TBG_tryrestart="--checkpoint.tryRestart"
# Select the backend for the restart (must fit the created checkpoint)
#    --checkpoint.restart.backend openPMD
# By default, the last checkpoint is restarted if not specified via
#   --checkpoint.restart.step 1000
# To restart in a new run directory point to the old run where to start from
#   --checkpoint.restart.directory /path/to/simOutput/checkpoints

# Presentation mode: loop a simulation via restarts
#   does either start from 0 again or from the checkpoint specified with
#   --checkpoint.restart.step as soon as the simulation reached the last time step;
#   in the example below, the simulation is run 5000 times before it shuts down
# Note: does currently not work with `Radiation` plugin
TBG_restartLoop="--checkpoint.restart.loop 5000"

# Live in situ visualization using ISAAC
#   Initial period in which a image shall be rendered
#     --isaac.period PERIOD
#   Name of the simulation run as seen for the connected clients
#     --isaac.name NAME
#   URL of the server
#     --isaac.url URL
#   Number from 1 to 100 decribing the quality of the transceived jpeg image.
#   Smaller values are faster sent, but of lower quality
#     --isaac.quality QUALITY
#   Resolution of the rendered image. Default is 1024x768
#     --isaac.width WIDTH
#     --isaac.height HEIGHT
#   Pausing directly after the start of the simulation
#     --isaac.directPause off=0(default)|on=1
#   By default the ISAAC Plugin tries to reconnect if the sever is not available
#   at start or the servers crashes. This can be deactivated with this option
#     --isaac.reconnect false
#   Enable and write benchmark results into the given file.
#     --isaac.timingsFilename benchResults.txt
TBG_isaac="--isaac.period 1 --isaac.name !TBG_jobName --isaac.url <server_url>"
TBG_isaac_quality="--isaac.quality 90"
TBG_isaac_resolution="--isaac.width 1024 --isaac.height 768"
TBG_isaac_pause="--isaac.directPause 1"
TBG_isaac_reconnect="--isaac.reconnect false"

# Print the maximum charge deviation between particles and div E to textfile
↪'chargeConservation.dat':
TBG_chargeConservation="--chargeConservation.period 100"

# Particle calorimeter: (virtually) propagates and collects particles to infinite␣
↪distance
TBG_<species>_calorimeter="--<species>_calorimeter.period 100 --<species>_calorimeter.
↪openingYaw 90 --<species>_calorimeter.openingPitch 30
                        --<species>_calorimeter.numBinsEnergy 32 --<species>_
↪calorimeter.minEnergy 10 --<species>_calorimeter.maxEnergy 1000
                        --<species>_calorimeter.logScale 1 --<species>_calorimeter.
↪file filePrefix --<species>_calorimeter.filter all"
```

(continues on next page)

(continued from previous page)

```
#################################################################################
## Section: Program Parameters
## This section contains TBG internal variables, often composed from required
## variables. These should not be modified except when you know what you are doing!
#################################################################################

# Number of compute devices in each dimension as "X Y Z"
TBG_deviceDist="!TBG_devices_x !TBG_devices_y !TBG_devices_z"


# Combines all declared variables. These are passed to PIConGPU as command line flags.
# The program output (stdout) is stored in a file called output.stdout.
TBG_programParams="-d !TBG_deviceDist \
                   -g !TBG_gridSize   \
                   -s !TBG_steps      \
                   !TBG_plugins"

# Total number of devices
TBG_tasks="$(( TBG_devices_x * TBG_devices_y * TBG_devices_z ))"
```

11.3 Batch System Examples

Section author: Axel Huebl, Richard Pausch, Klaus Steiniger

11.3.1 Linux workstation

PIConGPU can run on your laptop or workstation, even if there is no dedicated GPU available. In this case it will run on the CPU.

In order to run PIConGPU on your machine, use `bash` as the submit command, i.e. `tbg -s bash -t etc/picongpu/bash/mpirun.tpl -c etc/picongpu/1.cfg $SCRATCH/picRuns/001`

11.3.2 Slurm

Slurm is a modern batch system, e.g. installed on the Taurus cluster at TU Dresden, Hemera at HZDR, Cori at NERSC, among others.

Job Submission

PIConGPU job submission on the *Taurus* cluster at *TU Dresden*:

- `tbg -s sbatch -c etc/picongpu/0008gpus.cfg -t etc/picongpu/taurus-tud/k80.tpl $SCRATCH/runs/test-001`

Job Control

- interactive job:
  - `salloc --time=1:00:00 --nodes=1 --ntasks-per-node=2 --cpus-per-task=8 --partition gpu-interactive`
  - e.g. `srun "hostname"`
  - GPU allocation on taurus requires an additional flag, e.g. for two GPUs `--gres=gpu:2`
- details for my jobs:
  - `scontrol -d show job 12345` all details for job with <job id> `12345`
  - `squeue -u $(whoami) -l` all jobs under my user name
- details for queues:
  - `squeue -p queueName -l` list full queue
  - `squeue -p queueName --start` (show start times for pending jobs)
  - `squeue -p queueName -l -t R` (only show running jobs in queue)
  - `sinfo -p queueName` (show online/offline nodes in queue)
  - `sview` (alternative on taurus: `module load llview` and `llview`)
  - `scontrol show partition queueName`
- communicate with job:
  - `scancel <job id>` abort job
  - `scancel -s <signal number> <job id>` send signal or signal name to job
  - `scontrol update timelimit=4:00:00 jobid=12345` change the walltime of a job
  - `scontrol update jobid=12345 dependency=afterany:54321` only start job `12345` after job with id `54321` has finished
  - `scontrol hold <job id>` prevent the job from starting
  - `scontrol release <job id>` release the job to be eligible for run (after it was set on hold)

11.3.3 LSF

LSF (for *Load Sharing Facility*) is an IBM batch system (bsub/BSUB). It is used, e.g. on Summit at ORNL.

Job Submission

PIConGPU job submission on the *Summit* cluster at *Oak Ridge National Lab*:

- `tbg -s bsub -c etc/picongpu/0008gpus.cfg -t etc/picongpu/summit-ornl/gpu.tpl $PROJWORK/$proj/test-001`

Job Control

- interactive job:
  - `bsub -P $proj -W 2:00 -nnodes 1 -Is /bin/bash`
- details for my jobs:
  - `bjobs 12345` all details for job with <job id> `12345`
  - `bjobs [-l]` all jobs under my user name
  - `jobstat -u $(whoami)` job eligibility
  - `bjdepinfo 12345` job dependencies on other jobs
- details for queues:
  - `bqueues` list queues
- communicate with job:
  - `bkill <job id>` abort job
  - `bpeek [-f] <job id>` peek into `stdout`/`stderr` of a job
  - `bkill -s <signal number> <job id>` send signal or signal name to job
  - `bchkpnt` and `brestart` checkpoint and restart job (untested/unimplemented)
  - `bmod -W 1:30 12345` change the walltime of a job (currently not allowed)
  - `bstop <job id>` prevent the job from starting
  - `bresume <job id>` release the job to be eligible for run (after it was set on hold)

References

- https://docs.olcf.ornl.gov/systems/summit_user_guide.html#running-jobs
- https://www.ibm.com/docs/en/spectrum-lsf

CHAPTER

TWELVE

PICMI

PIConGPU supports PICMI to create simulation setups from Python.

Note: This is the **user** documentation. For the developer documentation on the translation layer *PyPIConGPU* see *here*.

Warning: The PICMI support is still **experimental**. You are very much encouraged to give feedback on your workflow, this documentation, the PICMI definition and everything related.

12.1 Intro

This is a brief overview over PICMI.

12.1.1 Example

```python
from picongpu import picmi

boundary_conditions = ["periodic", "periodic", "periodic"]
grid = picmi.Cartesian3DGrid(
    # note: [x] * 3 == [x, x, x]
    number_of_cells=[192] * 3,
    lower_bound=[0, 0, 0],
    upper_bound=[0.0111152256] * 3,
    # delta {x, y, z} is computed implicitly
    # lower & upper boundary conditions must be equal
    lower_boundary_conditions=boundary_conditions,
    upper_boundary_conditions=boundary_conditions,
)
solver = picmi.ElectromagneticSolver(method="Yee", grid=grid)

profile = picmi.UniformDistribution(
    density=1e20,
    # most probable E_kin = 5 mc^2
    # approx. 9000 keV for electrons
    # must be equal for all three components
    rms_velocity=[4.18 * picmi.constants.c] * 3,
)
electron = picmi.Species(
    name="e",
    # openPMD particle type
```

(continues on next page)

(continued from previous page)

```
    particle_type="electron",
    initial_distribution=profile,
)

sim = picmi.Simulation(
    time_step_size=9.65531e-14,
    max_steps=1024,
    solver=solver,
)

layout = picmi.PseudoRandomLayout(n_macroparticles_per_cell=25)
sim.add_species(electron, layout)

sim.write_input_file("generated_input")
```

Creates a directory `generated_input`, where you can run `pic-build` and subsequently `tbg`.

12.1.2 Generation Results

The recommended way to use the generated simulations is to

1. create the simulation in PICMI
2. call `Simulation.write_input_file(DIR)`
3. use the normal PIConGPU toolchain (`pic-build`, `tbg`)

Note: Rationale: PICMI does not (yet) support enough parameters to meaningfully control the execution process.

Additionally, the following methods work (but are **not recommended**):

- call `Simulation.step(NUM)`
  - directly builds and runs the simulation
  - `NUM` must be the **maximum number of steps**
  - has no diagnostic output (i.e. console hangs without output)
- call `Simulation.picongpu_run()`
  - equivalent to `Simulation.step()` with the maximum number of steps
- use the *PyPIConGPU runner*

12.1.3 Output

Warning: This is subject to change.

Output is currently **not configurable**.

Some output is automatically enabled, including PNGs. For this the period is chosen that the output is generated (approx.) 100 times over the entire simulation duration.

To configure output you must *change the generated files*.

12.1.4 Unsupported Features

You will be alerted for unsupported features.

Either a warning is printed or an error thrown (including because a class does not exist). In this case read the error message to fix this.

For reference you can see how the tests in `$PICSRC/test/python/picongpu/quick/picmi` use the interface.

12.1.5 Reference

The full PICMI reference is available upstream.

12.1.6 Extensions

Parameters/Methods prefixed with `picongpu_` are PIConGPU-exclusive.

Warning: We strive to quickyl contribute these parameters to PICMI upstream, so this list is to be considered volatile.

- Simulation
  - `picongpu_get_runner()`: Retrieve a *PyPIConGPU Runner*
  - `picongpu_template_dir`: Specify the template dir to use for code generation, please refer to *the documentation on the matter for details*
- Grid
  - `picongpu_n_gpus`: list of a 1 or 3 integers, greater than zero, describing GPU distribution in space 3-integer list: `[N_gpu_x, N_gpu_y, N_gpu_z]` 1-integer list: `[1, N_gpu_y, 1]` Default is `None` equal to `[1, 1, 1]`
- Gaussian Laser
  - Laguerre Modes (`picongpu_laguerre_modes` and `picongpu_laguerre_phases`): Two lists of float, passed to PIConGPU laser definition
- Species
  - `picongpu_ionization_electrons`: Electron species to use for ionization. Optional, will be guessed if possible.
  - `picongpu_fully_ionized`: When defining an element (using `particle_type`) it may or may not be ionizable
    - to **enable** ionization simulation set `charge_state` to an integer
    - to **disable** ionization (ions are only core without electrons) set `picongpu_fully_ionized=True`

    If neither is set a warning is printed prompting for either of the options above.

12.2 Custom Code Generation

The PIConGPU code generation works by encoding all parameters into a JSON representation, which is combined with a *template* to *render* the final configuration.

You can supply a **custom template** to **modify the generated code**, e.g. to enable additional output.

Note: It is even possible to use the JSON data during the generation. To get started read the *explanation of the translation process*, and especially the section on the *templating engine *Mustache**.

To see an example of the JSON data that is used either generate an input set with PICMI and examine the generated `pypicongpu.json` or simply have a look at the *example in the documentation*.

12.2.1 Step-by-Step Guide

1. Create a copy of the template: `cp -r $PICSRC/share/picongpu/pypicongpu/template my_template_dir`
2. Change whatever you need in the template

   e.g. `vim my_template_dir/etc/picongpu/N.cfg.mustache`

   find `pypicongpu_output_with_newlines` and insert:

```
--openPMD.period 10
--openPMD.file simData
--openPMD.ext bp
--checkpoint.backend openPMD
--checkpoint.period 100
--checkpoint.restart.backend openPMD
```

3. supply your template dir in your PICMI script:

```
# other setup...
sim = picmi.Simulation(
    time_step_size=9.65531e-14,
    max_steps=1024,
    solver=solver,
    # sets custom template dir
    picongpu_template_dir="my_template_dir")

sim.write_input_file("generated_input")
```

4. run PICMI script
5. inspect generated files

   e.g. `less generated_input/etc/picongpu/N.cfg` now contains the output added above

Warning: It is highly discouraged to incorporate editing generated PIConGPU input files **after** generation – just because it is very easy to make mistakes this way. Try to use the process outlined here.

CHAPTER

THIRTEEN

PYTHON UTILITIES

This section contains python utilities for more comfortable working with PIConGPU.

13.1 Memory Calculator

To aid you in the planning and setup of your simulation PIConGPU provides python tools for educated guesses on simulation parameters. They can be found under `lib/python/picongpu/extra/utils`.

Note: The utils have been moved to the *picongpu.extra* submodule.

Calculate the memory requirement per device.

```
from picongpu.extra.utils import MemoryCalculator
```

`class picongpu.extra.utils.memory_calculator.`**`MemoryCalculator`**(*n_x*, *n_y*, *n_z*, *precision_bits=32*)

Memory requirement calculation tool for PIConGPU

Contains calculation for fields, particles, random number generator and the calorimeter plugin. In-situ methods other than the calorimeter so far use up negligible amounts of memory on the device.

`__init__`(*n_x*, *n_y*, *n_z*, *precision_bits=32*)

Class constructor

Parameters

- **`n_x`** (*`int`*) – number of cells in x direction (per device)
- **`n_y`** (*`int`*) – number of cells in y direction (per device)
- **`n_z`** (*`int`*) – number of cells in z direction (per device)
- **`precision_bits`** (*`int`*) – floating point precision for PIConGPU run

`mem_req_by_calorimeter`(*n_energy*, *n_yaw*, *n_pitch*, *value_size=None*)

Memory required by the particle calorimeter plugin. Each of the (`n_energy` x `n_yaw` x `n_pitch`) bins requires a value (32/64 bits). The whole calorimeter is represented twice on each device, once for particles in the simulation and once for the particles that leave the box.

Parameters

- **`n_energy`** (*`int`*) – number of bins on the energy axis
- **`n_yaw`** (*`int`*) – number of bins for the yaw angle
- **`n_pitch`** (*`int`*) – number of bins for the pitch angle
- **`value_size`** (*`int`*) – value size in particle calorimeter {unit: byte} (default: 4)

Returns
req_mem – required memory {unit: bytes} per device

Return type
int

`mem_req_by_fields`(*n_x=None*, *n_y=None*, *n_z=None*, *field_tmp_slots=1*, *particle_shape_order=2*, *sim_dim=3*, *pml_n_x=0*, *pml_n_y=0*, *pml_n_z=0*)

Memory reserved for fields on each device

Parameters

- **n_x** (*int*) – number of cells in x direction (per device)
- **n_y** (*int*) – number of cells in y direction (per device)
- **n_z** (*int*) – number of cells in z direction (per device)
- **field_tmp_slots** (*int*) – number of slots for temporary fields (see PIConGPU `memory.param` : `fieldTmpNumSlots`)
- **particle_shape_order** (*int*) – numerical order of the assignment function of the chosen particle shape CIC : order 1 TSC : order 2 PQS : order 3 PCS : order 4 (see PIConGPU `species.param`)
- **sim_dim** (*int*) – simulation dimension (available for PIConGPU: 2 and 3)
- **pml_n_x** (*int*) – number of PML cells in x direction, combined for both sides
- **pml_n_y** (*int*) – number of PML cells in y direction, combined for both sides
- **pml_n_z** (*int*) – number of PML cells in z direction, combined for both sides

Returns
req_mem – required memory {unit: bytes} per device

Return type
int

`mem_req_by_particles`(*target_n_x=None*, *target_n_y=None*, *target_n_z=None*, *num_additional_attributes=0*, *particles_per_cell=2*, *sim_dim=3*)

Memory reserved for all particles of a species on a device. We currently neglect the constant species memory.

Parameters

- **target_n_x** (*int*) – number of cells in x direction containing the target
- **target_n_y** (*int*) – number of cells in y direction containing the target
- **target_n_z** (*int*) – number of cells in z direction containing the target
- **num_additional_attributes** (*int*) – number of additional attributes like e.g. `boundElectrons`
- **particles_per_cell** (*int*) – number of particles of the species per cell
- **sim_dim** (*int*) – simulation dimension (available for PIConGPU: 2 and 3)

Returns
req_mem – required memory {unit: bytes} per device and species

Return type
int

`mem_req_by_rng`(*n_x=None*, *n_y=None*, *n_z=None*, *generator_method='XorMin'*)

Memory reserved for the random number generator state on each device.

Check `random.param` for a choice of random number generators. If you find that your required RNG state is large (> 300 MB) please see `memory.param` for a possible adjustment of the `reservedGpuMemorySize`.

Parameters

- **`n_x`** (*int*) – number of cells in x direction (per device)
- **`n_y`** (*int*) – number of cells in y direction (per device)
- **`n_z`** (*int*) – number of cells in z direction (per device)
- **`generator_method`** (*str*) – random number generator method - influences the state size per cell possible options: "XorMin", "MRG32k3aMin", "AlpakaRand" - (GPU default: "XorMin") - (CPU default: "AlpakaRand")

Returns

req_mem – required memory {unit: bytes} per device

Return type

int

CHAPTER

FOURTEEN

EXAMPLE SETUPS

14.1 Bunch: Thomson scattering from laser electron-bunch interaction

Section author: Richard Pausch <r.pausch (at) hzdr.de>

Module author: Richard Pausch <r.pausch (at) hzdr.de>, Rene Widera <r.widera (at) hzdr.de>

This is a simulation of an electron bunch that collides head-on with a laser pulse. Depending on the number of electrons in the bunch, their momentum and their distribution and depending on the laser wavelength and intensity, the emitted radiation differs. A general description of this simulation can be found in [PauschDipl]. A detailed analysis of this bunch simulation can be found in [Pausch13]. A theoretical study of the emitted radiation in head-on laser electron collisions can be found in [Esarey93].

This test simulates an electron bunch with a relativistic gamma factor of gamma=5.0 and with a laser with a_0=1.0. The resulting radiation should scale with the number of real electrons (incoherent radiation).

14.1.1 References

14.2 Empty: Default PIC Algorithm

Section author: Axel Huebl <a.huebl (at) hzdr.de>

This is an "empty" example, initializing a default particle-in-cell cycle with default algorithms [BirdsallLangdon] [HockneyEastwood] but without a specific test case. When run, it iterates a particle-in-cell algorithm on a vacuum without particles or electro-magnetic fields initialized, which are the default `.param` files in `include/picongpu/param/`.

This is a case to demonstrate and test these defaults are still (syntactically) working. In order to set up your own simulation, there is no need to overwrite all `.param` files but only the ones that are different from the defaults. As an example, just overwrite the default laser (none) and initialize a species with a density distribution.

14.2.1 References

14.3 FoilLCT: Ion Acceleration from a Liquid-Crystal Target

Section author: Axel Huebl

Module author: Axel Huebl, T. Kluge

The following example models a laser-ion accelerator in the [TNSA] regime. An optically over-dense target ($n_{\max} = 192 n_{\mathrm{c}}$) consisting of a liquid-crystal material *8CB* (4-octyl-4'-cyanobiphenyl) $C_{21}H_{25}N$ is used [LCT].

Irradiated with a high-power laser pulse with $a_0 = 5$ the target is assumed to be partly pre-ionized due to realistic laser contrast and pre-pulses to C^{2+}, H^{+} and N^{2+} while being slightly expanded on its surfaces (modeled as exponential density slope). The overall target is assumed to be initially quasi-neutral and the *8CB* ion components are are not demixed in the surface regions. Surface contamination with, e.g. water vapor is neglected.

The laser is assumed to be in focus and approximated as a plane wave with temporally Gaussian intensity envelope of $\tau_I^{\mathrm{FWHM}} = 25$ fs.

This example is used to demonstrate:

- an ion acceleration setup with
- *composite, multi ion-species target material*
- *quasi-neutral initial conditions*
- ionization models for *field ionization* and *collisional ionization*

with PIConGPU.

14.3.1 References

14.4 KelvinHelmholtz: Kelvin-Helmholtz Instability

Section author: Axel Huebl <a.huebl (at) hzdr.de>

Module author: Axel Huebl <a.huebl (at) hzdr.de>, E. Paulo Alves, Thomas Grismayer

This example simulates a shear-flow instability known as the Kelvin-Helmholtz Instability in a near-relativistic setup as studied in [Alves12], [Grismayer13], [Bussmann13]. The default setup uses a pre-ionized quasi-neutral hydrogen plasma. Modifiying the ion species' mass to resample positrons instead is a test we perform regularly to control numerical heating and charge conservation.

14.4.1 References

14.5 LaserWakefield: Laser Electron Acceleration

Section author: Axel Huebl <a.huebl (at) hzdr.de>

Module author: Axel Huebl <a.huebl (at) hzdr.de>, René Widera, Heiko Burau, Richard Pausch, Marco Garten

Setup for a laser-driven electron accelerator [TajimaDawson] in the blowout regime of an underdense plasma [Modena] [PukhovMeyerterVehn]. A short (fs) laser pulse with ultra-high intensity (a_0 >> 1), modeled as a finite Gaussian pulse is focussed in a hydrogen gas target. The target is assumed to be pre-ionized with negligible temperature. The relevant area of interaction is followed by a co-moving window, in whose time span the movement of ions is considered irrelevant which allows us to exclude those from our setup.

This is a demonstration setup to get a visible result quickly and test available methods and I/O. The plasma gradients are unphysically high, the resolution of the laser wavelength is seriously bad, the laser parameters (e.g. pulse duration, focusing) are challenging to achieve technically and interaction region is too close to the boundaries of the simulation box. Nevertheless, this setup will run on a single GPU in full 3D in a few minutes, so just enjoy running it and interact with our plugins!

There exists an example how to run a parameter scan using Snakemake. Details can be found here.

14.5.1 References

14.6 TransitionRadiation : Transtion Radiation

Section author: Finn-Ole Carstens <f.carstens (at) hzdr.de>

This example simulates the coherent and incoherent transition radiation created by an electron bunch in-situ. The implemented transition radiation follows the studies from [Schroeder2004] and [Downer2018]. The transition radiation is computed for an infinitely large interface perpendicular to the y-axis of the simulation.

The electron bunch in this setup is moving with a 45° angle in the x-y plane with a Lorentz-factor of $gamma = 100$. The bunch has a Gaussian distribution with $sigma_y = 3.0 mu m$. The results can be interpreted with the according python script *lib/python/picongpu/plugins/plot_mpl/transition_radiation_visualizer.py*.

14.6.1 References

atomicPhysics: atomic Physics example for experimental FLYonPIC PIConGPU extension
was originally based on the Empty example by Axel Huebl

Section author: Brian Marre <b.marre (at) hzdr.de>

This is a minimum spec-exmaple for testing the still experimental atomic physics branch of picongpu. It uses mostly default picongpu algorithms and settings, except for a few changes listed below, to reduce computation time.

- reduced super cell size, only 2x2x2 cells form a super cell
- reduced number of particels overall, only 1 macro-ion and 1 macro-electron per super cell
- no ionization, instead initialization in a somewhat correct charge state for initial conditions

Use this as a starting point for your own atomic physics picongpu simulations, but beware this is still experimental.

CHAPTER FIFTEEN

WORKFLOWS

This section contains typical user workflows and best practices.

15.1 Adding Laser

Section author: Sergei Bastrakov

There are several alternative ways of adding an incoming laser (or any source of electromagnetic field) to a PIConGPU simulation:

1. selecting incident field profiles for respective boundaries in *incidentField.param*
2. using field or current background in *fieldBackground.param*

These ways operate independently of one another, each has its features and limitations. Incident field- and laser profiles currently match one another.

15.1.1 Incident Field

Incident field is an external field source producing a wave propagating inwards the simulation volume. The source is applied at a generating surface that is a boundary of an axis-aligned box located inside the simulation area. The implementation is based on the total field/scattered field formulation described in detail *here*. A user defines positioning of this box in the total domain in *incidentField.param*.

With properly chosen sources, the generated waves only propagate inwards the volume bounded by the generation surface and there is no considerable leakage or noise. The generating surface is otherwise transparent for all signals, such as reflections of lasers from a target. A typical setup includes a field absorber located outside of the surface (directly or at a distance).

The surface must be offset inwards relative to each boundary by at least the field absorber thickness along the boundary so that the generating surface is located in the internal area. An exception to this requirement is made for simulations using the moving window. Then the surface positions along window movement direction can be located outside of the initially simulated volume. In this case, parts of the surface located outside of the currently simulated volume are treated as if they had zero incident field and it is user's responsibility to apply a source matching such a case.

For each of the generation planes `XMin, XMax, YMin, YMax, ZMin, ZMax` (the latter two for 3d) a user sets an incident profile, or a typelist of such profiles, to be applied. In case a typelist is used, the result is a sum of all profiles in the list.

In principle, the same sources should be applied at the whole generating surface, not just at planes where the lasers enter. Then, the generated incident field will only exist in the internal volume, with application at the opposite side compensating and effectively removing it. Note that this effect is not related to having a field absorber, but a property of the total field/scattered field formulation. In practice there may be some noise due to numerical dispersion or imprecise source formulation. In this case, a user may apply sources only at the "enter" parts of the generating surface but not on the opposite side (which will then be transparent), and employ a field absorber if needed.

The configuration is done through parameter structures, depending on the profile type. Both profiles and parameter structures generally match their laser counterparts. The differences between matching incident field- and laser profiles are:

1. positioning of incident field is controlled for the generation plane and not via an internal member `::initPlaneY`
2. incident field profiles do not have an extra time delay equal to $\mathrm{initPlaneY} * \Delta y/c$ as lasers do (when needed, other parameters could be adjusted to accomodate for the delay)
3. default initial phase is chosen so that the laser starts smoothly at the generation plane (for laser it is always for plane $y = 0$)
4. incident field uses generalized coordinate system and treats transversal axes and parameters generically (explained in comments of the profile parameters in question)

Note that the profile itself only controls properties of the laser, but not where it will be applied to. It is a combination of profile and particular plane that together will produce an inward-going laser adhering to the profile. For pre-set profiles a proper orientation of the wave will be provided by internal implementation. With the `Free` profile, it is on a user to provide functors to calculate incident fields and ensure the orientation for the boundaries it is applied to (however, it does not have to work for all boundaries, only the ones in question). Please refer to *the detailed description* for setting up `Free` profile, also for the case when only one of the external fields is known in explicit form. For a laser profile with non zero field amplitudes on the transversal borders of the profile e.g. defined by the profile `Free` without a transversal envelop the trait `MakePeriodicTransversalHuygensSurfaceContiguous` must be specialized and returning true to handle field periodic boundaries correctly.

Incident field is compatible to all field solvers, however using field solvers other than Yee requires a larger offset of the generating surface from absorber depending on the stencil width along the boundary axis. As a rule of thumb, this extra requirement is (order of FDTD solver / 2 - 1) cells. Additionally, the current implementation requires the generation surface to be located sufficiently far away from local domain boundaries. The same rule of a thumb can be used, with offsets being at least that many cells away from domain boundaries. Validity of the provided offsets with respect to both conditions is checked at run time.

15.2 Boundary Conditions

Section author: Sergei Bastrakov, Lennert Sprenger

Two kinds of boundary conditions are supported: periodic and absorbing. They are set in a *.cfg file* with option `--periodic <x> <y> <z>`. Value 0 corresponds to absorbing boundaries along the axis (used by default), 1 corresponds to periodic. The same boundary condition kind is applied for all particles species and fields, on both sides.

15.2.1 Particles

By default, boundary kinds match the value of `--periodic` and so are either periodic or absorbing. For species with a particle pusher, it can be overridden with option *–<prefix>_boundary <x> <y> <z>*. The supported boundary kinds are: periodic, absorbing, reflecting, and thermal.

Currently only the following combinations of field and particle boundaries are supported. When fields are periodic along an axis, boundaries for all species must be periodic along this axis. When fields are absorbing (non-periodic), species must be absorbing, reflecting, or thermal.

By default, the particle boundaries are applied at the global domain boundaries. A user can change the boundary application area by setting an offset with the option *–<prefix>_boundaryOffset <x> <y> <z>*. The *boundaryOffset* is in terms of whole cells, so integers are expected. It sets an offset inwards from the global domain boundary. Periodic boundaries only allow 0 offset, thermal boundaries require a positive offset, and other kinds support non-negative offsets.

Boundary temperature for thermal boundaries, in keV, is set with option *–<prefix>_boundaryTemperature <x> <y> <z>*.

For example, reflecting and thermal boundary conditions for species *e* are configured by *–e_boundary reflecting thermal reflecting –e_boundaryOffset 0 1 10 –e_boundaryTemperature 0.0 20.0 0.0*

Particles are not allowed to be outside the boundaries for the respective species. (For the periodic case, there are no boundaries in that sense.) After species are initialized, all outer particles will be deleted. During the simulation, the crossing particles will be handled by boundary condition implementations and moved or deleted as a result.

The internal treatment of particles in the guard area is controlled by the `boundaryCondition` flag in *speciesDefinition.param*. However, this option is for expert users and generally should not be modified. To set physical boundary conditions, use the command-line option described above.

15.2.2 Fields

Periodic boundary conditions for fields do not allow customization or variants. The rest of the section concerns absorbing boundaries.

For the absorbing boundaries, there is a virtual field absorber layer inside the global simulation area near its boundaries. Field values in the layer are treated differently from the rest, by a combination of a field solver and a field absorber. It causes field dynamics inside the absorber layer to differ from that in a vacuum. Otherwise, the affected cells are a normal part of a simulation in terms of indexing and output. Note that particle absorption happens at the external surface of the field absorber layer, matching the global simulation area border.

The field absorber mechanism and user-controlled parameters depend on the field absorber kind enabled. It is controlled by command-line option `--fieldAbsorber`. For all absorber kinds, the parameters are controlled by *fieldAbsorber.param*.

By default, the Perfectly Matched Layer (PML) absorber is used. For this absorber, thickness of 8 to 12 cells is recommended. Other absorber parameters can generally be used with default values. PML generally provides much better absorber qualities than the exponential damping absorber.

In case PML field absorber is used together with absorbing particle boundaries, a special damping is applied for current density values in the PML area. This treatment is to smooth effects of charges leaving the simulation volume and thus to better represent open boundaries.

For the exponential absorber, thickness of about 32 cells is recommended.

15.3 Setting the Number of Cells

Section author: Axel Huebl

Together with the grid resolution in *grid.param*, the number of cells in our *.cfg files* determine the overall size of a simulation (box). The following rules need to be applied when setting the number of cells:

Each device needs to:

1. contain an integer *multiple* of supercells
2. at least *three* supercells
3. for non periodic boundary conditions, the number of absorbing boundary cells for devices at the simulation boundary (see *grid.param*) must fit into the local volume

The grid size will be automatically adjusted if the conditions above are not fulfilled. This behavior can be disabled by using the command line option `--autoAdjustGrid off`

Supercell sizes in terms of number of cells are set in *memory.param* and are by default `8x8x4` for 3D3V simulations on GPUs. For 2D3V simulations, `16x16` is usually a good supercell size, however the default is simply cropped to `8x8`, so make sure to change it to get more performance.

15.4 Changing the Resolution with a Fixed Target

Section author: Axel Huebl

One often wants to refine an already existing resolution in order to model a setup more precisely or to be able to model a higher density.

1. change cell sizes and time step in *grid.param*
2. change number of GPUs in *.cfg file*
3. change number of *number of cells and distribution over GPUs* in *.cfg file*
4. adjust (transveral) positioning of targets in *density.param*
5. *recompile*

15.5 Calculating the Memory Requirement per Device

Section author: Marco Garten

The planning of simulations for realistically sized problems requires a careful estimation of memory usage and is often a trade-off between resolution of the plasma, overall box size and the available resources. The file *memory_calculator.py* contains a class for this purpose.

The following paragraph shows the use of the `MemoryCalculator` for the `4.cfg` setup of the *FoilLCT example* example.

It is an estimate for how much memory is used per device if the whole target would be fully ionized but does not move much. Of course, the real memory usage depends on the case and the dynamics inside the simulation. We calculate the memory of just one device per row of GPUs in laser propagation direction. We hereby assume that particles are distributed equally in the transverse direction like it is set up in the FoilLCT example.

We encourage to try out this script with different settings, to see how they influence the distribution of the total memory requirement between devices.

```python
from picongpu.extra.utils import MemoryCalculator
from math import ceil


cell_size = 0.8e-6 / 384.0  # 2.083e-9 m
y0 = 0.5e-6  # position of foil front surface (m)
y1 = 1.5e-6  # position of foil rear surface (m)
L = 10e-9  # pre-plasma scale length (m)
L_cutoff = 4 * L  # pre-plasma length (m)

sim_dim = 2
# number of cells in the simulation
Nx_all, Ny_all, Nz_all = 256, 1280, 1
# number of GPU rows in each direction
x_rows, y_rows, z_rows = 2, 2, 1
# number of cells per GPU
Nx, Ny, Nz = Nx_all / x_rows, Ny_all / y_rows, Nz_all / z_rows

vacuum_cells = ceil((y0 - L_cutoff) / cell_size)  # in front of the target
# target cells (between surfaces + pre-plasma)
target_cells = ceil((y1 - y0 + 2 * L_cutoff) / cell_size)
# number of cells (y direction) on each GPU row
GPU_rows = [0] * y_rows
cells_to_spread = vacuum_cells + target_cells
```

(continues on next page)

(continued from previous page)

```
# spread the cells on the GPUs
for ii, _ in enumerate(GPU_rows):
    if cells_to_spread >= Ny:
        GPU_rows[ii] = Ny
        cells_to_spread -= Ny
    else:
        GPU_rows[ii] = cells_to_spread
        break
# remove vacuum cells from the front rows
extra_cells = vacuum_cells
for ii, _ in enumerate(GPU_rows):
    if extra_cells >= Ny:
        GPU_rows[ii] = 0
        extra_cells -= Ny
    else:
        GPU_rows[ii] -= extra_cells
        break

pmc = MemoryCalculator(Nx, Ny, Nz)

# typical number of particles per cell which is multiplied later for
# each species and its relative number of particles
N_PPC = 6
# conversion factor to megabyte
megabyte = 1.0 / (1024 * 1024)

target_x = Nx  # full transverse dimension of the GPU
target_z = Nz


def sx(n):
    return {1: "st", 2: "nd", 3: "rd"}.get(n if n < 20 else int(str(n)[-1]), "th")


for row, target_y in enumerate(GPU_rows):
    print("{}{} row of GPUs:".format(row + 1, sx(row + 1)))
    print("* Memory requirement per GPU:")
    # field memory per GPU
    field_gpu = pmc.mem_req_by_fields(Nx, Ny, Nz, field_tmp_slots=2, particle_shape_
↪order=2, sim_dim=sim_dim)
    print(" + fields: {:.2f} MB".format(field_gpu * megabyte))

    # electron macroparticles per supercell
    e_PPC = N_PPC * (
        # H,C,N pre-ionization - higher weighting electrons
        3
        # electrons created from C ionization
        + (6 - 2)
        # electrons created from N ionization
        + (7 - 2)
    )
    # particle memory per GPU - only the target area contributes here
    e_gpu = pmc.mem_req_by_particles(
        target_x,
        target_y,
        target_z,
```

(continues on next page)

(continued from previous page)

```
        num_additional_attributes=0,
        particles_per_cell=e_PPC,
    )
    H_gpu = pmc.mem_req_by_particles(
        target_x,
        target_y,
        target_z,
        # no bound electrons since H is pre-ionized
        num_additional_attributes=0,
        particles_per_cell=N_PPC,
    )
    C_gpu = pmc.mem_req_by_particles(
        target_x,
        target_y,
        target_z,
        num_additional_attributes=1,
        particles_per_cell=N_PPC,  # number of bound electrons
    )
    N_gpu = pmc.mem_req_by_particles(
        target_x,
        target_y,
        target_z,
        num_additional_attributes=1,
        particles_per_cell=N_PPC,
    )
    # memory for calorimeters
    cal_gpu = pmc.mem_req_by_calorimeter(n_energy=1024, n_yaw=360, n_pitch=1) * 2  #␣
↪electrons and protons
    # memory for random number generator states
    rng_gpu = pmc.mem_req_by_rng(Nx, Ny, Nz)

    print(" + species:")
    print("  - e: {:.2f} MB".format(e_gpu * megabyte))
    print("  - H: {:.2f} MB".format(H_gpu * megabyte))
    print("  - C: {:.2f} MB".format(C_gpu * megabyte))
    print("  - N: {:.2f} MB".format(N_gpu * megabyte))
    print(" + RNG states: {:.2f} MB".format(rng_gpu * megabyte))
    print(" + particle calorimeters: {:.2f} MB".format(cal_gpu * megabyte))

    mem_sum = field_gpu + e_gpu + H_gpu + C_gpu + N_gpu + rng_gpu + cal_gpu
    print("* Sum of required GPU memory: {:.2f} MB".format(mem_sum * megabyte))
```

This will give the following output:

```
1st row of GPUs:
* Memory requirement per GPU:
 + fields: 4.58 MB
 + species:
  - e: 114.16 MB
  - H: 9.51 MB
  - C: 10.74 MB
  - N: 10.74 MB
 + RNG states: 1.88 MB
 + particle calorimeters: 5.62 MB
* Sum of required GPU memory: 157.23 MB
2nd row of GPUs:
```

(continues on next page)

(continued from previous page)

```
* Memory requirement per GPU:
 + fields: 4.58 MB
 + species:
  - e: 27.25 MB
  - H: 2.27 MB
  - C: 2.56 MB
  - N: 2.56 MB
 + RNG states: 1.88 MB
 + particle calorimeters: 5.62 MB
* Sum of required GPU memory: 46.72 MB
```

If you have a machine or cluster node with NVIDIA GPUs you can find out the available memory size by typing `nvidia-smi` on a shell.

15.6 Definition of Composite Materials

Section author: Axel Huebl

The easiest way to define a composite material in PIConGPU is starting relative to an idealized full-ionized electron density. As an example, lets use $C_{21}H_{25}N$ (*"8CB"*) with a plasma density of $n_{\text{e,max}} = 192\, n_{\text{c}}$ contributed by the individual ions relatively as:

- Carbon: $21 \cdot 6 / N_{\Sigma\text{e-}}$
- Hydrogen: $25 \cdot 1 / N_{\Sigma\text{e-}}$
- Nitrogen: $1 \cdot 7 / N_{\Sigma\text{e-}}$

and $N_{\Sigma\text{e-}} = 21_{\text{C}} \cdot 6_{\text{C}^{6+}} + 25_{\text{H}} \cdot 1_{\text{H}^{+}} + 1_{\text{N}} \cdot 7_{\text{N}^{7+}} = 158$.

Set the idealized electron density in *density.param* as a reference and each species' relative `densityRatio` from the list above accordingly in *speciesDefinition.param* (see the input files in the *FoilLCT example* for details).

In order to initialize the electro-magnetic fields self-consistently, read *quasi-neutral initialization*.

15.7 Quasi-Neutral Initialization

Section author: Axel Huebl

In order to initialize the electro-magnetic fields self-consistently, one needs to fulfill Gauss's law $\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon_0}$ (and $\vec{\nabla} \cdot \vec{B} = 0$). The trivial solution to this equation is to start *field neutral* by microscopically placing a charge-compensating amount of free electrons on the same position as according ions.

15.7.1 Fully Ionized Ions

For fully ionized ions, just use `ManipulateDerive` in *speciesInitialization.param* and derive macro-electrons $1 : 1$ from macro-ions but increase their weighting by $1 : Z$ of the ion.

```
using InitPipeline = pmacc::mp_list<
    /* density profile from density.param and
     *     start position from particle.param */
    CreateDensity<
        densityProfiles::YourSelectedProfile,
        startPosition::YourStartPosition,
        Carbon
    >,
```

(continues on next page)

(continued from previous page)

```
    /* create a macro electron for each macro carbon but increase its
     *     weighting by the ion's proton number so it represents all its
     *     electrons after an instantanous ionization */
    ManipulateDerive<
        manipulators::ProtonTimesWeighting,
        Carbon,
        Electrons
    >
>;
```

If the `Carbon` species in this example has an attribute `boundElectrons` (optional, see *speciesAttributes.param and speciesDefinition.param*) and its value is not manipulated the default value is used (zero bound electrons, fully ionized). If the attribute `boundElectrons` is not added to the `Carbon` species the charge state is considered constant and taken from the `chargeRatio< ... >` particle flag.

15.7.2 Partly Ionized Ions

For partial pre-ionization, the *FoilLCT example* shows a detailed setup. First, define a functor that manipulates the number of bound electrons in *particle.param*, e.g. to *twice pre-ionized*.

```
#include "picongpu/particles/traits/GetAtomicNumbers.hpp"
// ...

namespace manipulators
{
    //! ionize ions twice
    struct TwiceIonizedImpl
    {
        template< typename T_Particle >
        DINLINE void operator()(
            T_Particle& particle
        )
        {
            constexpr float_X protonNumber =
                GetAtomicNumbers< T_Particle >::type::numberOfProtons;
            particle[ boundElectrons_ ] = protonNumber - float_X( 2. );
        }
    };

    //! definition of TwiceIonizedImpl manipulator
    using TwiceIonized = generic::Free< TwiceIonizedImpl >;

} // namespace manipulators
```

Then again in *speciesInitialization.param* set your initialization routines to:

```
using InitPipeline = pmacc::mp_list<
    /* density profile from density.param and
     *     start position from particle.param */
    CreateDensity<
        densityProfiles::YourSelectedProfile,
        startPosition::YourStartPosition,
        Carbon
    >,
    /* partially pre-ionize the carbons by manipulating the carbon's
     *     `boundElectrons` attribute,
```

(continues on next page)

(continued from previous page)

```
     *     functor defined in particle.param: set to C2+ */
    Manipulate<
        manipulators::TwiceIonized,
        Carbon
    >,
    /* does also manipulate the weighting x2 while deriving the electrons
     *     ("twice pre-ionized") since we set carbon as C2+ */
    ManipulateDerive<
        manipulators::binary::UnboundElectronsTimesWeighting,
        Carbon,
        Electrons
    >
>;
```

15.8 Probe Particles

Section author: Axel Huebl

Probe particles ("probes") can be used to record field quantities at selected positions over time.

As a geometric data-reduction technique, analyzing the discrete, regular field of a particle-in-cell simulation only at selected points over time can greatly reduce the need for I/O. Such particles are often arranged at isolated points, regularly as along lines, in planes or in any other user-defined manner.

Probe particles are usually neutral, non-interacting test particles that are statically placed in the simulation or co-moving with along pre-defined path. Self-consistently interacting particles are usually called *tracer particles*.

15.8.1 Workflow

- `speciesDefinition.param`: create a stationary probe species, add `probeE` and `probeB` attributes to it for storing interpolated fields

```
using ParticleFlagsProbes = MakeSeq_t<
    particlePusher< particles::pusher::Probe >,
    shape< UsedParticleShape >,
    interpolation< UsedField2Particle >
>;

using Probes = Particles<
    PMACC_CSTRING( "probe" ),
    ParticleFlagsProbes,
    MakeSeq_t<
        position< position_pic >,
        probeB,
        probeE
    >
>;
```

and add it to `VectorAllSpecies`:

```
using VectorAllSpecies = MakeSeq_t<
    Probes,
    // ...
>;
```

- `density.param`: select in which cell a probe particle shall be placed, e.g. in each 4th cell per direction:

```
// put probe particles every 4th cell in X, Y(, Z)
using ProbeEveryFourthCell = EveryNthCellImpl<
    mCT::UInt32<
        4,
        4,
        4
    >
>;
```

- `particle.param`: initialize the individual probe particles in-cell, e.g. always in the left-lower corner and only one per selected cell

```
namespace startPosition
{
    //! Configuration of initial in-cell particle position
    struct OnePositionParameter
    {
        /** Maximum number of macro-particles per cell during density profile␣
↪evaluation.
         *
         * Determines the weighting of a macro particle as well as the number of
         * macro-particles which sample the evolution of the particle distribution
         * function in phase space.
         *
         * unit: none
         */
        static constexpr uint32_t numParticlesPerCell = <PARTICLES_PER_CELL_TO_SPAWN>;

        /** each x, y, z in-cell position component in range [0.0, 1.0)
         *
         * @details in 2D the last component is ignored
         */
        static constexpr auto inCellOffset = float3_X(0., 0., 0.);
    };

    /** Definition of OnePosition start position functor that
     * places macro-particles at the initial in-cell position defined above.
     */
    using OnePosition = OnePositionImpl<OnePositionParameter>;
} // namespace startPosition
```

- `speciesInitialization.param`: initialize particles for the probe just as with regular particles

```
using InitPipeline = pmacc::mp_list<
    // ... ,
    CreateDensity<
        densityProfiles::ProbeEveryFourthCell,
        startPosition::OnePosition,
        Probes
    >
>;
```

- `fileOutput.param`: make sure the the probe particles are part of `FileOutputParticles`

```
// either all via VectorAllSpecies or just select
using FileOutputParticles = MakeSeq_t< Probes >;
```

15.8.2 Known Limitations

Note: currently, only the electric field $\vec{E}$ and the magnetic field $\vec{B}$ can be recorded

Note: we currently do not support time averaging

Warning: If the probe particles are dumped in the file output, the instantaneous fields they recorded will be one time step behind the last field update (since our runOneStep pushed the particles first and then calls the field solver). Adding the attributes `probeE` or `probeB` to a species will increase the particle memory footprint only for the corresponding species by `3 * sizeof(float_X)` byte per attribute and particle.

15.9 Tracer Particles

Section author: Axel Huebl

Tracer particles are like *probe particles*, but interact self-consistenly with the simulation. They are usually used to visualize *representative* particle trajectories of a larger distribution.

In PIConGPU each species can be a tracer particle species, which allows tracking fields along the particle trajectory.

15.9.1 Workflow

- `speciesDefinition.param`:

Add the particle attribute `particleId` to your species to give each particle a unique id. The id is optional and only required if the particle should be tracked over time. Adding `probeE` creates a tracer species stores the interpolated electric field seen by the tracer particle.

```
using ParticleFlagsTracerElectrons = MakeSeq_t<
    particlePusher< particles::pusher::Boris >,
    shape< UsedParticleShape >,
    interpolation< UsedField2Particle >,
    currentSolver::Esirkepov<UsedParticleCurrentSolver>
>;

using TracerElectron = Particles<
    PMACC_CSTRING( "e_tracer" ),
    ParticleFlagsTracerElectrons,
    MakeSeq_t<
        position< position_pic >,
        momentum,
        weighting,
        particleId,
        probeE
    >
>;
```

and add it to `VectorAllSpecies`:

```
using VectorAllSpecies = MakeSeq_t<
    TracerElectron,
```

(continues on next page)

(continued from previous page)

```
    // ...
>;
```

- create tracer particles by either
  - `speciesInitialization.param`: initializing a low percentage of your initial density inside this species or
  - `speciesInitialization.param`: assigning the target (electron) species of an ion's ionization routine to the tracer species or
  - `speciesInitialization.param`: moving some particles of an already initialized species to the tracer species (upcoming)
- `fileOutput.param`: make sure the the tracer particles are part of `FileOutputParticles`

```
// either all via VectorAllSpecies or just select
using FileOutputParticles = MakeSeq_t< TracerElectron >;
```

The electron tracer species is equivalent to normal electrons, the initial density distribution can be configured in *speciesInitialization.param*.

15.9.2 Known Limitations

- currently, only the electric field $\vec{E}$ and the magnetic field $\vec{B}$ can be recorded
- we currently do not support time averaging

15.10 Particle Filters

Section author: Axel Huebl, Sergei Bastrakov

A common task in both modeling, initializing and in situ processing (output) is the selection of particles of a particle species by attributes. PIConGPU implements such selections as *particle filters*.

Particle filters are simple mappings assigning each particle of a species either `true` or `false` (ignore / filter out). These filters can be defined in *particleFilters.param*.

15.10.1 Example

Let us select particles with momentum vector within a cone with an opening angle of five degrees (pinhole):

```
namespace picongpu
{
namespace particles
{
namespace filter
{
    struct FunctorParticlesForwardPinhole
    {
        static constexpr char const * name = "forwardPinhole";

        template< typename T_Particle >
        HDINLINE bool operator()(
            T_Particle const & particle
        )
```

(continues on next page)

(continued from previous page)

```
        {
            bool result = false;
            float3_X const mom = particle[ momentum_ ];
            float_X const normMom = pmacc::math::l2norm( mom );

            if( normMom > float_X( 0. ) )
            {
                /* place detector in y direction, "infinite distance" to target,
                 * and five degree opening angle
                 */
                constexpr float_X openingAngle = 5.0 * PI / 180.;
                float_X const dotP = mom.y() / normMom;
                float_X const degForw = math::acos( dotP );

                if( math::abs( degForw ) <= openingAngle * float_X( 0.5 ) )
                    result = true;
            }
            return result;
        }
    };
    using ParticlesForwardPinhole = generic::Free<
       FunctorParticlesForwardPinhole
    >;
```

and add `ParticlesForwardPinhole` to the `AllParticleFilters` list:

```
    using AllParticleFilters = MakeSeq_t<
        All,
        ParticlesForwardPinhole
    >;

} // namespace filter
} // namespace particles
} // namespace picongpu
```

15.10.2 Filtering Based on Global Position

A particle only stores its relative position inside the cell. Thus, with a simple filter like one shown above, it is not possible to get global position of a particle. However, there are helper wrapper filters that provide such information in addition to the particle data.

For a special case of slicing along one axis this is a simple existing filter that only needs to be parametrized:

```
namespace picongpu
{
namespace particles
{
namespace filter
{
    namespace detail
    {
        //! Parameters to be used with RelativeGlobalDomainPosition, change the␣
↪values inside
        struct SliceParam
        {
            // Lower bound in relative coordinates: global domain is [0.0, 1.0]
```

(continues on next page)

(continued from previous page)

```
            static constexpr float_X lowerBound = 0.55_X;

            // Upper bound in relative coordinates
            static constexpr float_X upperBound = 0.6_X;

            // Axis: x = 0; y= 1; z = 2
            static constexpr uint32_t dimension = 0;

            // Text name of the filter, will be used in .cfg file
            static constexpr char const* name = "slice";
        };

        //! Use the existing RelativeGlobalDomainPosition filter with our parameters
        using Slice = RelativeGlobalDomainPosition<SliceParam>;
    }
```

and add `detail::Slice` to the `AllParticleFilters` list:

```
    using AllParticleFilters = MakeSeq_t<
        All,
        detail::Slice
    >;

} // namespace filter
} // namespace particles
} // namespace picongpu
```

For a more general case of filtering based on cell index (possibly combined with other particle properties) use the following pattern:

```
namespace picongpu
{
namespace particles
{
namespace filter
{
    namespace detail
    {
        struct AreaFilter
        {
            static constexpr char const* name = "areaFilter";

            template<typename T_Particle>
            HDINLINE bool operator()(
                DataSpace<simDim> const totalCellOffset,
                T_Particle const & particle
            )
            {
                /* Here totalCellOffset is the cell index of the particle in the␣
↪total coordinate system.
                 * So we can define conditions based on both cell index and other␣
↪particle data.
                 */
                return (totalCellOffset.x() >= 10) && (particle[momentum_].x() < 0.0_
↪X);
             }
         };
```

(continues on next page)

(continued from previous page)

```
        //! Wrap AreaFilter so that it fits the general filter interface
        using Area = generic::FreeTotalCellOffset<AreaFilter>;
    }
```

and add `detail::Area` to the `AllParticleFilters` list:

```
    using AllParticleFilters = MakeSeq_t<
        All,
        detail::Area
    >;

} // namespace filter
} // namespace particles
} // namespace picongpu
```

15.10.3 Limiting Filters to Eligible Species

Besides *the list of pre-defined filters* with parametrization, users can also define generic, "free" implementations as shown above. All filters are added to `AllParticleFilters` and then *combined with all available species* from `VectorAllSpecies` (see *speciesDefinition.param*).

In the case of user-defined free filters we can now check if each species in `VectorAllSpecies` fulfills the requirements of the filter. That means: if one accesses specific *attributes* or *flags* of a species in a filter, they must exist or will lead to a compile error.

As an example, *probe particles* usually do not need a `momentum` attribute which would be used for an energy filter. So they should be ignored from compilation when combining filters with particle species.

In order to exclude all species that have no `momentum` attribute from the `ParticlesForwardPinhole` filter, specialize the C++ trait `SpeciesEligibleForSolver`. This trait is implemented to be checked during compile time when combining filters with species:

```
// ...

} // namespace filter

namespace traits
{
    template<
        typename T_Species
    >
    struct SpeciesEligibleForSolver<
        T_Species,
        filter::ParticlesForwardPinhole
    >
    {
        using type = typename pmacc::traits::HasIdentifiers<
            typename T_Species::FrameType,
            MakeSeq_t< momentum >
        >::type;
    };
} // namespace traits
} // namespace particles
} // namespace picongpu
```

CHAPTER

SIXTEEN

CROSS-COMPILE FOR RISC-V

This section contains information on how to cross-compile for RISC-V.

This section assumes you have a x86 system with a compiled gnu/clang compiler which can target RISC-V. A detailed description on how to setup can be find under RISC-V-Toolchain.

You must compile all dependencies static only! We observed problems with the CMake MPI detection therefore we disable the CMake MPI search with *-DMPI_CXX_WORKS=ON*` and provide all required compiler and linker parameters via the environment variable `CXXFLAGS`, `CFLAGS` and `LDFLAGS`.

```
# set MPI_ROOT to the root MPI directory
export LDFLAGS="-L$MPI_ROOT/lib -lmpi -lompitrace -lopen-rte -lopen-pal -ldl -
↪lpthread -lutil -lrt"
# compile for 64bit RISC-V with double floating point support
export CXXFLAGS="-I$MPI_ROOT/include -pthread -march=rv64gc -mabi=lp64d"
export CFLAGS=$CXXFLAGS
```

To be able to cross compile you should point to the CMake toolchain file shipped together with PIConGPU. Depending on your environment, please select the Clang or GNU toolchain. The execution backend is provided in this example explicitly. We recommend to write a *profile* for the system. Depending on the toolchain the environment variable `RISCV_GNU_INSTALL_ROOT` or `RISCV_CLANG_INSTALL_ROOT` should be provided.

```
export RISCV_GNU_INSTALL_ROOT="$(dirname $(which riscv64-unknown-linux-gnu-gcc))/.."
export RISCV_CLANG_INSTALL_ROOT="$(dirname $(which clang))/.."
```

```
# ``omp2b`` for OpenMP and ``serial`` for serial execution on one core
pic-build -b omp2b -c"-DMPI_CXX_WORKS=ON  -DCMAKE_TOOLCHAIN_FILE=<PATH_TO_PICONGPU_
↪SRC>/cmake/riscv64-gnu.toolchain.cmake"
```

After PIConGPU is compiled you can interactively join to the RISC-V compute node and test if you compiled for the right target architecture `./bin/picongpu --help`. After this short test you should follow the typical workflow and start your simulation via *TBG*.

CHAPTER

SEVENTEEN

DUMPING METADATA

Starting your simulation with

```
${EXECUTABLE} ${ARGS} --dump-metadata "${OPTIONAL_FILENAME}" --no-start-simulation
```

will dump a json respresentation of some metadata to *${OPTIONAL_FILENAME}*. If no *${OPTIONAL_FILENAME}* is given, the default value

```
"picongpu-metadata.json"
```

is used. This feature might in a future revision default to being active.

You might want to dump metadata without actually running a simulation. In this case, you can add the *–no-start-simulation* flag which will make the code skip the actual simulation. If your intention is instead to also run the simulation afterwards, just leave it out and the program will proceed as normal.

The dumping happens after all initialisation work is done immediately before the simulation starts (or is skipped). This implies that

- No dynamic information about the simulation can be included (e.g. information about the state at time step 10).
- The dumped information will represent the actual parameters used. This implies that the parameters reported by this feature will differ from the ones provided on the commandline if, e.g., automatic adjustment of the grid size (see *autoAdjustGrid*) kicks in.

Note: Since we are still performing all initialisation work in order to get to the actual parameter values that affect the simulation, it might be necessary to run this feature in a (very short) full batch submission with all resources (like GPUs, etc.) available as for the full simulation run even when running with *–no-start-simulation*.

The content of the output is a **summary of the physical content of the simulated conditions** and the format is described *below*. It is important to note that the structure of the content is aligned with its categorisation in the physical context and not (enforced to be) aligned with the internal code structure.

Note: The scope of this feature is to provide a human- and machine-readable **summary of the physical content of the simulated conditions**. The envisioned use cases are:

- a theoretician quickly getting an overview over their simulation data,
- an experimentalist comparing with simulation data or
- a database using such information for tagging, filtering and searching.

The following related aspects are out of scope (for the PIConGPU development team):

- Reproducibility: The only faithful, feature-complete representation of the input necessary to reproduce a PIConGPU simulation is the complete input directory. If a more standardised and human-readable repesentation is desired, PICMI provides access to a small subset of features.

- Completeness: This feature is intended to be fed with well-structured information considered important by the researchers. It is *customisable* but the design does not allow to ensure any form of completeness with appropriate maintenance effort. We therefore do not aim to describe simulations exhaustively.
- (De-)Serialisation: We do not provide infrastructure to fully or partially reconstruct C++ objects from the dumped information.
- Standardisation or generalisation of the format: The format and content are the result of our best effort to be useful. Any form of standardisation or generalisation beyond this scope requires a resource commitment from outside the PIConGPU development team. We are happy to implement any such result.

17.1 The Format

The created file is a human-readable text file containing valid *json* the content of which is partially *customisable*. We do not enforce a particular format but suggest that you stick as closely as possible to the naming conventions from *PyPIConGPU* and PICMI. For example, the *LaserWakefield* example dumps the following metadata which might be supplemented with further details as appropriate for the described elements of the simulation:

```json
{
  "incidentField": {
    "XMax": [
      {}
    ],
    "XMin": [
      {}
    ],
    "YMax": [
      {}
    ],
    "YMin": [
      {
        "polarisation": {
          "direction": [
            1.0,
            0.0,
            0.0
          ],
          "type": "circular"
        }
      }
    ],
    "ZMax": [
      {}
    ],
    "ZMin": [
      {}
    ]
  },
  "simulation": {
    "steps": 0
  }
}
```

17.2 Customisation

17.2.1 Content Creation

The main customisation point for adding and adjusting the output related to some *typename TObject*, say a Laser or the *Simulation* object itself, is providing a specialisation for *picongpu::traits::GetMetadata* defaulting to

```
template<typename TObject>
struct GetMetadata<TObject, std::enable_if_t<providesMetadataAtRT<TObject>>>
{
    // Holds a constant reference to the RT instance it's supposed to report about.
    // Omit this for the CT specialisation!
    TObject const& obj;

    nlohmann::json description() const
    {
        return obj.metadata();
    }
};

template<typename TObject>
struct GetMetadata<TObject, std::enable_if_t<providesMetadataAtCT<TObject>>>
{
    // CT version has no members. Apart from that, the interface is identical to the␣
↪RT version.

    nlohmann::json description() const
    {
        return TObject::metadata();
    }
};
```

For example, customising the metadata for *MyClass* with some runtime (RT) as well as some compiletime (CT) information could look something like this

```
template<>
struct picongpu::traits::GetMetadata<MyClass>
{
    MyClass const& obj;

    json description() const
    {
        json result = json::object(); // always use objects and not arrays as root
        result["my"]["cool"]["runtimeValue"] = obj.runtimeValue;
        result["my"]["cool"]["compiletimeValue"] =␣
↪MyClass::MyCompileTimeInformation::value;
        result["somethingElseThatSeemedImportant"] = "not necessarily derived from␣
↪obj or MyClass";
        return result;
    }
};
```

This can be put anywhere in the code where *MyClass* is known, e.g., in a pertinent *.param* file or directly below the declaration of *MyClass* itself.

The *json* object returned from *description()* is related to the final output via a merge_patch operation but we do not guarantee any particular order in which these are merged. So it is effectively the responsibility of the programmer to make sure that no metadata entries overwrite each other.

17.2.2 Tackling Acess Restrictions

These external classes might run into access restrictions when attempting to dump private or protected members. These can be circumvented in three ways:

1. If *MyClass* already implements a *.metadata()* method, it might already provide the necessary information through that interface, e.g.

   ```
   template<>
   struct picongpu::traits::GetMetadata<SomethingWithPrivateInfo>
   {
       SomethingWithPrivateInfo const& obj;

       json description() const
       {
           auto result = obj.metadata();
           // could also depend on the (publicly accessible members of) `obj`:
           result["customisedInfo"] = "Some customised string.";
           return result;
       }
   };
   ```

   This is the preferred way of handling this situation (if applicable). The default implementation of *picongpu::traits::GetMetadata* forwards to such *.metadata()* methods anyway.

2. Declare *picongpu::traits::GetMetadata<MyClass>* a friend of *MyClass*, e.g.

   ```
   struct SomethingWithoutUsefulMetadata : SomethingWithPrivateInfo
   {
       friend picongpu::traits::GetMetadata<SomethingWithoutUsefulMetadata>;
       // ...
   ```

   This way is minimally invasive and preferred if your change is only applicable to your personal situation and is not intended to land into mainline.

3. Implement/adjust the *.metadata()* member function of *MyClass*, e.g.

   ```
   struct SomethingWithRTInfo
   {
       int info = 0;

       json metadata() const
       {
           auto result = json::object();
           result["info"] = info;
           return result;
       }
   };
   ```

   This method is preferred if your change is general enough to make it into the mainline. If so, you are invited to open a pull request (see CONTRIBUTING.md for more details). It is also the approach used to provide you with default implementations to build upon.

17.2.3 Content Registration

If you are not only adjusting existing output but instead you are adding metadata to a class that did not report any in the past, this class must register itself **before the simulation starts**. Anything that experiences some form of initialisation at runtime, e.g., *plugins* should register themselves after their initialisation. To stick with the example, a plugin could add

```
addMetadataOf(*this);
```

at the end of its *pluginLoad()* method (see the Simulation class or an example).

Classes that only affect compiletime aspects of the program need to be registered in *include/picongpu/param/metadata.param* by extending the compiletime list *MetadataRegisteredAtCT*. Remember: Their specialisation of *picongpu::traits::GetMetadata* does not hold a reference.

Classes that get instantiated within a running simulation (and not in the initialisation phase) cannot be included (because they are dynamic information, see above) unless their exact state could be forseen at compile time in which case they can be handled exactly as compiletime-only classes.

17.2.4 Metadata Handling Via Policies

It is sometimes convenient to have different strategies for handling metadata at hand which can be applied to independent of the exact content. Despite them not being formal entities in the code, an approach via *policies* can come in handy. For example, the *AllowMissingMetadata* policy can wrap any CT type in order to handle cases when no metadata is available. This is its implementation:

```
template<typename TObject>
struct AllowMissingMetadata
{
    // it's probably a nice touch to provide this, so people don't need a lot of
    // template metaprogramming to get `TObject`
    using type = TObject;
};

template<typename TObject>
struct GetMetadata<AllowMissingMetadata<TObject>> : GetMetadata<TObject>
{
    nlohmann::json description() const
    {
        return handle(GetMetadata<TObject>::description());
    }

    static nlohmann::json handle(nlohmann::json const& result)
    {
        // okay, we've found metadata, so we return it
        return result;
    }

    static nlohmann::json handle(detail::ReturnTypeFromDefault<TObject> const& result)
    {
        // also okay, we couldn't find metadata, so we'll return an empty object
        return nlohmann::json::object();
    }
};
```

Another example is the categorisation of different incident fields which – by themselves – cannot report from which direction they are incident (see the GetMetadata trait). The *IncidentFieldPolicy* provides a strategy to gather all pertinent metadata and assemble the *incidentField* subtree of the output by providing the necessary context.

17.2.5 Handling Custom Types

The nlohmann-json library in use allows to serialise arbitrary types as described here. As an example, we have implemented a serialisation for *pmacc::math::Vector* in GetMetadata.

CHAPTER

EIGHTEEN

THE PARTICLE-IN-CELL ALGORITHM

Section author: Axel Huebl, Klaus Steiniger

Please also refer to the textbooks [BirdsallLangdon], [HockneyEastwood], our *latest paper on PIConGPU* and the works in [Huebl2014] and [Huebl2019] .

18.1 System of Equations

$$
\begin{aligned}
\nabla \cdot \mathbf{E} &= \frac{1}{\varepsilon_0}\sum_s \rho_s \\
\nabla \cdot \mathbf{B} &= 0 \\
\nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\
\nabla \times \mathbf{B} &= \mu_0\left(\sum_s \mathbf{J}_s + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t}\right)
\end{aligned}
$$

for multiple particle species s. $\mathbf{E}(t)$ represents the electric, $\mathbf{B}(t)$ the magnetic, ρ_s the charge density and $\mathbf{J}_s(t)$ the current density field.

Except for normalization of constants, PIConGPU implements the governing equations in SI units.

18.2 Relativistic Plasma Physics

The 3D3V particle-in-cell method is used to describe many-body systems such as a plasmas. It approximates the Vlasov–Maxwell–Equation

$$\partial_t f_s(\mathbf{x},\mathbf{v},t) + \mathbf{v} \cdot \nabla_x f_s(\mathbf{x},\mathbf{v},t) + \frac{q_s}{m_s}\left[\mathbf{E}(\mathbf{x},t) + \mathbf{v} \times \mathbf{B}(\mathbf{x},t)\right] \cdot \nabla_v f_s(\mathbf{x},\mathbf{v},t) = 0 \tag{18.1}$$

with f_s as the distribution function of a particle species s, $\mathbf{x}, \mathbf{v}, t$ as position, velocity and time and $\frac{q_s}{m_s}$ the charge to mass-ratio of a species. The momentum is related to the velocity by $\mathbf{p} = \gamma m_s \mathbf{v}$.

The equations of motion are given by the Lorentz force as

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{V_s}(t) &= \frac{q_s}{m_s}\left[\mathbf{E}(\mathbf{X_s}(t),t) + \mathbf{V_s}(t) \times \mathbf{B}(\mathbf{X_s}(t),t)\right] \\
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{X_s}(t) &= \mathbf{V_s}(t).
\end{aligned}
$$

Attention: TODO: write proper relativistic form

$\mathbf{X}_s = (\mathbf{x}_1, \mathbf{x}_2, ...)_s$ and $\mathbf{V}_s = (\mathbf{v}_1, \mathbf{v}_2, ...)_s$ are vectors of *marker* positions and velocities, respectively, which describe the ensemble of particles belonging to species s.

Note: Particles in a particle species can have different charge states in PIConGPU. In the general case, $\frac{q_s}{m_s}$ is not required to be constant per particle species.

18.3 Electro-Magnetic PIC Method

Fields such as $\mathbf{E}(t), \mathbf{B}(t)$ and $\mathbf{J}(t)$ are discretized on a regular mesh in Eulerian frame of reference (see [EulerLagrangeFrameOfReference]). See *section Finite-Difference Time-Domain Method* describing how Maxwell's equations are discretized on a mesh in PIConGPU.

The distribution of **Particles** is described by the distribution function $f_s(\mathbf{x}, \mathbf{v}, t)$. This distribution function is sampled by *markers*, which are commonly referred to as *macroparticles*. These markers represent blobs of incompressible phase fluid moving in phase space. The temporal evolution of the distribution function is simulated by advancing the markers over time according to the Vlasov–Maxwell–Equation in Lagrangian frame (see eq. (18.1) and [EulerLagrangeFrameOfReference]). A marker has a finite-size and a velocity, such that it can be regarded as a cloud of particles, whose center of mass is the marker's position and whose mean velocity is the marker's velocity. The cloud shape $S^n(x)$ of order n of a marker describes its charge density distribution. See *section Hierarchy of Charge Assignment Schemes* for a list of available marker shapes in PIConGPU.

18.4 References

CHAPTER

NINETEEN

FINITE-DIFFERENCE TIME-DOMAIN METHOD

Section author: Klaus Steiniger, Jakob Trojok, Sergei Bastrakov

For the discretization of Maxwell's equations on a mesh in PIConGPU, only the equations

$$\frac{1}{c^2}\frac{\partial}{\partial t}\vec{E} = \nabla \times \vec{B} - \mu_0 \vec{J}$$
$$\frac{\partial}{\partial t}\vec{B} = -\nabla \times \vec{E}$$

are solved. This becomes possible, first, by correctly solving Gauss's law $\nabla \cdot \vec{E} = \frac{1}{\varepsilon_0}\sum_s \rho_s$ using Esirkepov's current deposition method [Esirkepov2001] (or variants thereof) which solve the discretized continuity equation exactly. Second, by assuming that the initially given electric and magnetic field satisfy Gauss' laws. Starting simulations in an initially charge free and magnetic-divergence-free space, i.e.

$$\nabla \cdot \vec{E} = 0$$
$$\nabla \cdot \vec{B} = 0$$

is standard in PIConGPU. Alternatively, one could use non-charge-free initialization and solve the Poisson equation for initial values of $\vec{E}$.

19.1 Discretization on a staggered mesh

In the Finite-Difference Time-Domain method, above Maxwell's equations are discretized by replacing the partial space and time derivatives with centered finite differences. For example, the partial space derivative along x of a scalar field u at position (i, j, k) and time step n becomes

$$\partial_x u(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = \frac{u^n_{i+1/2,j,k} - u^n_{i-1/2,j,k}}{\Delta x}$$

and the temporal derivative becomes

$$\partial_t u(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = \frac{u^{n+1/2}_{i,j,k} - u^{n-1/2}_{i,j,k}}{\Delta t},$$

when replacing with the lowest order central differences. Note, with this leapfrog discretization or staggering, derivatives of field quantities are calculated at positions between positions where the field quantities are known.

The above discretization uses one neighbor to each side from the point where the derivative is calculated yielding a second order accurate approximation of the derivative. Using more neighbors for finite difference calculation of the spatial derivatives is possible in PIConGPU and increases the approximation order of these derivatives. Note, however, that the order of the whole Maxwell's solver also depends on accuracy of $\vec{J}$ calculation on the grid. For those values PIConGPU provides only second-order accuracy in terms of time and spatial grid steps (as the underlying discretized continuity equation is of that order) regardless of the chosen field solver. Thus, in the general case the Maxwell's solver as a whole still has second order accuracy in space, and only provides arbitrary order in finite difference approximation of curls.

For the latter, the accuracy order scales with twice the number of neighbors M used to approximate the derivative. The arbitrary order finite difference derivative approximation of order $2M$ reads

$$\partial_x u(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = \sum_{l=1/2}^{M-1/2} \left[g_l^{2M} \frac{u^n_{i+l,j,k} - u^n_{i-l,j,k}}{\Delta x} \right], \text{where}$$

$$g_l^{2M} = \frac{(-1)^{l-1/2}}{2l^2} \frac{((2M-1)!!)^2}{(2M-1-2l)!!(2M-1+2l)!!}$$

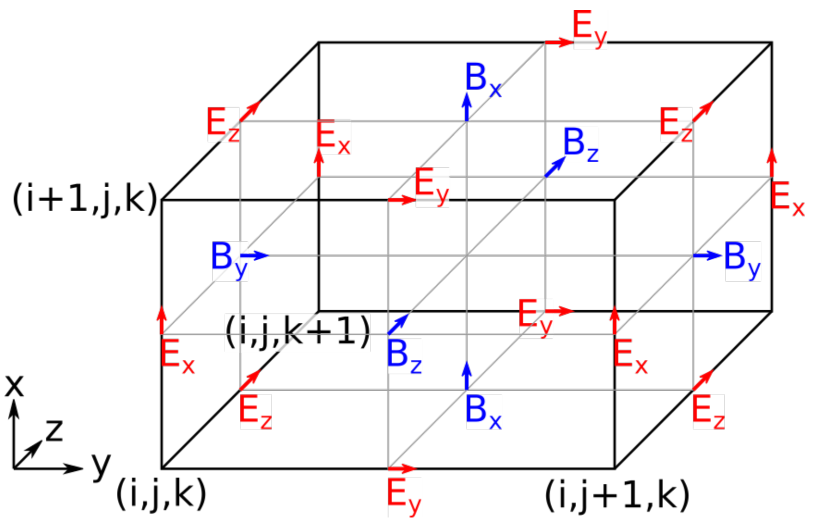
with $l = -M + 1/2, -M + 1 + 1/2, ..., -1/2, 1/2, ..., M - 1/2$ [Ghrist2000]. A recurrence relation for the weights exists,

$$g_l^{2M} = (-1)\frac{(l-1)^2}{l^2} \frac{(2M+1-2l)}{(2M-1+2l)} g_{l-1}^{2M}$$

$$g_{\frac{1}{2}}^{2M} = \frac{16^{1-M}}{M} \left(\frac{(2M-1)!}{[(M-1)!]^2} \right)^2$$

19.2 Maxwell's equations on the mesh

When discretizing on the mesh with centered finite differences, the spatial positions of field components need to be chosen such that a field component, whose **temporal derivative** is calculated on the left hand side of a Maxwell equation, is spatially positioned between the two field components whose **spatial derivative** is evaluated on the right hand side of the respective Maxwell equation. In this way, the spatial points where a left hand side temporal derivative of a field is evaluated lies exactly at the position where the spatial derivative of the right hand side fields is calculated. The following image visualizes the arrangement of field components in PIConGPU.



Component-wise and using second order finite differences for the derivative approximation, Maxwell's equations read in PIConGPU

$$\begin{aligned}
\frac{E_x|_{i+1/2,j,k}^{n+1} - E_x|_{i+1/2,j,k}^{n}}{c^2\Delta t} =& \frac{B_z|_{i+1/2,j+1/2,k}^{n+1/2} - B_z|_{i+1/2,j-1/2,k}^{n+1/2}}{\Delta y} \\
&- \frac{B_y|_{i+1/2,j,k+1/2}^{n+1/2} - B_y|_{i+1/2,j,k-1/2}^{n+1/2}}{\Delta z} - \mu_0 J_x|_{i+1/2,j,k}^{n+1/2} \\
\frac{E_y|_{i,j+1/2,k}^{n+1} - E_y|_{i,j+1/2,k}^{n}}{c^2\Delta t} =& \frac{B_x|_{i,j+1/2,k+1/2}^{n+1/2} - B_x|_{i,j+1/2,k-1/2}^{n+1/2}}{\Delta z} \\
&- \frac{B_z|_{i+1/2,j+1/2,k}^{n+1/2} - B_z|_{i-1/2,j+1/2,k}^{n+1/2}}{\Delta x} - \mu_0 J_y|_{i,j+1/2,k}^{n+1/2} \\
\frac{E_z|_{i,j,k+1/2}^{n+1} - E_z|_{i,j,k+1/2}^{n}}{c^2\Delta t} =& \frac{B_y|_{i+1/2,j,k+1/2}^{n+1/2} - B_y|_{i-1/2,j,k+1/2}^{n+1/2}}{\Delta x} \\
&- \frac{B_x|_{i,j+1/2,k+1/2}^{n+1/2} - B_x|_{i,j-1/2,k+1/2}^{n+1/2}}{\Delta y} - \mu_0 J_z|_{i,j,k+1/2}^{n+1/2} \\
\frac{B_x|_{i,j+1/2,k+1/2}^{n+3/2} - B_x|_{i,j+1/2,k+1/2}^{n+1/2}}{\Delta t} =& \frac{E_y|_{i,j+1/2,k+1}^{n+1} - E_y|_{i,j+1/2,k}^{n+1}}{\Delta z} - \frac{E_z|_{i,j+1,k+1/2}^{n+1} - E_z|_{i,j,k+1/2}^{n+1}}{\Delta y} \\
\frac{B_y|_{i+1/2,j,k+1/2}^{n+3/2} - B_y|_{i+1/2,j,k+1/2}^{n+1/2}}{\Delta t} =& \frac{E_z|_{i+1,j,k+1/2}^{n+1} - E_z|_{i,j,k+1/2}^{n+1}}{\Delta x} - \frac{E_x|_{i+1/2,j,k+1}^{n+1} - E_x|_{i+1/2,j,k}^{n+1}}{\Delta z} \\
\frac{B_z|_{i+1/2,j+1/2,k}^{n+3/2} - B_z|_{i+1/2,j+1/2,k}^{n+1/2}}{\Delta t} =& \frac{E_x|_{i+1/2,j+1,k}^{n+1} - E_x|_{i+1/2,j,k}^{n+1}}{\Delta y} - \frac{E_y|_{i+1,j+1/2,k}^{n+1} - E_y|_{i,j+1/2,k}^{n+1}}{\Delta x}
\end{aligned}$$

As can be seen from these equations, the components of the source current are located at the respective components of the electric field. Following Gauss's law, the charge density is located at the cell corner.

Using Esirkepov's notation for the discretized differential operators,

$$\begin{aligned}
\nabla^+ u_{i,j,k} &= \left(\frac{u_{i+1,j,k} - u_{i,j,k}}{\Delta x}, \frac{u_{i,j+1,k} - u_{i,j,k}}{\Delta y} \frac{u_{i,j,k+1} - u_{i,j,k}}{\Delta z}\right) \\
\nabla^- u_{i,j,k} &= \left(\frac{u_{i,j,k} - u_{i-1,j,k}}{\Delta x}, \frac{u_{i,j,k} - u_{i,j-1,k}}{\Delta y} \frac{u_{i,j,k} - u_{i,j,k-1}}{\Delta z}\right) ,
\end{aligned}$$

the shorthand notation for the discretized Maxwell equations in PIConGPU reads

$$\begin{aligned}
\frac{\vec{E}|^{n+1} - \vec{E}|^{n}}{c^2\Delta t} &= \nabla^- \times \vec{B}|^{n+1/2} - \mu_0 \vec{J}|^{n+1/2} \\
\frac{\vec{B}|^{n+3/2} - \vec{B}|^{n+1/2}}{\Delta t} &= -\nabla^+ \times \vec{E}|^{n+1} \\
\nabla^- \cdot \vec{E}|^{n+1} &= \rho|^{n+1} \\
\nabla^+ \cdot \vec{B}|^{n+3/2} &= 0 ,
\end{aligned}$$

with initial conditions

$$\begin{aligned}
\nabla^- \cdot \vec{E} &= 0 \\
\nabla^+ \cdot \vec{B} &= 0 .
\end{aligned}$$

The components $E_x|_{1/2,0,0} = E_y|_{0,1/2,0} = E_z|_{0,0,1/2} = B_x|_{I,J+1/2,K+1/2} = B_y|_{I+1/2,J,K+1/2} = B_z|_{I+1/2,J+1/2,K} = 0$ for all times when using absorbing boundary conditions. Here, I, J, K are the maximum values of i, j, k defining the total mesh size.

Note, in PIConGPU the $\vec{B}$-field update is split in two updates of half the time step, e.g.

$$\frac{B_x|_{i,j+1/2,k+1/2}^{n+1} - B_x|_{i,j+1/2,k+1/2}^{n+1/2}}{\Delta t/2} = \frac{E_y|_{i,j+1/2,k+1}^{n+1} - E_y|_{i,j+1/2,k}^{n+1}}{\Delta z} - \frac{E_z|_{i,j+1,k+1/2}^{n+1} - E_z|_{i,j,k+1/2}^{n+1}}{\Delta y}$$

and

$$\frac{B_x|_{i,j+1/2,k+1/2}^{n+3/2} - B_x|_{i,j+1/2,k+1/2}^{n+1}}{\Delta t/2} = \frac{E_y|_{i,j+1/2,k+1}^{n+1} - E_y|_{i,j+1/2,k}^{n+1}}{\Delta z} - \frac{E_z|_{i,j+1,k+1/2}^{n+1} - E_z|_{i,j,k+1/2}^{n+1}}{\Delta y}$$

for the B_x component, where the second half of the update is performed at the beginning of the next time step such that the electric and magnetic field are known at equal time in the particle pusher and at the end of a time step.

19.3 Dispersion relation of light waves on a mesh

The dispersion relation of a wave relates its oscillation period in time T to its oscillation wavelength λ, i.e. its angular frequency $\omega = \frac{2\pi}{T}$ to wave vector $\vec{k} = \frac{2\pi}{\lambda}\vec{e}_k$. For an electromagnetic wave in vacuum,

$$\left[\frac{\omega}{c}\right]^2 = k_x^2 + k_y^2 + k_z^2 \,.$$

However, on a 3D mesh, with arbitrary order finite differences for the spatial derivatives, the dispersion relation becomes

$$\left[\frac{1}{c\Delta t}\sin\left(\frac{\omega\Delta t}{2}\right)\right]^2 = \left[\sum_{l=1/2}^{M-1/2} g_l^{2M}\frac{\sin(\tilde{k}_x l\Delta x)}{\Delta x}\right]^2 + \left[\sum_{l=1/2}^{M-1/2} g_l^{2M}\frac{\sin(\tilde{k}_y l\Delta y)}{\Delta y}\right]^2 + \left[\sum_{l=1/2}^{M-1/2} g_l^{2M}\frac{\sin(\tilde{k}_z l\Delta z)}{\Delta z}\right]^2$$

where $\tilde{k}_x$, $\tilde{k}_y$, and $\tilde{k}_z$ are the wave vector components on the mesh in x, y, and z direction, respectively. As is obvious from the relation, the numerical wave vector will be different from the real world wave vector for a given frequency ω due to discretization.

19.3.1 Dispersion Relation for Yee's Method

Yee's method [Yee1966] uses second order finite differences for the approximation of spatial derivatives. The corresponding dispersion relation reads

$$\left[\frac{1}{c\Delta t}\sin\left(\frac{\omega\Delta t}{2}\right)\right]^2 = \left[\frac{1}{\Delta x}\sin\left(\frac{\tilde{k}_x\Delta x}{2}\right)\right]^2 + \left[\frac{1}{\Delta y}\sin\left(\frac{\tilde{k}_y\Delta y}{2}\right)\right]^2 + \left[\frac{1}{\Delta z}\sin\left(\frac{\tilde{k}_z\Delta z}{2}\right)\right]^2 .$$

Obviously, this is a special case of the general dispersion relation, where $M = 1$.

Solving for a wave's numerical frequency ω in dependence on its numerical wave vector $\vec{\tilde{k}} = (\tilde{k}\cos\phi\sin\theta, \tilde{k}\sin\phi\sin\theta, \tilde{k}\cos\theta)$ (spherical coordinates),

$$\omega = \frac{2}{\Delta t}\arcsin\xi \,,$$

where

$$\xi = c\Delta t\sqrt{\left[\frac{1}{\Delta x}\sin\left(\frac{\tilde{k}_x\Delta x}{2}\right)\right]^2 + \left[\frac{1}{\Delta y}\sin\left(\frac{\tilde{k}_y\Delta y}{2}\right)\right]^2 + \left[\frac{1}{\Delta z}\sin\left(\frac{\tilde{k}_z\Delta z}{2}\right)\right]^2} .$$

Denoting

$$\xi_{\max} = c\Delta t\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}$$

we have $\xi \leq \xi_{\max}$ with equality possible for diagonal wave propagation and a certain relation between time and spatial grid steps.

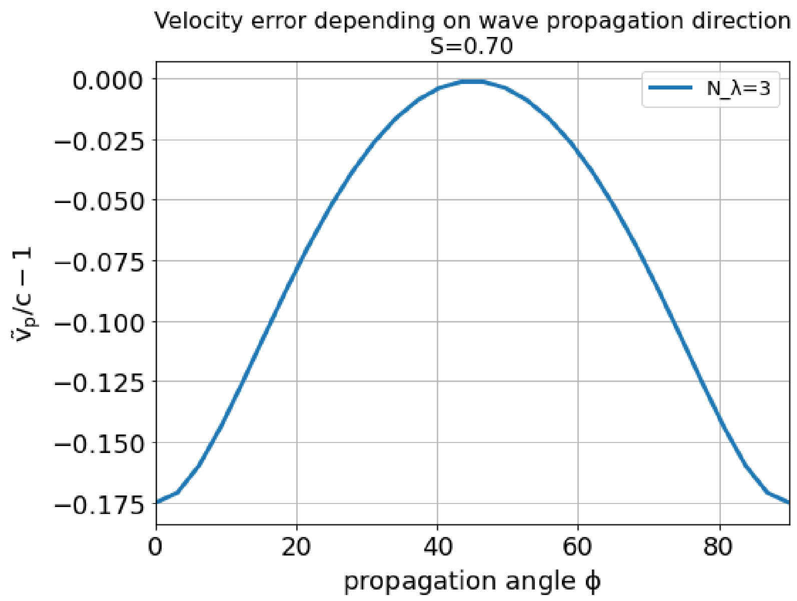
This reveals two important properties of the field solver. (The 2D version is obtained by letting $\tilde{k}_z = 0$.)

First, only within the range $\xi_{\max} \leq 1$ the field solver operates stably. This gives the *Courant-Friedrichs-Lewy* stability condition relating time step to mesh spacing

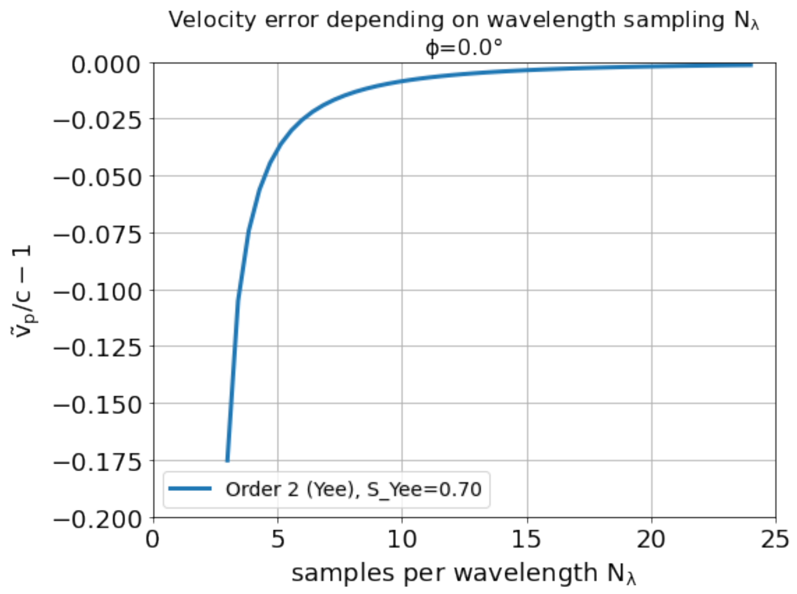
$$c\Delta t < \frac{1}{\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}}$$

Typically, $\xi_{\max} = 0.995$ is chosen. Outside this stability region, the frequency ω corresponding to a certain wave vector becomes imaginary, meaning that wave amplitudes can be nonphysically exponentially amplified [Taflove2005].

Second, there exists a purely numerical anisotropy in a wave's phase velocity $\tilde{v}_p = \omega/\tilde{k}$ (speed of electromagnetic wave propagation) depending on its propagation direction ϕ, as depicted in the following figure



assuming square cells $\Delta x = \Delta y = \Delta$ and where $S = c\Delta t/\Delta$, $N_\lambda = \lambda/\Delta$. That is, for the chosen sampling of three samples per wavelength λ, the phase velocities along a cell edge and a cell diagonal differ by approximately 20%. The velocity error is largest for propagation along the edge. The phase velocity error can be significantly reduced by increasing the sampling, as visualized in the following figure by the scaling of the velocity error with wavelength sampling for propagation along the cell edge



Another conclusion from this figure is, that a short-pulse laser with a large bandwidth will suffer from severe dispersion if the sampling is bad. In the extreme case where a wavelength is not even sampled twice on the mesh, its field is exponentially damped [Taflove2005].

Given that most simulations employ short-pulse lasers propagating along the y-axis and featuring a large bandwidth, the resolution of the laser wavelength should be a lot better than in the example, e.g. $N_\lambda = 24$, to reduce errors due to numerical dispersion.

Note, the reduced phase velocity of light can further cause the emission of numerical Cherenkov radiation by fast charged particles in the simulation [Lehe2012]. The largest emitted wavelength equals the wavelength whose phase velocity is as slow as the particle's velocity, provided it is resolved at least twice on the mesh.

19.3.2 Dispersion Relation for Arbitrary Order Finite Differences

Solving the higher order dispersion relation for the angular frequency yields:

$$\omega = \frac{2}{\Delta t} \arcsin \xi \,,$$

where

$$\begin{aligned}
\xi &= c\Delta t \sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2} \,, \text{ and} \\
\xi_x &= \sum_{l=1/2}^{M-1/2} g_l^{2M} \frac{\sin(\tilde{k}_x l \Delta x)}{\Delta x} \,, \\
\xi_y &= \sum_{l=1/2}^{M-1/2} g_l^{2M} \frac{\sin(\tilde{k}_y l \Delta y)}{\Delta y} \,, \\
\xi_z &= \sum_{l=1/2}^{M-1/2} g_l^{2M} \frac{\sin(\tilde{k}_z l \Delta z)}{\Delta z} \,.
\end{aligned}$$

With

$$\xi_{\max} = c\Delta t \left[\sum_{l=1/2}^{M-1/2} (-1)^{l-\frac{1}{2}} g_l^{2M} \right] \sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}$$

we have $\xi \leq \xi_{\max}$.

The equations are structurally the same as for Yee's method, but contain the alternating sum of the weighting coefficients of the spatial derivative. Again, Yee's Formula is the special case where $M = 1$. For the solver to be stable, $\xi_{\max} < 1$ is required as before. Thus the stability condition reads

$$c\Delta t < \frac{1}{\left[\sum\limits_{l=1/2}^{M-1/2} (-1)^{l-\frac{1}{2}} g_l^{2M} \right] \sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}}$$

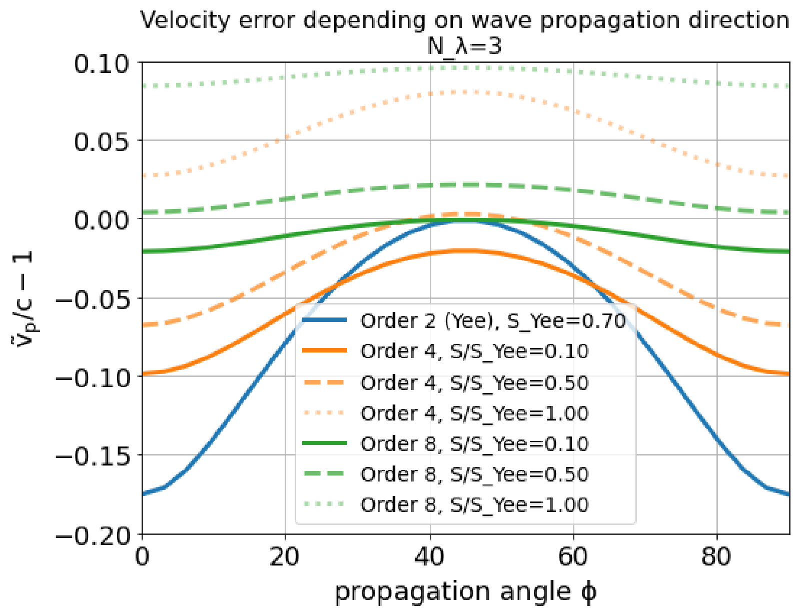
As explained for Yee's method, $\xi_{\max} = 0.995$ is normally chosen and not meeting the stability condition can lead to nonphysical exponential wave amplification.

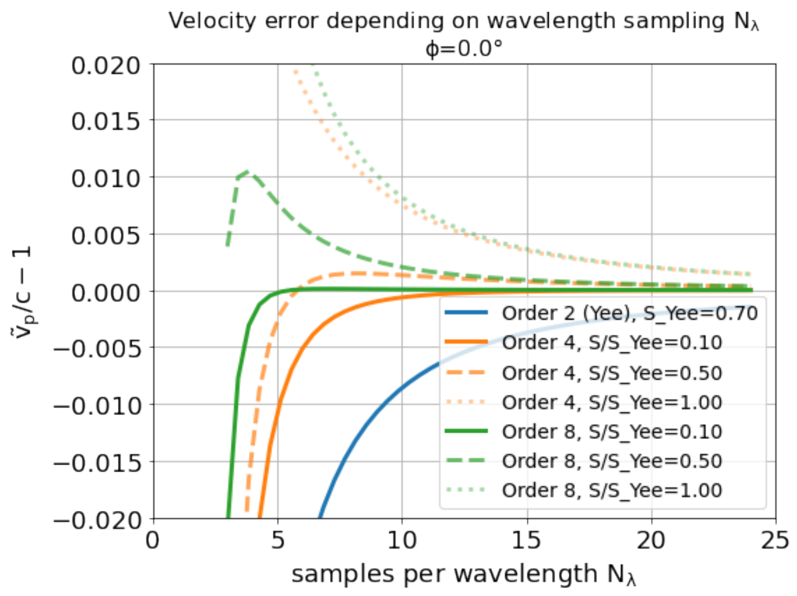
Sample values for the additional factor $\left[\sum\limits_{l=1/2}^{M-1/2} (-1)^{l-\frac{1}{2}} g_l^{2M} \right]$ appearing in the AOFDTD stability condition compared to Yee's method, are

Number of neighbors M	Value of additional factor $\sum (-1)^{l-\frac{1}{2}} g_l^{2M}$
1	1.0
2	1.166667
3	1.241667
4	1.286310
5	1.316691
6	1.339064
7	1.356416
8	1.370381

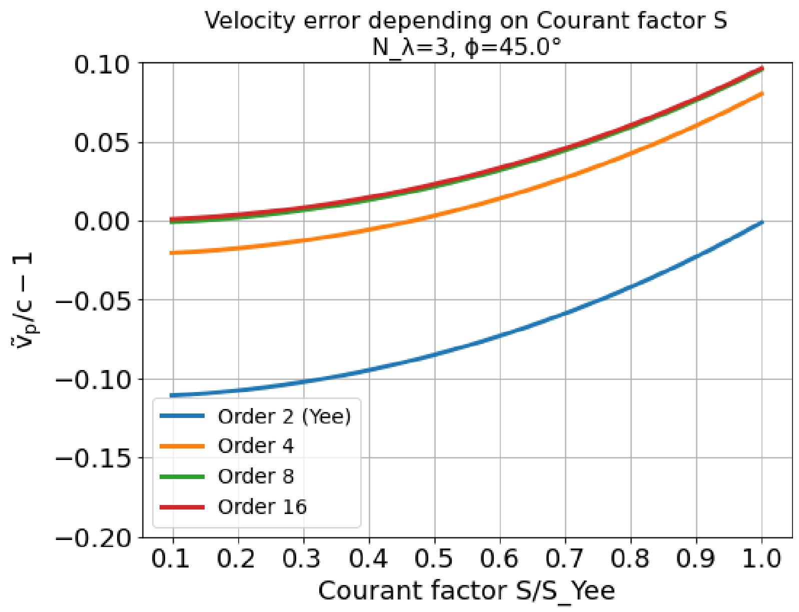
which implies a reduction of the usable time step Δt by the given factor if more than one neighbor is used.

Regarding the numerical anisotropy of the phase velocity, using higher order finite differences for the approximation of spatial derivatives significantly improves the dispersion properties of the solver. Most notably, the velocity anisotropy reduces and the dependence of phase velocity on sampling reduces, too. Yet higher order solvers still feature dispersion. As shown in the following picture, its effect is, however, not reduction of phase velocity but increase of phase velocity beyond the physical vacuum speed of light. But this can be tweaked by reducing the time step relative to the limit set by the stability criterion.





Note, it is generally not a good idea to reduce the time step in Yee's method significantly below the stability criterion as this increases the absolute phase velocity error. See the following figure,



from which the optimum Courant factor $S = c\Delta t/\Delta$ can be read off for a 2D, square mesh, too.

An important conclusion from the above figures showing velocity error over sampling is, that a higher order solver, with a larger mesh spacing and a smaller time step than given by the above stability limit, produces physically more accurate results than the standard Yee solver operating with smaller mesh spacing and a time step close to the stability limit.

That is, it can be beneficial not only in terms of **physical accuracy**, but also in terms of **memory complexity** and **time to solution**, to chose a higher order solver with lower spatial resolution and increased time sampling relative to the stability limit. Memory complexity scales with number of cells N_{cells} required to sample a given volume N_{cells}^d, where $d = 2, 3$ is the dimension of the simulation domain, which decreases for larger cells. Time to solution scales with the time step and this can be larger with solvers of higher order compared to the Yee solver with comparable dispersion properties (which requires a smaller cell size than the arbitrary order solver) since the time step is limited by the stability condition which scales with cell size. Since the cell size can be larger for arbitrary order solvers, the respective time step limit given by the stability condition is larger and operating with a time step ten times smaller than the limit might still result in a larger step than those of the comparable Yee solver. Finally, physical accuracy is increased by the reduction of the impact of dispersion effects.

19.4 Usage

The field solver can be chosen and configured in *fieldSolver.param*.

19.5 Substepping

Any field solver can be combined with substepping in time. In this case, each iteration of the main PIC loop involves multiple invocations of the chosen field solver. Substepping is fully compatible with other numerics, such as absorbers, incident field, laser generation. A substepping field solver has the same orders of accuracy in spatial and time steps as the base solver.

A user sets main PIC time step value as usual in *grid.param*, and selects the number of field solver substeps via template argument *T_numSubsteps*. Field solver will internally operate with $\Delta t_{sub} = \Delta t/\text{T\_numSubsteps}$. Solver properties including the Courant-Friedrichs-Lewy condition are then expressed with Δt_{sub}, which is less restricting. However, regardless of field solver and substepping PIConGPU also limits its main time step to

$$\Delta t < \Delta t_{max}, \Delta t_{max} = \frac{1}{c} \min\{\Delta x, \Delta y, \Delta z\}$$

for 3D; for 2D the requirement is similar but does not involve Δz.

Still, with substepping solver a user could sometimes increase the value of Δt used. Consider a simplified case of $\Delta x = \Delta y = \Delta z$, and Yee solver with a time step value near the Courant-Friedrichs-Lewy condition threshold $\Delta t_{base} = 0.995 \frac{\Delta x}{c\sqrt{3}}$. Raising the main time step to t_{max} and substepping by 2 would make the field solver have $\Delta t_{sub} < \Delta t_{base}$ thus satisfying the Courant-Friedrichs-Lewy condition and providing slightly better resolution. Whether this approach is applicable then depends on whether t_{max} sufficiently resolves plasma wavelength and other time scales in the rest of the PIC loop.

19.6 References

CHAPTER

TWENTY

TOTAL FIELD/SCATTERED FIELD

Section author: Sergei Bastrakov

This section describes the Total Field/Scattered Field (TF/SF) technique for adding incident field to a simulation. TF/SF is used for simulating a field produced by external sources and coming towards the simulation volume from any boundaries and directions. PIConGPU implements this technique as incident field. There are also *other ways to generate a laser*.

First, we present the general idea of TF/SF and describe its formulation for the standard Yee field solver. Then we show how it is generalized for any FDTD-type solver, this general version is implemented in PIConGPU. Finally, some aspects of using the PIConGPU incident field are covered.

20.1 Introduction

When applying TF/SF, we operate on a domain where field values are updated by an explicit FDTD-type Maxwell's solver. In case a simulation uses a field absorber, the absorber operates externally to this area and independently of TF/SF (no special coupling needed).

This section uses the same Yee grid and notation as the *FDTD description*. The description and naming is generally based on [Potter2017] [Taflove2005], however there are some differences in notation (also those two sources use different Yee grid layouts).

The field produced in the domain by external sources (such as incoming laser) is called *incident field* and denoted $\vec{E}^{inc}(x, y, z, t)$ and $\vec{B}^{inc}(x, y, z, t)$. *Total field (TF)* is the combined field from the internally occuring field dynamic and the incident field, we denote it $\vec{E}^{tot}(x, y, z, t)$ and $\vec{B}^{tot}(x, y, z, t)$. Thus, total field is a "normal" result of a simulation with the laser. Finally, the *scattered field (SF)* is the difference of these two fields:

$$\vec{E}^{scat}(x, y, z, t) = \vec{E}^{tot}(x, y, z, t) - \vec{E}^{inc}(x, y, z, t)$$
$$\vec{B}^{scat}(x, y, z, t) = \vec{B}^{tot}(x, y, z, t) - \vec{B}^{inc}(x, y, z, t)$$

Scattered field does not have the laser effect, and transparently propagates the waves outgoing from the TF area towards domain boundaries. However, note that Maxwell's equations hold in both the TF and SF regions. Special handling is needed only near the boundary betweeen the regions.

The field values are represented as a Yee grid of $\vec{E}$ and $\vec{B}$ values in the domain, TF/SF does not require additionally stored data. However, it changes the interpretation of grid values: the domain is virtually subdivided into the internal area where the stored grid values are TF and external area with SF values.

The subvidision is done with an axis-aligned *Huygens surface* S. The Huygens surface is chosen so that no Yee grid nodes lay on it. So it could be located at an arbitrary position that does not collide with cell- and half-cell boundaries.

In PIConGPU, the position of the Huygens surface is defined relative to the internal boundary of the field absorber. The surface is located inwards relative to each boundary by a user-defined position in full cells and an additional 0.75 cells. The equations presented in this section hold for this 0.75 shift and the Yee grid layout used in PIConGPU. In principle, a similar derivation can be done in any case, but the resulting expression would not generally match index-by-index.

We use TF/SF formulation to generate fields at the Huygens surface that propagate inwards the simulation area and act as application of $\vec{E}^{inc}(x,y,z,t)$, $\vec{B}^{inc}(x,y,z,t)$. The fields would mostly propagate in this direction (and not equally to both sides), and as a theoretical limit only inwards. However, in practice due to numerical dispersion a small portion, such as 10^{-3} of the amplitude, would also propagate externally.

The TF/SF technique can be interpreted in two ways. First, as manipulations on discrete FDTD equations given the virtual separation of the domain described above. This "math" view is used in the following description for the Yee and general-case FDTD solvers.

Alternatively, TF/SF can be treated as applying the equivalence theorem on the Huygens surface. Then it is equivalent to impressing the electric and magnetic current sheets, $\vec{J}$ and $\vec{K}$ respectively, on S:

$$\vec{J}|_S = -\vec{n} \times \frac{1}{\mu_0}\vec{B}^{inc}|_S$$
$$\vec{K}|_S = \vec{n} \times \vec{E}^{inc}|_S$$

where $\vec{n}$ is a unit normal vector on S pointing towards the internal volume. Both points of view in principle result in the same scheme.

20.2 TF/SF for Standard Yee Solver

20.2.1 Single Boundary

First we describe application of a field source only at the x_{min} boundary. For this case, suppose the source is given along a plane $x = x_{min} + \text{position}\Delta x + 0.75\Delta x$ and acts along the whole domain in y and z. The source affects transversal field components E_y, E_z, B_y, B_z. Components E_x, B_x are not affected by TF/SF and we do not consider them in the following description. We also omit the $\vec{J}$ term which is applied as usual in the field solver and is also not affected by TF/SF.

The Yee solver update equations for the affected field components and terms are as follows:

$$\begin{aligned}
\frac{E_y|_{i,j+1/2,k}^{n+1} - E_y|_{i,j+1/2,k}^{n}}{c^2\Delta t} &= \frac{B_x|_{i,j+1/2,k+1/2}^{n+1/2} - B_x|_{i,j+1/2,k-1/2}^{n+1/2}}{\Delta z} \\
&- \frac{B_z|_{i+1/2,j+1/2,k}^{n+1/2} - B_z|_{i-1/2,j+1/2,k}^{n+1/2}}{\Delta x} \\
\frac{E_z|_{i,j,k+1/2}^{n+1} - E_z|_{i,j,k+1/2}^{n}}{c^2\Delta t} &= \frac{B_y|_{i+1/2,j,k+1/2}^{n+1/2} - B_y|_{i-1/2,j,k+1/2}^{n+1/2}}{\Delta x} \\
&- \frac{B_x|_{i,j+1/2,k+1/2}^{n+1/2} - B_x|_{i,j-1/2,k+1/2}^{n+1/2}}{\Delta y} \\
\frac{B_y|_{i+1/2,j,k+1/2}^{n+3/2} - B_y|_{i+1/2,j,k+1/2}^{n+1/2}}{\Delta t} &= \frac{E_z|_{i+1,j,k+1/2}^{n+1} - E_z|_{i,j,k+1/2}^{n+1}}{\Delta x} - \frac{E_x|_{i+1/2,j,k+1}^{n+1} - E_x|_{i+1/2,j,k}^{n+1}}{\Delta z} \\
\frac{B_z|_{i+1/2,j+1/2,k}^{n+3/2} - B_z|_{i+1/2,j+1/2,k}^{n+1/2}}{\Delta t} &= \frac{E_x|_{i+1/2,j+1,k}^{n+1} - E_x|_{i+1/2,j,k}^{n+1}}{\Delta y} - \frac{E_y|_{i+1,j+1/2,k}^{n+1} - E_y|_{i,j+1/2,k}^{n+1}}{\Delta x}
\end{aligned}$$

When using TF/SF technique, first a usual Yee field solver update is applied to the whole grid, regardless of TF and SF regions. Then a separate stage that we call *incident field solver* is run to modify the calculated values where necessary. The combined effect of the Yee- and incident field solvers is that Maxwell's equations hold on the whole grid and the correct incident field is generated. We now proceed to describe how are these values identified and what is the modification necessary.

As mentioned above, values like $E_y|_{i,j+1/2,k}^{n+1}$ are stored for the whole Yee grid. Whether they represent the total or the scattered field, depends on the position of the node relative to the Huygens surface. To avoid confusion, we use the $E_y|_{i,j+1/2,k}^{n+1}$ notation for stored grid values, and $E_y^{other}(i\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1)\Delta t)$ to denote fields at the same time and space position, but not stored long-term.

Since the Maxwell's equations hold in both the TF and SF regions, all Yee solver updates involving only grid values from the same region produced correct values that do not need any further modification. A correction is only needed for grid values that were calculated using a mix of TF and SF values. Since the standard Yee solver uses a 2-point central derivative operator, those are a single layer of $\vec{E}$ and $\vec{B}$ values located near S.

Taking into account the 0.75 shift inwards used by PIConGPU, denote the x position of S as $x_S = (i_S + 0.75)\Delta x$. Then the grid values to be modified by the incident field solver are $E_y|_{i_S+1,j+1/2,k}^{n+1}$, $E_z|_{i_S+1,j,k+1/2}^{n+1}$, $B_y|_{i_S+1/2,j,k+1/2}^{n+3/2}$, and $B_z|_{i_S+1/2,j+1/2,k}^{n+3/2}$. (All grid values to the right side of those were calculated using only TF values and all grid values on the left side were calculated using only SF values.)

Consider the update of $E_y|_{i_S+1,j+1/2,k}^{n+1}$ performed by a standard Yee solver for each j,k. All terms but $B_z|_{i_S+1/2,j+1/2,k}^{n+1/2}$ in this update are in the TF region. Thus, this value has to be modified by the incident field solver in order to preseve the Maxwell's equations.

To derive the modification necessary, consider a hypothetical Maxwell's-preserving update at this point where all participating values were TF:

$$\frac{E_y^{tot}\left((i_S+1)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1)\Delta t\right)-E_y|_{i_S+1,j+1/2,k}^{n}}{c^2\Delta t}=$$
$$\frac{B_x|_{i_S+1,j+1/2,k+1/2}^{n+1/2}-B_x|_{i_S+1,j+1/2,k-1/2}^{n+1/2}}{\Delta z}-$$
$$\frac{B_z|_{i_S+3/2,j+1/2,k}^{n+1/2}-B_z^{tot}\left((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t\right)}{\Delta x}$$

Since $B_z|_{i_S+1/2,j+1/2,k}^{n+1/2}$ is an SF and by definition of TF and SF,

$$B_z^{tot}\left((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t\right)=$$
$$B_z|_{i_S+1/2,j+1/2,k}^{n+1/2}+B_z^{inc}\left((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t\right)$$

Substituting it into the update equation and regrouping the terms yields:

$$E_y^{tot}((i_S+1)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1)\Delta t)=E_y|_{i_S+1,j+1/2,k}^{n}$$
$$+c^2\Delta t\left(\frac{B_x|_{i_S+1,j+1/2,k+1/2}^{n+1/2}-B_x|_{i_S+1,j+1/2,k-1/2}^{n+1/2}}{\Delta z}-\right.$$
$$\left.\frac{B_z|_{i_S+3/2,j+1/2,k}^{n+1/2}-(B_z|_{i_S+1/2,j+1/2,k}^{n+1/2}+B_z^{inc}((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t))}{\Delta x}\right)$$
$$=E_y|_{i_S+1,j+1/2,k}^{n}+c^2\Delta t\left(\frac{B_x|_{i_S+1,j+1/2,k+1/2}^{n+1/2}-B_x|_{i_S+1,j+1/2,k-1/2}^{n+1/2}}{\Delta z}-\right.$$
$$\left.\frac{B_z|_{i_S+3/2,j+1/2,k}^{n+1/2}-B_z|_{i_S+1/2,j+1/2,k}^{n+1/2}}{\Delta x}\right)$$
$$+\frac{c^2\Delta t}{\Delta x}B_z^{inc}((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t)$$
$$=E_y|_{i_S+1,j+1/2,k}^{n+1}+\frac{c^2\Delta t}{\Delta x}B_z^{inc}((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t)$$

Thus, in the incident field stage we have to apply the following update to the grid value calculated by a normal Yee solver :

$$E_y|_{i_S+1,j+1/2,k}^{n+1}+=\frac{c^2\Delta t}{\Delta x}B_z^{inc}((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t)$$

Grid value $E_z|_{i_S+1,j,k+1/2}^{n+1}$ is also located in the TF region and with a similar derivation the update for it is

$$E_z|_{i_S+1,j,k+1/2}^{n+1}+=-\frac{c^2\Delta t}{\Delta x}B_y^{inc}((i_S+1/2)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1/2)\Delta t)$$

Values $B_y|_{i_S+1/2,j,k+1/2}^{n+3/2}$, and $B_z|_{i_S+1/2,j+1/2,k}^{n+3/2}$ are in the SF region. For them the Yee solver update includes one term from the TF region, $E_z|_{i_S,j,k+1/2}^{n+1}$ and $E_y|_{i_S,j+1/2,k}^{n+1}$ respectively. Making a similar replacement of an SF value as a difference between a TF value and the incident field value and regrouping, the following update must be applied:

$$\begin{aligned}
B_y|_{i_S+1/2,j,k+1/2}^{n+3/2} += -\frac{\Delta t}{\Delta x} E_z^{inc}((i_S+1)\Delta x, j\Delta y, (k+1/2)\Delta z, (n+1)\Delta t) \\
B_z|_{i_S+1/2,j+1/2,k}^{n+3/2} += \frac{\Delta t}{\Delta x} E_y^{inc}((i_S+1)\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1)\Delta t)
\end{aligned}$$

The derivation for the x_{max} boundary can be done in a similar fashion. Denote the position of S as $x_S = (i_{S,max} + 0.25)\Delta x$. Note that our 0.75 cells inwards shift of S is symmetrical in terms of distance. It implies that the Yee grid incides along x are not fully symmetric between the two sides of each bondary. The update scheme for x_{max} is:

$$\begin{aligned}
E_y|_{i_{S,max},j+1/2,k}^{n+1} += -\frac{c^2\Delta t}{\Delta x} B_z^{inc}((i_{S,max}+1/2)\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1/2)\Delta t) \\
E_z|_{i_{S,max},j,k+1/2}^{n+1} += \frac{c^2\Delta t}{\Delta x} B_y^{inc}((i_{S,max}+1/2)\Delta x, j\Delta y, (k+1/2)\Delta z, (n+1/2)\Delta t) \\
B_y|_{i_{S,max}+1/2,j,k+1/2}^{n+3/2} += \frac{\Delta t}{\Delta x} E_z^{inc}((i_{S,max}+1)\Delta x, j\Delta y, (k+1/2)\Delta z, (n+1)\Delta t) \\
B_z|_{i_{S,max}+1/2,j+1/2,k}^{n+3/2} += -\frac{\Delta t}{\Delta x} E_y^{inc}((i_{S,max}+1)\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1)\Delta t)
\end{aligned}$$

20.2.2 Multiple Boundaries

In the general case, S is comprised of several axis-aligned boundary hyperplanes, 6 planes in 3D, and 4 lines in 2D.

The scheme described above is symmetric for all axes. In case incident field is coming from multiple boundaries, the updates are in principle the same. They can generally be treated as sequential application of the single-boundary case.

Applying TF/SF for each boundary affects the derivatives in the normal direction relative to the boundary. For the standard Yee solver, a single layer of $\vec{E}$ and $\vec{B}$ values along the boundary is affected. Along other directions, we update all grid values that are internal relative to the Huygens surface. In case a "corner" grid node is near several boundaries, it is updated in all the respective applications of TF/SF.

20.3 General Case FDTD

The same principle as for the Yee solver can be applied to any FDTD-type field solver. Same as above, we consider the case of x_{min} boundary and E_y field component. The other boundaries and components are treated symmetrically.

We now apply a general-case spatial-only finite-difference operator to calculate derivatives along x. Such operators on the Yee grid naturally have an antisymmetry of coefficients in x relative to the evaluation point. The antisymmetry is not critical for the following description, but is present in the FDTD solvers implemented and allow simplifying the formulations, and so we assume it. For dB_z/dx such an operator has the following general form:

$$\begin{aligned}
&\partial_x B_z(i\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1/2)\Delta t) = \\
&\frac{1}{\Delta x} \sum_{ii=0}^{m_x} \sum_{jj=-m_y}^{m_y} \sum_{kk=-m_z}^{m_z} \alpha_{ii,jj,kk} \left(B_z|_{i+(ii+1/2),j+jj+1/2,k+kk}^{n+1/2} - B_z|_{i-(ii+1/2),j+jj+1/2,k+kk}^{n+1/2} \right)
\end{aligned}$$

Note that there is also typically a symmetry of coefficients along y and z: $\alpha_{ii,jj,kk} = \alpha_{ii,-jj,kk}$, $\alpha_{ii,jj,kk} = \alpha_{ii,jj,-kk}$ but it is not significant for TF/SF. The derivative operator used by the standard Yee solver has $m_x = m_y = m_z = 0, \alpha_{0,0,0} = 1$.

Same as before, denote the x position of S as $x_S = (i_S + 0.75)\Delta x$. In order to stay within the grid, we require that $i_S \geq m_x$. The incident field solver has to update the grid values of E_y for which calculating dB_z/dx involves a mix of TF and SF values. These values can be present in both the TF and SF regions around S:

$$E_{TF} = \{E_y|^{n+1}_{i_S+1+ii,j+1/2,k} : ii = 0, 1, \dots, m_x\}$$
$$E_{SF} = \{E_y|^{n+1}_{i_S+1-ii,j+1/2,k} : ii = 1, 2, \dots, m_x\}$$

Take a node in the TF region $E_y|^{n+1}_{i_0,j+1/2,k} \in E_{TF}$ ($i_0 = i_S + 1 + ii_0$ for some $ii_0 \in [0, m_x]$). During the FDTD update of this node, the dB_z/dx operator is calculated:

$$\partial_x B_z(i_0\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1/2)\Delta t) =$$
$$\frac{1}{\Delta x}\sum_{ii=0}^{m_x}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}\alpha_{ii,jj,kk}\left(B_z|^{n+1/2}_{i_0+(ii+1/2),j+jj+1/2,k+kk} - B_z|^{n+1/2}_{i_0-(ii+1/2),j+jj+1/2,k+kk}\right)$$

We split the outer sum over ii into two parts:

$$\partial_x B_z(i_0\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1/2)\Delta t) =$$
$$\frac{1}{\Delta x}\sum_{ii=0}^{i_0-i_S-2}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}\alpha_{ii,jj,kk}\left(B_z|^{n+1/2}_{i_0+(ii+1/2),j+jj+1/2,k+kk} - B_z|^{n+1/2}_{i_0-(ii+1/2),j+jj+1/2,k+kk}\right) +$$
$$\frac{1}{\Delta x}\sum_{ii=i_0-i_S-1}^{m_x}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}\alpha_{ii,jj,kk}\left(B_z|^{n+1/2}_{i_0+(ii+1/2),j+jj+1/2,k+kk} - B_z|^{n+1/2}_{i_0-(ii+1/2),j+jj+1/2,k+kk}\right)$$

The first sum over $ii \in [0, i_0 - i_S - 2]$ only uses B_z grid values in the TF region (the minimal index in x used is $B_z|^{n+1/2}_{i_S+3/2,j+jj+1/2,k+kk}$ for $ii = i_0 - i_S - 2$). Note that if $i_0 - i_S - 2 < 0$, this sum has no terms and is equal to 0; the same applies for the following sums. Since the E_y value in question is also a TF, these terms do not require any action by incident field solver. The remaining sum over $ii \in [i_0 - i_S - 1, m_x]$ contains differences of a TF value and an SF value. Each of the latter ones requires a term in the incident field solver update of $E_y|^{n+1}_{i_0,j+1/2,k}$.

Performing the same kind of substitution and regrouping demonstrated above for the standard Yee solver yields

$$E_y^{tot}(i_0\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1)\Delta t) = E_y|^{n+1}_{i_0,j+1/2,k} +$$
$$\frac{c^2\Delta t}{\Delta x}\sum_{ii=i_0-i_S-1}^{m_x}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}(\alpha_{ii,jj,kk}\cdot$$
$$B_z^{inc}((i_0 - (ii+1/2))\Delta x, (j+jj+1/2)\Delta y, (k+kk)\Delta z, (n+1/2)\Delta t))$$

Thus, we apply the following update for each grid value $E_y|^{n+1}_{i_0,j+1/2,k} \in E_{TF}$:

$$E_y|^{n+1}_{i_0,j+1/2,k} += $$
$$\frac{c^2\Delta t}{\Delta x}\sum_{ii=i_0-i_S-1}^{m_x}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}(\alpha_{ii,jj,kk}\cdot$$
$$B_z^{inc}((i_0 - (ii+1/2))\Delta x, (j+jj+1/2)\Delta y, (k+kk)\Delta z, (n+1/2)\Delta t))$$

For values in SF the treatment is similar. For a node $E_y|^{n+1}_{i_0,j+1/2,k} \in E_{SF}$ ($i_0 = i_S + 1 - ii_0$ for some $ii_0 \in [1, m_x]$) we apply dB_z/dx operator and split the outer sum the same way:

$$\partial_x B_z(i_0\Delta x, (j+1/2)\Delta y, k\Delta z, (n+1/2)\Delta t) =$$
$$\frac{1}{\Delta x}\sum_{ii=0}^{i_S-i_0}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}\alpha_{ii,jj,kk}\left(B_z|^{n+1/2}_{i_0+(ii+1/2),j+jj+1/2,k+kk} - B_z|^{n+1/2}_{i_0-(ii+1/2),j+jj+1/2,k+kk}\right) +$$
$$\frac{1}{\Delta x}\sum_{ii=i_S+1-i_0}^{m_x}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}\alpha_{ii,jj,kk}\left(B_z|^{n+1/2}_{i_0+(ii+1/2),j+jj+1/2,k+kk} - B_z|^{n+1/2}_{i_0-(ii+1/2),j+jj+1/2,k+kk}\right)$$

The first sum only has values in the SF region, and the second sum contains differences of TF and SF values. Note that now $E_y|_{i_0,j+1/2,k}^{n+1}$ is in the SF region and so we express the whole update as for SF:

$$E_y^{scat}(i_0\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1)\Delta t) = E_y|_{i_0,j+1/2,k}^{n+1}+$$
$$\frac{c^2\Delta t}{\Delta x}\sum_{ii=i_S+1-i_0}^{m_x}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}(\alpha_{ii,jj,kk}\cdot$$
$$B_z^{inc}((i_0+(ii+1/2))\Delta x,(j+jj+1/2)\Delta y,(k+kk)\Delta z,(n+1/2)\Delta t))$$

Thus, we apply the following update for each grid value $E_y|_{i_0,j+1/2,k}^{n+1}\in E_{SF}$:

$$E_y|_{i_0,j+1/2,k}^{n+1}+=$$
$$\frac{c^2\Delta t}{\Delta x}\sum_{ii=i_S+1-i_0}^{m_x}\sum_{jj=-m_y}^{m_y}\sum_{kk=-m_z}^{m_z}(\alpha_{ii,jj,kk}\cdot$$
$$B_z^{inc}((i_0+(ii+1/2))\Delta x,(j+jj+1/2)\Delta y,(k+kk)\Delta z,(n+1/2)\Delta t))$$

Other field components, axes and directions are treated in a similar way.

20.3.1 Example: 4th Order FDTD

For example, consider the *4th order FDTD* and x_{min} boundary. Its derivative operator has $m_x=1, m_y=m_z=0$, $\alpha_{0,0,0}=27/24$, $\alpha_{1,0,0}=-1/24$. Three layers of E_y are updated, the first in the SF region and the latter two in the TF region:

$$E_y|_{i_S,j+1/2,k}^{n+1}+=\frac{c^2\Delta t}{\Delta x}\left(-\frac{1}{24}B_z^{inc}\left((i_S+3/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t\right)\right)$$
$$E_y|_{i_S+1,j+1/2,k}^{n+1}+=\frac{c^2\Delta t}{\Delta x}\left(\frac{27}{24}B_z^{inc}\left((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t\right)\right.$$
$$\left.-\frac{1}{24}B_z^{inc}\left((i_S-1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t\right)\right)$$
$$E_y|_{i_S+2,j+1/2,k}^{n+1}+=\frac{c^2\Delta t}{\Delta x}\left(-\frac{1}{24}B_z^{inc}\left((i_S+1/2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1/2)\Delta t\right)\right)$$

Updates of E_z are done in a similar fashion:

$$E_z|_{i_S,j,k+1/2}^{n+1}+=-\frac{c^2\Delta t}{\Delta x}\left(-\frac{1}{24}B_y^{inc}\left((i_S+3/2)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1/2)\Delta t\right)\right)$$
$$E_z|_{i_S+1,j,k+1/2}^{n+1}+=-\frac{c^2\Delta t}{\Delta x}\left(\frac{27}{24}B_y^{inc}\left((i_S+1/2)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1/2)\Delta t\right)\right.$$
$$\left.-\frac{1}{24}B_y^{inc}\left((i_S-1/2)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1/2)\Delta t\right)\right)$$
$$E_z|_{i_S+2,j,k+1/2}^{n+1}+=-\frac{c^2\Delta t}{\Delta x}\left(-\frac{1}{24}B_y^{inc}\left((i_S+1/2)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1/2)\Delta t\right)\right)$$

Three layers of B_y are updated, the first two in the SF region and the last one in the TF region:

$$B_y|_{i_S-1/2,j,k+1/2}^{n+3/2}+=-\frac{\Delta t}{\Delta x}\left(-\frac{1}{24}E_z^{inc}\left((i_S+1)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1)\Delta t\right)\right)$$
$$B_y|_{i_S+1/2,j,k+1/2}^{n+3/2}+=-\frac{\Delta t}{\Delta x}\left(\frac{27}{24}E_z^{inc}\left((i_S+1)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1)\Delta t\right)\right.$$
$$\left.-\frac{1}{24}E_z^{inc}\left((i_S+2)\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1)\Delta t\right)\right)$$
$$B_y|_{i_S+3/2,j,k+1/2}^{n+3/2}+=-\frac{\Delta t}{\Delta x}\left(-\frac{1}{24}E_z^{inc}\left(i_S\Delta x,j\Delta y,(k+1/2)\Delta z,(n+1)\Delta t\right)\right)$$

Finally, updates of B_z are as follows:

$$
\begin{aligned}
&B_z|_{i_S-1/2,j+1/2,k}^{n+3/2} += \frac{\Delta t}{\Delta x}\left(-\frac{1}{24}E_y^{inc}\left((i_S+1)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1)\Delta t\right)\right)\\
&B_z|_{i_S+1/2,j+1/2,k}^{n+3/2} += \frac{\Delta t}{\Delta x}\left(\frac{27}{24}E_y^{inc}\left((i_S+1)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1)\Delta t\right)\right.\\
&\left.-\frac{1}{24}E_y^{inc}\left((i_S+2)\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1)\Delta t\right)\right)\\
&B_z|_{i_S+3/2,j+1/2,k}^{n+3/2} += \frac{\Delta t}{\Delta x}\left(-\frac{1}{24}E_y^{inc}\left(i_S\Delta x,(j+1/2)\Delta y,k\Delta z,(n+1)\Delta t\right)\right)
\end{aligned}
$$

20.4 Calculating Incident B from E

Consider a case when both $\vec{E}^{inc}(x,y,z,t)$ and $\vec{B}^{inc}(x,y,z,t)$ are theoretically present, but only $\vec{E}^{inc}(x,y,z,t)$ is known in explicit form.

When slowly varying elvelope approximation (SVEA) is applicable, one may employ it to calculate the other field as $\vec{B}^{inc}(x,y,z,t) = \vec{k} \times \vec{E}^{inc}(x,y,z,t)/c$.

Otherwise one may use TF/SF with only the modified known field set as incident and the other one set to 0. Generally, the interpretation of the result is assisted by the equivalence theorem, and in particular Love and Schelkunoff equivalence principles [Harrington2001] [Balanis2012]. Having $\vec{E}^{inc}(x,y,z,t) = \vec{0}$ means only electric current $\vec{J}$ would be impressed on S. Taking into account no incident fields in the SF region, the region is effectively a perfect magnetic conductor. Likewise, having $\vec{B}^{inc}(x,y,z,t) = \vec{0}$ corresponds to only magnetic current and effectively a perfect electric conductor in the SF region. Practically, one may try using the known incident field with twice the amplitude and keeping the other incident field zero, as demonstrated in [Rengarajan2000]. Note that using this approach in PIConGPU results in generating pulses going both inwards and outwards of the Huygens surface (similar to laser profiles with `initPlaneY > 0`). Therefore, in this case it is recommended to have no density outside the surface and use a strong field absorber to negate the effect of the artificial outwards-going pulse.

20.5 References

CHAPTER

TWENTYONE

HIERARCHY OF CHARGE ASSIGNMENT SCHEMES

Section author: Klaus Steiniger

In PIConGPU, the cloud shapes $S^n(x)$ are pre-integrated to *assignment functions* $W^n(x)$.

$$W^n(x) = \Pi(x) * S^n(x) = \int\limits_{-\infty}^{+\infty} \Pi(x')S^n(x'-x)dx' \, , \text{ where } \Pi(x) = \begin{cases} 0 & |x|\frac{1}{2} \\ \frac{1}{2} & |x| = \frac{1}{2} \\ 1 & |x|\frac{1}{2} \end{cases}$$

is the top-hat function and $*$ the convolution.

Evaluating the assignment functions at mesh points directly provides the fraction of charge from the marker assigned to that point.

The assignment functions are implemented as B-splines. The zeroth order assignment function W^0 is the top-hat function Π. It represents charge assignment to the nearest mesh point only, resulting in a stepwise charge density distribution. Therefore, it should not be used. The assignment function of order n is generated by convolution of the assignment function of order $n-1$ with the top-hat function

$$W^n(x) = W^{n-1}(x) * \Pi(x) = \int\limits_{-\infty}^{+\infty} W^{n-1}(x')\Pi(x'-x)dx' \, .$$

The three dimensional assignment function is a multiplicative union of B-splines $W^n(x,y,z) = W^n(x)W^n(y)W^n(z)$.

PIConGPU implements these up to order $n = 4$. The naming scheme follows [HockneyEastwood], tab. 5-1, p. 144, where the name of a scheme is defined by the visual form of its cloud shape S.

Scheme	Order	Assignment function
NGP (nearest-grid-point)	0	stepwise
CIC (cloud-in-cell)	1	piecewise linear spline
TSC (triangular shaped cloud)	2	piecewise quadratic spline
PQS (piecewise quadratic cloud shape)	3	piecewise cubic spline
PCS (piecewise cubic cloud shape)	4	piecewise quartic spline

21.1 References

CHAPTER

TWENTYTWO

LANDAU-LIFSCHITZ RADIATION REACTION

Module author: Richard Pausch, Marija Vranic

To do

22.1 References

CHAPTER

TWENTYTHREE

FIELD IONIZATION

Section author: Marco Garten

Module author: Marco Garten

Get started here https://github.com/ComputationalRadiationPhysics/picongpu/wiki/Ionization-in-PIConGPU

PIConGPU features an adaptable ionization framework for arbitrary and combinable ionization models.

Note: Most of the calculations and formulae in this section of the docs are done in the **Atomic Units (AU)** system.

$$\hbar = \mathrm{e} = m_\mathrm{e} = 1$$

Table 23.1: **Atomic Unit System**

AU	SI
length	$5.292 \cdot 10^{-11}\,\mathrm{m}$
time	$2.419 \cdot 10^{-17}\,\mathrm{s}$
energy	$4.360 \cdot 10^{-18}\,\mathrm{J}$ (= 27.21 eV = 1 Rydberg)
electrical field	$5.142 \cdot 10^{11}\,\frac{\mathrm{V}}{\mathrm{m}}$

23.1 Overview: Implemented Models

ionization regime	implemented model	reference
Multiphoton	None, yet	
Tunneling	• `Keldysh` • `ADKLinPol` • `ADKCircPol`	• [BauerMulser1999] • [DeloneKrainov] • [DeloneKrainov]
Barrier Suppression	• `BSI` • `BSIEffectiveZ` (R&D) • `BSIStarkShifted` (R&D)	• [MulserBauer2010] • [ClementiRaimondi1963] [ClementiRaimondi1967] • [BauerMulser1999]

Attention: Models marked with "(R&D)" are under *research and development* and should be used with care.

23.2 Ionization Current

In order to conserve energy, PIConGPU supports an ionization current to decrease the electric field according to the amount of energy lost to field ioniztion processes. The current for a single ion is

$$\mathbf{J}_{\text{ion}} = E_{\text{ion}} \frac{\mathbf{E}}{|\mathbf{E}|^2 \Delta t V_{\text{cell}}}$$

It is assigned to the grid according to the macroparticle shape. E_{ion} is the energy required to ionize the atom/ion, $\mathbf{E}$ is the electric field at the particle position and V_{cell} is the cell volume. This formula makes the assumption that the ejection energy of the electron is zero. See [Mulser]. The ionization current is accessible in *speciesDefinition.param*. To activate ionization current, set the second template of the ionization model to particles::ionization::current::EnergyConservation. By default the ionization current is deactivated.

23.3 Usage

Input for ionization models is defined in *speciesDefinition.param*, *ionizer.param and ionizationEnergies.param*.

23.4 Barrier Suppression Ionization

The so-called barrier-suppression ionization regime is reached for strong fields where the potential barrier binding an electron is completely suppressed.

23.5 Tunneling Ionization

Tunneling ionization describes the process during which an initially bound electron quantum-mechanically tunnels through a potential barrier of finite height.

23.5.1 Keldysh

$$\Gamma_{\text{K}} = \frac{(6\pi)^{1/2}}{2^{5/4}} E_{\text{ip}} \left(\frac{F}{(2E_{\text{ip}})^{3/2}} \right)^{1/2} \exp\left(-\frac{2\,(2E_{\text{ip}})^{3/2}}{3F} \right)$$

The Keldysh ionization rate has been implemented according to the equation (9) in [BauerMulser1999]. See also [Keldysh] for the original work.

Note: Assumptions:

- low field - perturbation theory
- $\omega_{\text{laser}} \ll E_{\text{ip}}$
- $F \ll F_{\text{BSI}}$
- tunneling is instantaneous

23.5.2 Ammosov-Delone-Krainov (ADK)

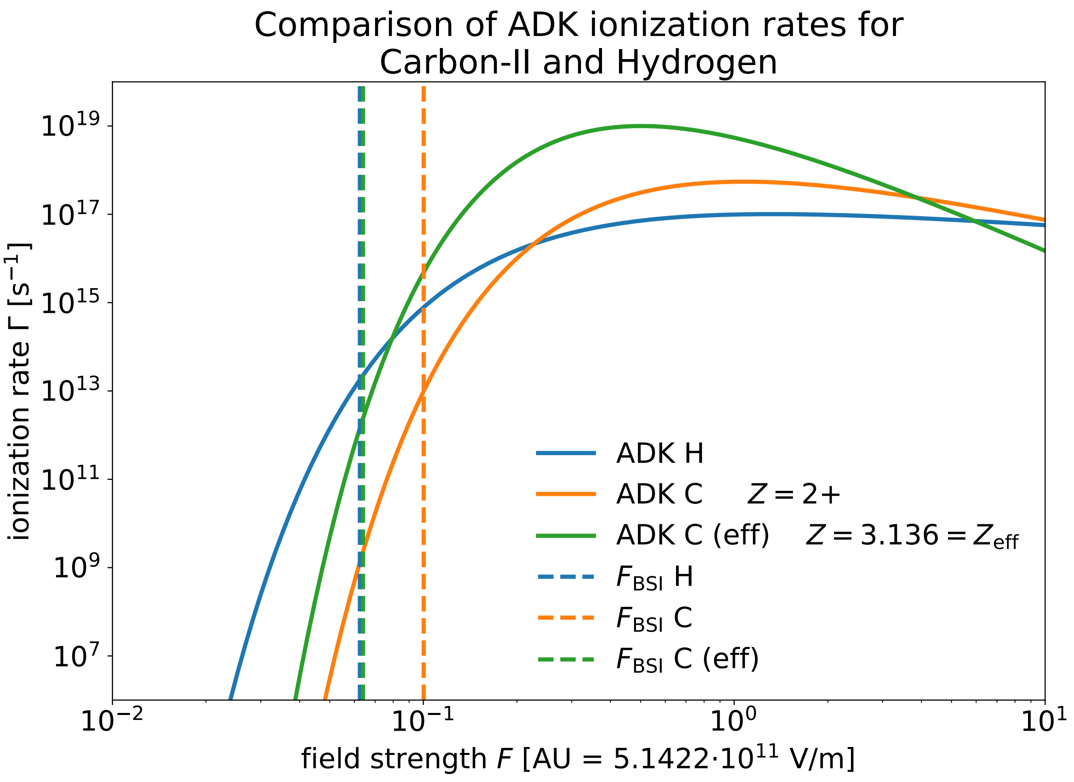
$$\Gamma_{\mathrm{ADK}} = \underbrace{\sqrt{\frac{3n^{*3}F}{\pi Z^3}}}_{\text{lin. pol.}} \frac{FD^2}{8\pi Z} \exp\left(-\frac{2Z^3}{3n^{*3}F}\right) \tag{23.1}$$

$$D \equiv \left(\frac{4\mathrm{e}Z^3}{Fn^{*4}}\right)^{n^*} \qquad n^* \equiv \frac{Z}{\sqrt{2E_{\mathrm{ip}}}} \tag{23.2}$$

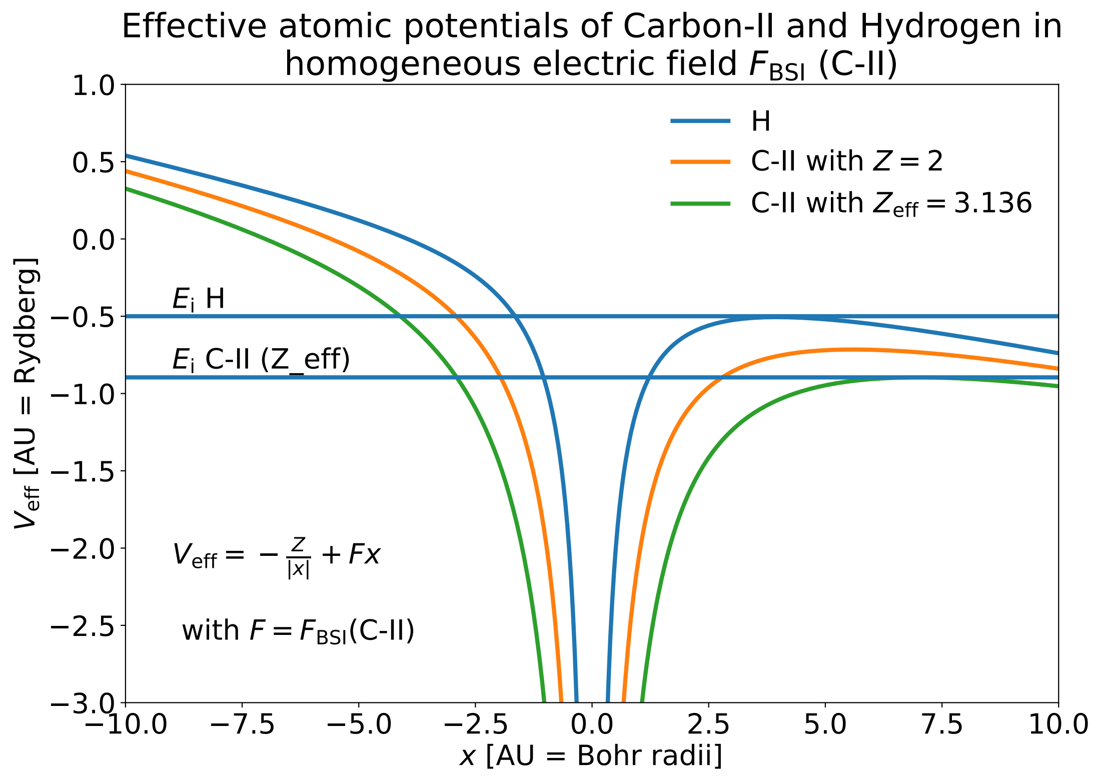
We implemented equation (7) from [DeloneKrainov] which is a *simplified result assuming s-states* (since we have no atomic structure implemented, yet). Leaving out the pre-factor distinguishes `ADKCircPol` from `ADKLinPol`. `ADKLinPol` results from replacing an instantaneous field strength F by $F\cos(\omega t)$ and averaging over one laser period.

Attention: Be aware that Z denotes the **residual ion charge** and not the proton number of the nucleus! Be aware that e in D denotes Euler's number, not the elementary charge

In the following comparison one can see the `ADKLinPol` ionization rates for the transition from Carbon II to III (meaning 1+ to 2+). For a reference the rates for Hydrogen as well as the barrier suppression field strengths F_{BSI} have been plotted. They mark the transition from the tunneling to the barrier suppression regime.



When we account for orbital structure in shielding of the ion charge Z according to [ClementiRaimondi1963] in `BSIEffectiveZ` the barrier suppression field strengths of Hydrogen and Carbon-II are very close to one another. One would expect much earlier ionization of Hydrogen due to lower ionization energy. The following image shows how this can be explained by the shape of the ion potential that is assumed in this model.



23.6 Predicting Charge State Distributions

Especially for underdense targets, it is possible to already give an estimate for how the laser pulse ionizes a specific target. Starting from an initially unionized state, calculating ionization rates for each charge state for a given electric field via a Markovian approach of transition matrices yields the charge state population for each time.

Here, we show an example of Neon gas ionized by a Gaussian laser pulse with maximum amplitude $a_0 = 10$ and pulse duration (FWHM intensity) of 30 fs. The figure shows the ionization rates and charge state population produced by the `ADKLinPol` model obtained from the pulse shape in the lower panel, as well as the step-like ionization produced by the `BSI` model.

You can test the implemented ionization models yourself with the corresponding module shipped in `picongpu/lib/python`.

```python
import numpy as np
import scipy.constants as sc
from picongpu.utils import FieldIonization

# instantiate class object that contains functions for
#   - ionization rates
#   - critical field strengths (BSI models)
#   - laser intensity conversion
FI = FieldIonization()

# dictionary with atomic units
AU = FI.atomic_unit

# residual charge state AFTER ionization
Z_H = 1
# hydrogen ionization energy (13.6 eV) converted to atomic units
E_H_AU = 13.6 * sc.electron_volt / AU['energy']
```

(continues on next page)

(continued from previous page)

```
# output: 0.50
print("%.2f" % (E_H_AU))
# barrier suppression threshold field strength
F_BSI_H = FI.F_crit_BSI(Z=Z_H, E_Ip=E_H_AU)
# output: 3.21e+10 V/m
print("%.2e V/m" % (F_BSI_H * AU['electric field']))
```

23.7 References

CHAPTER

TWENTYFOUR

COLLISIONAL IONIZATION

24.1 LTE Models

Module author: Marco Garten

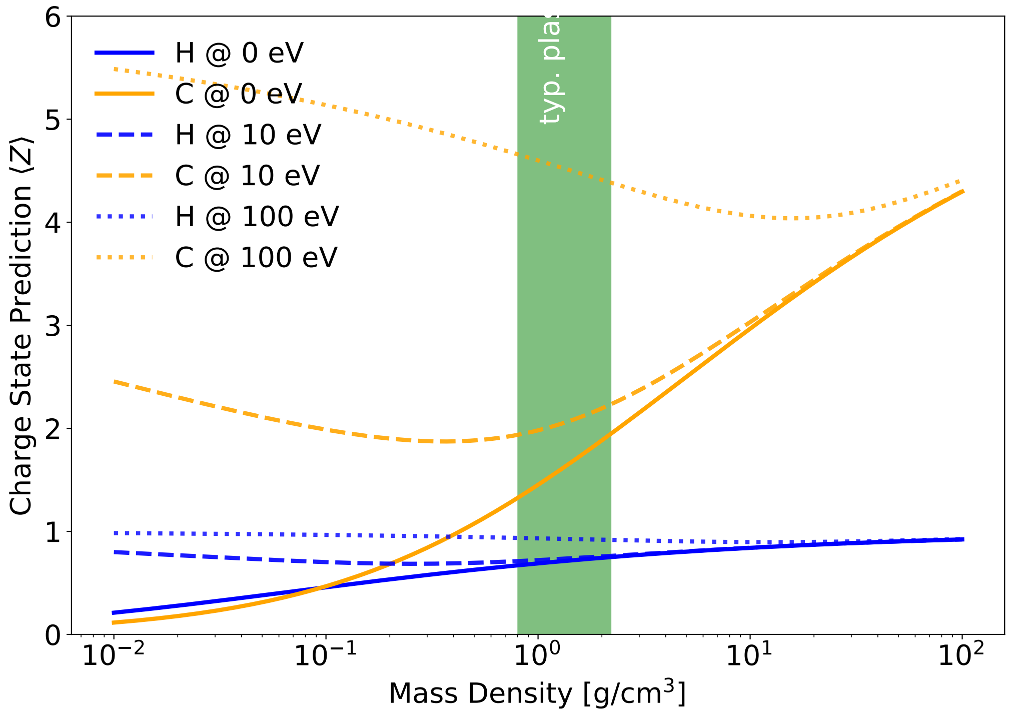
Implemented LTE Model: Thomas-Fermi Ionization according to [More1985]

Get started here https://github.com/ComputationalRadiationPhysics/picongpu/wiki/Ionization-in-PIConGPU

The implementation of the Thomas-Fermi model takes the following input quantities.

- ion proton number Z
- ion species mass density ρ
- electron "temperature" T

Due to the nature of our simulated setups it is also used in non-equilibrium situations. We therefore implemented additional conditions to mediate unphysical behavior but introduce arbitrariness.



Here is an example of hydrogen (in blue) and carbon (in orange) that we would use in a compound plastic target, for instance. The typical plastic density region is marked in green. Two of the artifacts can be seen in this plot:

1. Carbon is predicted to have an initial charge state $\langle Z \rangle > 0$ even at $T = 0\,\mathrm{eV}$.
2. Carbon is predicted to have a charge state of $\langle Z \rangle \approx 2$ at solid plastic density and electron temperature of $T = 10\,\mathrm{eV}$ which increases even as the density decreases. The average electron kinetic energy at such a temperature is 6.67 eV which is less than the 24.4 eV of binding energy for that state. The increase in charge state with decreasing density would lead to very high charge states in the pre-plasmas that we model.

1. Super-thermal electron cutoff

   We calculate the temperature according to $T_\mathrm{e} = \frac{2}{3} E_\mathrm{kin,e}$ in units of electron volts. We thereby assume an *ideal electron gas*. Via the variable `CUTOFF_MAX_ENERGY_KEV` in `ionizer.param` the user can exclude electrons with kinetic energy above this value from average energy calculation. That is motivated by a lower interaction cross section of particles with high relative velocities.

2. Lower ion-density cutoff

   The Thomas-Fermi model displays unphysical behaviour for low ion densities in that it predicts an increasing charge state for decreasing ion densities. This occurs already for electron temperatures of 10 eV and the effect increases as the temperature increases. For instance in pre-plasmas of solid density targets the charge state would be overestimated where

   - on average electron energies are not large enough for collisional ionization of a respective charge state
   - ion density is not large enough for potential depression
   - electron-ion interaction cross sections are small due to small ion density

   It is strongly suggested to do approximations for **every** setup or material first. To that end, a parameter scan with [FLYCHK2005] can help in choosing a reasonable value.

3. Lower electron-temperature cutoff

   Depending on the material the Thomas-Fermi prediction for the average charge state can be unphysically high. For some materials it predicts non-zero charge states at 0 temperature. That can be a reasonable approximation for metals and their electrons in the conduction band. Yet this cannot be generalized for all materials and therefore a cutoff should be explicitly defined.

   - define via `CUTOFF_LOW_TEMPERATURE_EV` in *ionizer.param*

24.2 NLTE Models

Module author: Axel Huebl

in development

CHAPTER

TWENTYFIVE

BINARY COLLISIONS

Section author: Pawel Ordyna

25.1 1 Introduction

The base PIC algorithm can't resolve interactions between particles, such as coulomb collisions, on a scale smaller than one cell. Taking them into consideration requires additional simulation steps. In many PIC software solutions collisions are introduced as binary collisions. With that approach every macro particle collides only with one other macro particle so that the computation time scales linearly with the number of particles, and not quadratically.

We have used the binary collisions algorithm introduced in *[1]* and updated it according to the corrections suggested in *[2]*. In that way, we were recreating the implementation from `smilei` *[3]*. We have also tested our code against `smilei` by running the test cases from `smilei`'s documentation with PIConGPU. We describe the algorithm and the underlying equations in detail in section *3*. Section *2* explains the usage of the collision feature. We show in section *4* the test simulations that we used to compare `PIConGPU` with `smilei`.

25.2 2 Usage

To enable collisions in your simulation you have to edit `collision.param`. There you can define the collision initialization pipeline. The collisions are set up with a `Collider` class. Each collider defines a list of species pairs that should collide with each other. A pair can consist of two different species, for collisions between two different particle groups, or two identical species, for collision within one group. The underlying collision algorithm is defined by a collision functor. At the moment, there are two collision functor classes available: `RelativisticCollisionConstLog` for running with a fixed coulomb logarithm, and `RelativisticCollisionDynamicLog` that calculates the coulomb log automatically. You can put in your initialization pipeline as many `Collider` objects as you need. Leaving the `CollisionPipeline` empty disables collisions.

25.2.1 2.1 Basics

Imagine you have two species in your simulation: `Ions` and `Electrons`. To enable collisions between the Ions and Electrons with a constant coulomb logarithm you would edit your param file in a following way:

```
namespace picongpu
{
    namespace particles
    {
        namespace collision
        {
            namespace precision
            {
                using float_COLL = float_64;
```

(continues on next page)

(continued from previous page)

```
            } // namespace precision

            constexpr bool debugScreeningLength = false;
            using CollisionScreeningSpecies = MakeSeq_t<>;

            struct Params
            {
                static constexpr float_X coulombLog = 5.0_X;
            };
            using Pairs = MakeSeq_t<Pair<Electrons, Ions>>;

            /** CollisionPipeline defines in which order species interact with each␣
↪other
             *
             * the functors are called in order (from first to last functor)
             */
            using CollisionPipeline = boost::mp11::
                mp_list<Collider<relativistic::RelativisticCollisionConstLog<Params>,␣
↪Pairs> >;

            /** Chunk size used for cell list allocations.
             *
             * To reduce the fragmentation of the heap memory on accelerators the␣
↪collision algorithm is allocating a
             * multiple of this value to store a cell list of particle IDs. The value␣
↪must be non zero.
             */
            constexpr uint32_t cellListChunkSize = particles::TYPICAL_PARTICLES_PER_
↪CELL;
        } // namespace collision
    } // namespace particles
} // namespace picongpu
```

Notice how the Coulomb logarithm is send to the functor class in a struct.

If you now would like to add internal collisions (electrons – electrons and ions – ions) you just need to extend the line 20 so that it looks like that:

```
using Pairs = MakeSeq_t<Pair<Electrons, Ions>, Pair<Electrons, Electrons>, Pair<Ions,␣
↪Ions>>;
```

But what if you don't want to have the same Coulomb logarithm for all collision types? For that you need more colliders in your pipeline. Here is an example with $\Lambda = 5$ for electron-ion collisions and $\Lambda = 10$ for electron-electron and ion-ion collisions.

```
struct Params1
{
   static constexpr float_X coulombLog = 5.0_X;
};
struct Params2
{
   static constexpr float_X coulombLog = 10.0_X;
};
using Pairs1 = MakeSeq_t<Pair<Electrons, Ions>>;
using Pairs2 = MakeSeq_t<Pair<Electrons, Electrons>, Pair<Ions, Ions>>;
using CollisionPipeline =
    boost::mp11::mp_list<
```

(continues on next page)

(continued from previous page)

```
        Collider<relativistic::RelativisticCollisionConstLog<Params1>, Pairs1>,
        Collider<relativistic::RelativisticCollisionConstLog<Params2>, Pairs2>
    >;
```

Automatic coulomb log calculation can be enabled for a collider by changing the collision functor. For example the previous setup with automatic calculation for the inter-species collisions would have the following `CollisionPieline` (and Params1 is not longer needed)

```
using CollisionPipeline =
    boost::mp11::mp_list<
        Collider<relativistic::RelativisticCollisionDynamicLog<>, Pairs1>,
        Collider<relativistic::RelativisticCollisionConstLog<Params2>, Pairs2>
    >;
```

The dynamic logarithm implementation uses a Debye length that is pre-calculated once for all colliders on each time step. So, whenever there is at least one collider with the `RelativisticCollisionDynamicLog` present in the `CollisionPipeline` this precalculation needs to be enabled by adding Species to the `CollisionScreeningSpecies` sequence. To include all species just set `using CollisionScreeningSpecies = VectorAllSpecies;`. But, this can be an arbitrary list of (filtered) species, see the `CollisionsBeamRelaxation` test for reference.

Note: The Debye length calculation requires at least 2 `FieldTmp` slots enabled in `memory.param` when `CollisionScreeningSpecies` has only one element and at least 3 otherwise.

25.2.2 2.2 Particle filters

You can also use particle filters to further refine your setup. The `Collider` class can take one more, optional, template argument defining a pair of particle filters. Each filter is applied respectively to the first and the second species in a pair. You need to define your filters in `particleFilters.param` and than you can use them, for example, like that:

```
using Pairs1 = MakeSeq_t<Pair<Electrons, Ions>>;
using Pairs2 = MakeSeq_t<Pair<Electrons, Electrons>, Pair<Ions, Ions>>;
using CollisionPipeline =
    boost::mp11::mp_list<
        Collider<
            relativistic::RelativisticCollisionConstLog<Params1>,
            Pairs1,
            FilterPair<filter::FilterA, filter::FilterB>>,
        Collider<
            relativistic::RelativisticCollisionConstLog<Params2>,
            Pairs2,
            OneFilter<filter::FilterA>>
        >;
```

Here only the electrons passing the A-filter will collide with ions but only with the ions that pass the B-filter. If the filters are identical you can use `OneFilter` instead of `FilterPair`. For collisions within one species the filters in `FilterPair` **have** to be identical since there is only one particle group colliding.

A full functional example can be found in the `CollisionsBeamRelaxation` test, where particle filters are used to enable each of the three colliders only in a certain part of the simulation box.

25.2.3 2.3 Precision

Highly relativistic particles can cause numerical errors in the collision algorithm that result in NaN values. To avoid that, by default, all the kinematics of a single binary collision is calculated in the 64 bit precision, regardless of the chosen simulation precision. Until now, this has been enough to avoid NaNs but we are looking into better solutions to this problem. You can change this setting by editing the

```
using float_COLL = float_64;
```

line. You can set it to

```
using float_COLL = float_X;
```

to match the simulation precision or

```
using float_COLL = float_32;
```

for explicit single precision usage. If you use PIConGPU with the 32 bit precision, lowering the collision precision will speed up your simulation and is recommended for non–relativistic setups.

25.2.4 2.4 Debug output

It is possible to write the average coulomb logarithm and s parameter (see *3.4 Details on the s parameter*) values (averaged over all collisions in a time-step) for each collider. This debug output can be enabled per collider by setting the optional template parameter of the collision functor to true:

```
using CollisionPipeline =
    boost::mp11::mp_list<
        Collider<relativistic::RelativisticCollisionDynamicLog<true>, Pairs1>,
        Collider<relativistic::RelativisticCollisionConstLog<Params2, true>, Pairs2>
    >;
```

The debug info is written to a text file `debug_values_collider_<collider index in the pipeline>_species_pair_<pair index in the list of pairs used with the collider>.dat` The output file has three columns: iteration, coulomb log, s param. It it also possible to write out the precalculated Debye length averaged over all simulation cells by setting `constexpr bool debugScreeningLength = true;` The output is written to a file with two columns: iteration, Debye length [m]. The file name is `average_debye_length_for_collisions.dat`.

25.3 3 Algorithm

25.3.1 3.1 Algorithm overview

A short summary of the important algorithm steps in the case of inter-species collisions. The case of intra-collisions is very similar. See figures Fig. 25.1, Fig. 25.3, Fig. 25.2, Fig. 25.4 for more details.

1. Sort particles from a super cell into particle lists, one list for each grid cell.
2. In each cell, shuffle the list with more particles.
3. Collide each particle from the first longer list with a particle from the shorter one (or equally long). When you run out of particles in the shorter list, start from the beginning of that list and collide some particles more than once.
    1. Determine how many times the second particle will be collided with some particle from the longer list (in the current simulation step).
    2. Read particle momenta.

3. Change into the center of mass frame.
4. Calculate the s parameter.
5. Generate a random azimuthal collision angle $\varphi \in (0, 2\pi]$.
6. Get the cosine of the 2nd angle θ from its probability distribution (depends on s).
7. Use the angles to calculate the final momenta (in the COM frame).
8. Get the new momenta into the lab frame.
9. Apply the new momentum to the macro particle A (smaller weighting).
   Do the same for the macro particle B (bigger weighting) but with a probability equal to the weighting ratio of the particles A and B.

4. Free up the memory used for the particle lists.

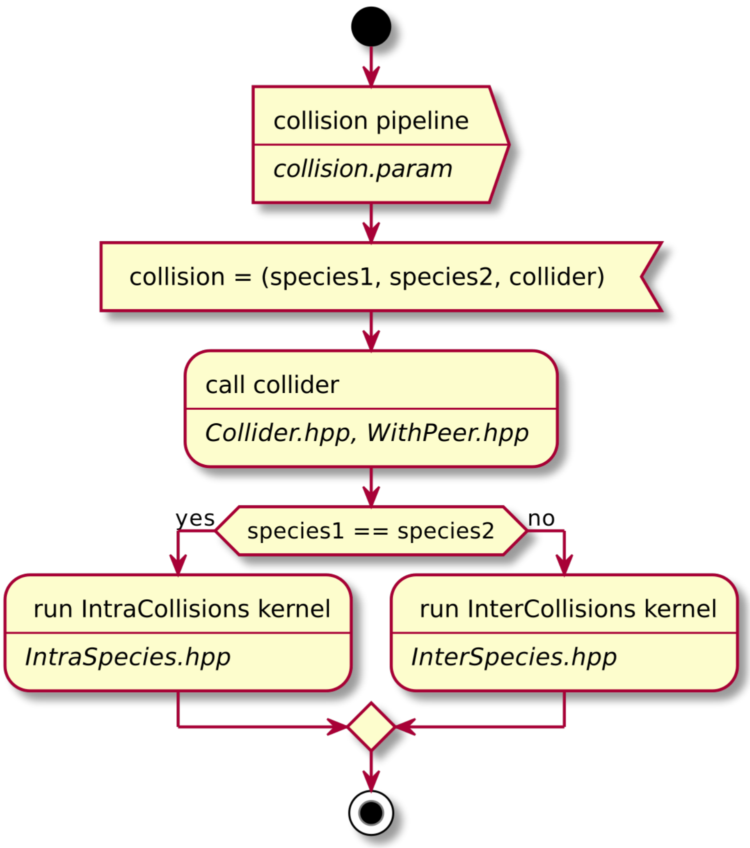

collision pipeline
collision.param
collision = (species1, species2, collider)
call collider
Collider.hpp, WithPeer.hpp
yes
species1 == species2
no
run IntraCollisions kernel
IntraSpecies.hpp
run InterCollisions kernel
InterSpecies.hpp

Fig. 25.1: Flow chart showing the complete algorithm. For more detail on intra-collisions see fig. Fig. 25.2, for more details on inter-collisions see fig. Fig. 25.3. Numbers in brackets refer to equations other to sections.

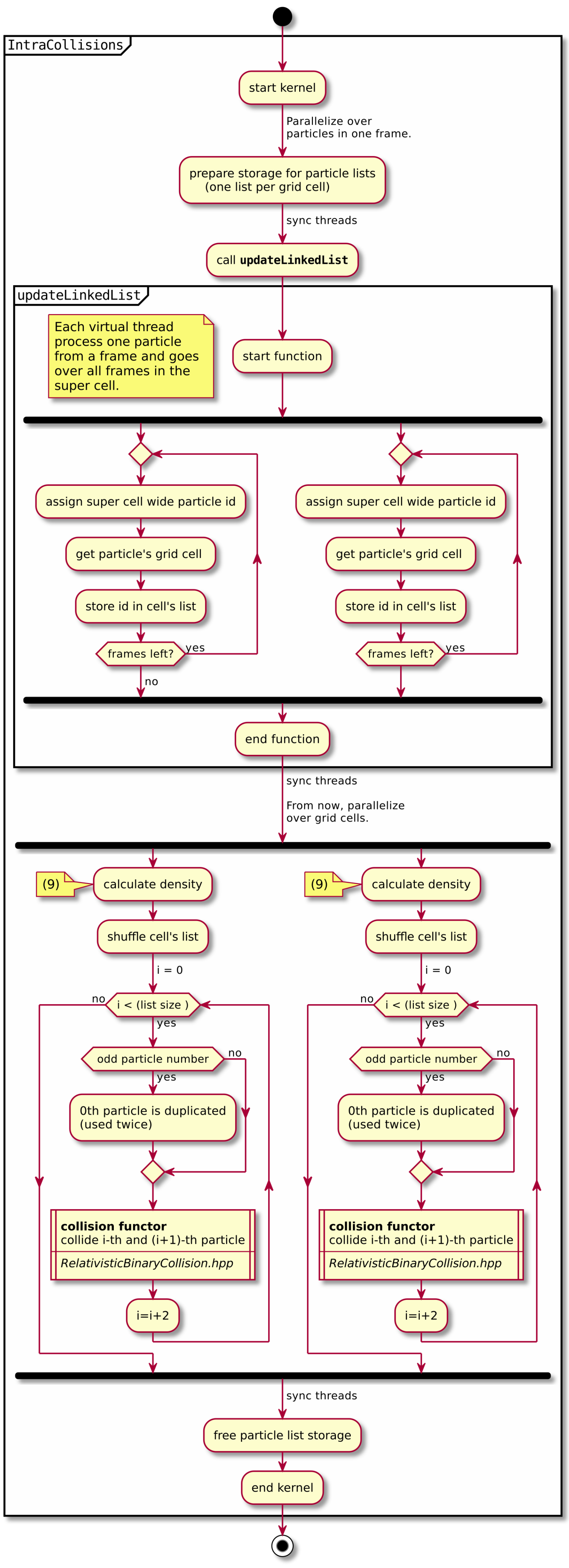

IntraCollisions
start kernel
Parallelize over particles in one frame.
prepare storage for particle lists (one list per grid cell)
sync threads
call updateLinkedList
updateLinkedList
Each virtual thread process one particle from a frame and goes over all frames in the super cell.
start function
assign super cell wide particle id
get particle's grid cell
store id in cell's list
frames left?
yes
no
assign super cell wide particle id
get particle's grid cell
store id in cell's list
frames left?
yes
end function
sync threads
From now, parallelize over grid cells.
(9)
calculate density
shuffle cell's list
i = 0
i < (list size)
no
yes
odd particle number
no
yes
0th particle is duplicated (used twice)
collision functor
collide i-th and (i+1)-th particle
RelativisticBinaryCollision.hpp
i=i+2
(9)
calculate density
shuffle cell's list
i = 0
i < (list size)
no
yes
odd particle number
no
yes
0th particle is duplicated (used twice)
collision functor
collide i-th and (i+1)-th particle
RelativisticBinaryCollision.hpp
i=i+2
sync threads
free particle list storage
end kernel

Fig. 25.2: Flow chart showing the part of the collision algorithm that is unique for intra-collisions. For more details on collisions functor see fig. Fig. 25.4 . Numbers in brackets refer to equations other to sections.

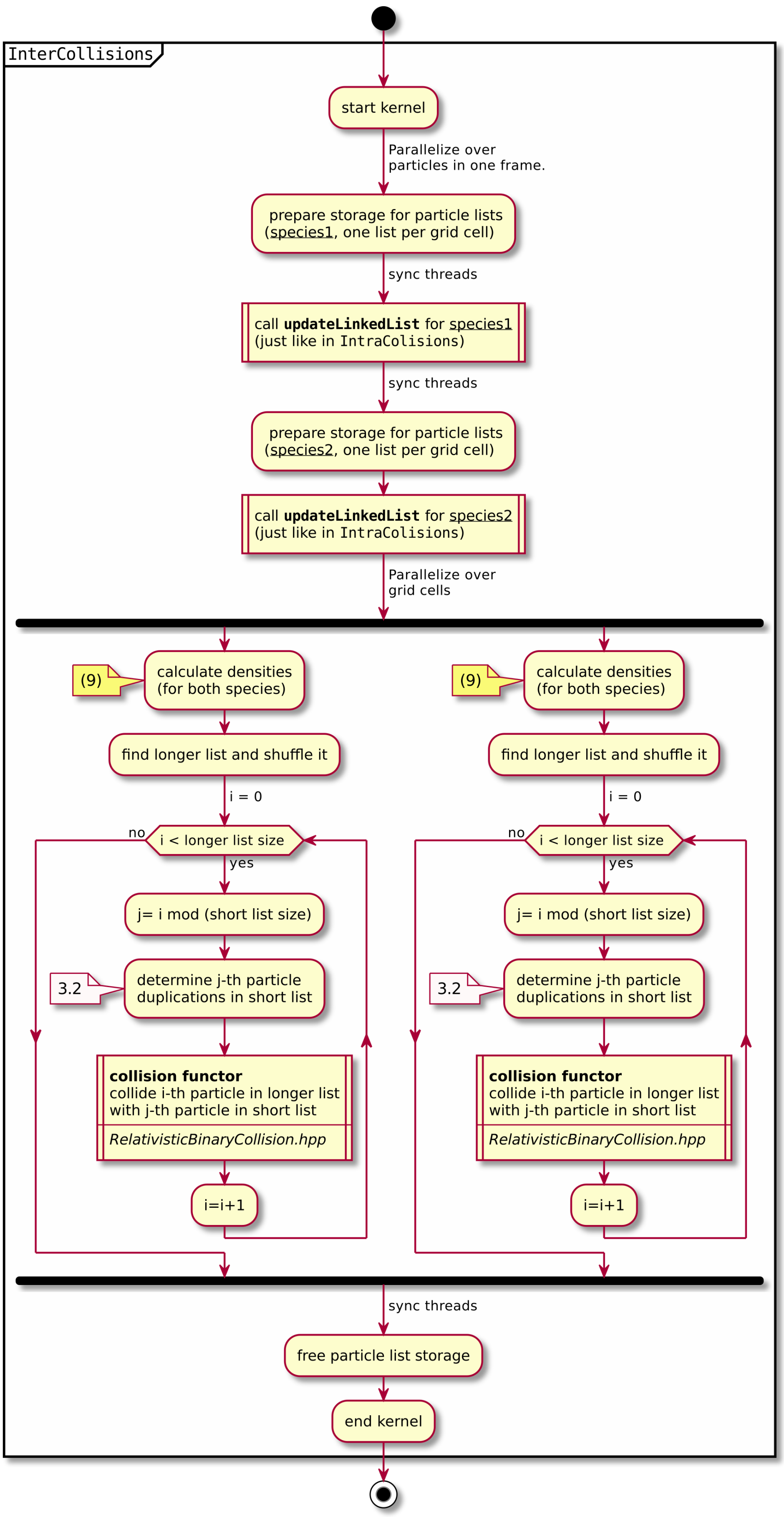

InterCollisions
start kernel
Parallelize over particles in one frame.
prepare storage for particle lists (species1, one list per grid cell)
sync threads
call **updateLinkedList** for species1 (just like in IntraColisions)
sync threads
prepare storage for particle lists (species2, one list per grid cell)
call **updateLinkedList** for species2 (just like in IntraColisions)
Parallelize over grid cells
(9)
calculate densities (for both species)
find longer list and shuffle it
i = 0
no
i < longer list size
yes
j= i mod (short list size)
3.2
determine j-th particle duplications in short list
collision functor collide i-th particle in longer list with j-th particle in short list
RelativisticBinaryCollision.hpp
i=i+1
(9)
calculate densities (for both species)
find longer list and shuffle it
i = 0
no
i < longer list size
yes
j= i mod (short list size)
3.2
determine j-th particle duplications in short list
collision functor collide i-th particle in longer list with j-th particle in short list
RelativisticBinaryCollision.hpp
i=i+1
sync threads
free particle list storage
end kernel

Fig. 25.3: Flow chart showing the part of the collision algorithm that is unique for inter-collisions. Numbers in brackets refer to equations other to sections.

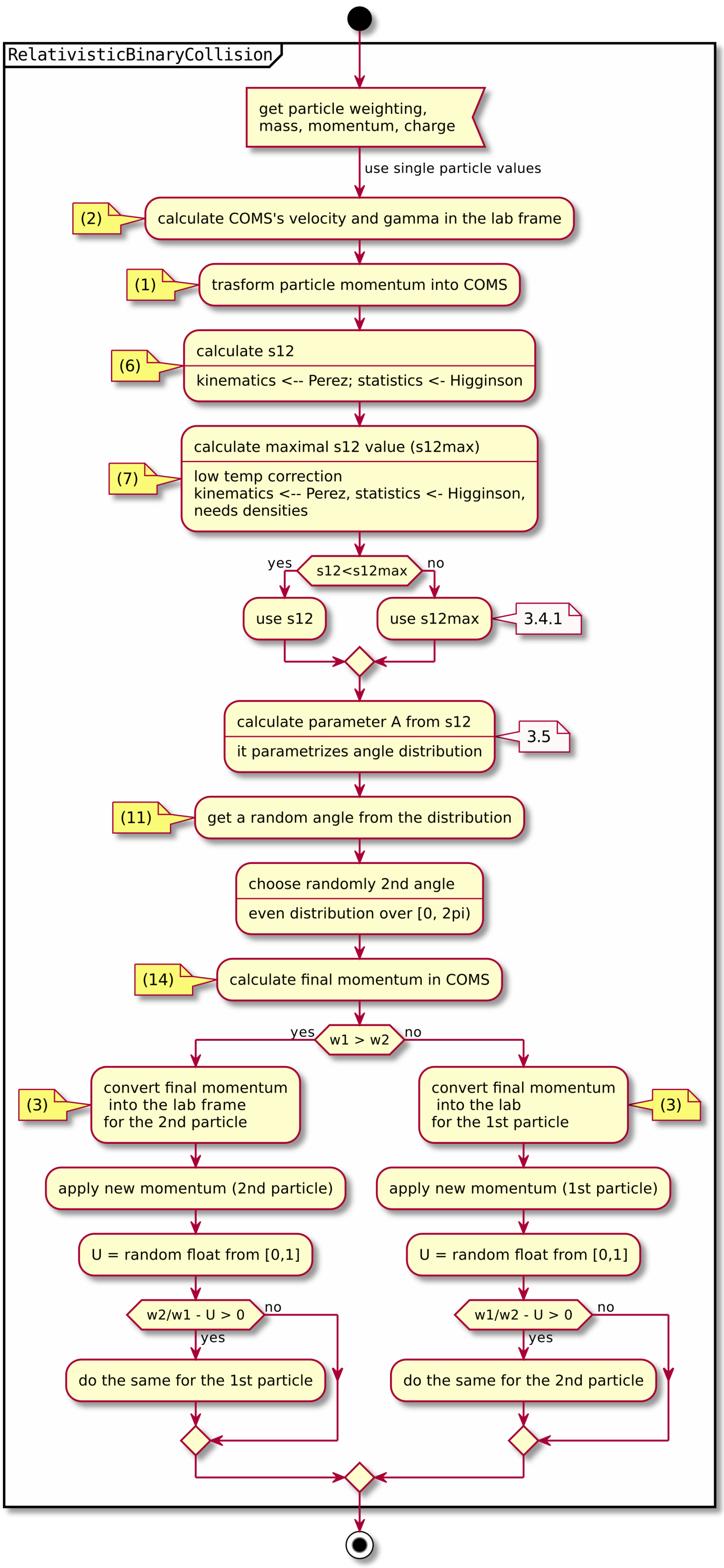

RelativisticBinaryCollision
get particle weighting, mass, momentum, charge
use single particle values
(2)
calculate COMS's velocity and gamma in the lab frame
(1)
trasform particle momentum into COMS
(6)
calculate s12
kinematics <-- Perez; statistics <- Higginson
(7)
calculate maximal s12 value (s12max)
low temp correction
kinematics <-- Perez, statistics <- Higginson,
needs densities
yes
s12<s12max
no
use s12
use s12max
3.4.1
calculate parameter A from s12
it parametrizes angle distribution
3.5
(11)
get a random angle from the distribution
choose randomly 2nd angle
even distribution over [0, 2pi)
(14)
calculate final momentum in COMS
yes
w1 > w2
no
(3)
convert final momentum into the lab frame for the 2nd particle
convert final momentum into the lab for the 1st particle
(3)
apply new momentum (2nd particle)
apply new momentum (1st particle)
U = random float from [0,1]
U = random float from [0,1]
w2/w1 - U > 0
no
yes
w1/w2 - U > 0
no
yes
do the same for the 1st particle
do the same for the 2nd particle

Fig. 25.4: Flow chart showing the `RelativisticBinaryCollision` collisions functor. Numbers in brackets refer to equations other to sections.

25.3.2 3.2 Details on macro particle duplication

First step that requires some more detailed explanation is the step 3.1 . In a situation where there are less macro particles, inside one cell, of one species than the other one not every macro particle has its collision partner. Similar problem emerges in a case of intra-collisions when the particle number is odd. We deal with that issue using an approach introduced in *[2]*. We collide, in such situation, some macro particles more than once. To account for that, we use corrected particle weights $w_{0/1} = \frac{1}{\max d_0, d_1}$, where $d_{0/1}$ are the number of collisions for the colliding macro particles.

Let us consider the inter-collisions first. The i–th particle from the longer list is collided with the $(i \mod m)$ –th particle in the shorter one (m is the length of the shorter list). All of the particles from the longer list will collide just once. So the correction for each binary collision is $1/d$ of the particle from the shorter list. d is determined in the following way:

```
d = floor(n / m);
if (i % m ) < (n % m) d = d + 1;
```

i – particle index in the long list, n – long list length, m – short list length, d – times the particle from the shorter list is used in the current step.

In the intra-collisions, the i–th (i is odd) particle collides with the $i + 1$–th one. When there is, in total, an odd number of particles to collide, the first particle on the list collides twice. At first it is collided with the second one and in the end with the last one. All other particles collide once. So d will be 2 for the first collision (1st with 2nd particle) and for the last one (n-th with 1st particle). For the other collisions it's 1.

25.3.3 3.3 Details on the coordinate transform

A binary collision is calculated in this model in the center of mass frame. A star * denotes a COMS variable.

We use the coordinate transform from *[1]*:

$$\mathbf{p}^* = \mathbf{p}_{\text{lab}} + (\frac{\gamma_C - 1}{|\mathbf{v}_C|^2}\mathbf{v}_C \cdot \mathbf{v}_{\text{lab}} - \gamma_C)m\gamma\mathbf{v}_C \ , \tag{25.1}$$

where $\mathbf{v}_C$ is the velocity of the CMOS in the lab frame, γ is the [list::duplications] factor in the lab frame, m the particle mass and γ_C the gamma factor of the CMOS frame.

$$\mathbf{v}_C = \frac{\mathbf{p}_{\text{lab},0} + \mathbf{p}_{\text{lab},1}}{m_0\gamma_0 + m_1\gamma_1} \tag{25.2}$$

The inverse transformation:

$$\mathbf{p}_{\text{lab}} = \mathbf{p}^* + (\frac{\gamma_C - 1}{|\mathbf{v}_C|^2}\mathbf{v}_C \cdot \mathbf{p}^* + m\gamma^*\gamma_C)\mathbf{v}_C \ , \tag{25.3}$$

where

$$\gamma^* = \gamma_C\gamma(1 - \frac{\mathbf{v}_C \cdot \mathbf{v}_{\text{lab}}}{c^2}) \ . \tag{25.4}$$

25.3.4 3.4 Details on the s parameter

$$s = \frac{1}{2}N\left\langle\theta^{*2}\right\rangle \tag{25.5}$$

N is the number of real collisions. It's the number of small angle collisions of a test particle represented by one of the macro particles with all the potential collision partners in a cell (here real particles not macro particles) in the current time step assuming the relative velocity is the one of the two colliding macro particles. $\left\langle\theta^{*2}\right\rangle$ is the averaged squared scattering angle for a single collision (of real particles). According to *[2]* s is a normalized path length.

To calculate this parameter we use the relativistic formula from *[1]* and adjust it so it fits the new corrected algorithm from *[2]*.

$$\begin{aligned} s_{01} =& \frac{\Delta T \log \Lambda q_0^2 q_1^2}{4\pi\varepsilon_0^2 c^4 m_0\gamma_0 m_1\gamma_1} \\ & \times \frac{\gamma_C \left|\mathbf{p}_0^*\right|}{m_0\gamma_0 + m_1\gamma_1} (m_0\gamma_0^* m_1\gamma_1^* c^2 \left|\mathbf{p}_0^*\right|^{-2} + 1)^2 \\ & \times N_{\text{partners}} V_{\text{cell}}^{-1} \max \frac{w_0}{d}, \frac{w_1}{d} \,. \end{aligned} \tag{25.6}$$

Here: ΔT – time step duration, $\log\Lambda$ – Coulomb logarithm, q_0, q_1 – particle charges, γ_0, γ_1 particles gamma factors(lab frame), N_{partners} is the number of collision partners (macro particles), V_{cell} – cell volume, w_0, w_1 particle weightings, d was defined in *3.2*.

For inter-species collisions N_{partners} is equal to the size of the long particle list. For intra-species collisions N_{partners} = $n - 1 + (n \mod 2)$,where n is the number of macro particles to collide.

The fact that s_{01} depends only on the higher weighting is accounted for by the rejection method in the 3.9 step.

3.4.1 Low temperature limit

According to *[1]* equation (25.6) will provide non physical values for low temperatures. More specifically, it will result in s values corresponding to scattering lengths smaller than the average particle distance $(\frac{V}{n})^{\frac{1}{3}}$. *[1]* provides a maximal value for s_{01}:

$$\begin{aligned} s_{01}^{\max} =& (\frac{4\pi}{3})^{1/3} \frac{\Delta T (m_0 + m_1)}{\max m_0 n_0^{2/3}, m_1 n_1^{2/3}} \mathbf{v}_{\text{rel}}^* \\ & \times N_{\text{partners}} V_{\text{cell}}^{-1} \max \frac{w_0}{d}, \frac{w_1}{d} \,. \end{aligned} \tag{25.7}$$

with

$$\mathbf{v}_{\text{rel}}^* = \frac{(m_1\gamma_1 + m_2\gamma_2) p_1^*}{m_1\gamma_1^* m_2\gamma_2^* \gamma_C} \,. \tag{25.8}$$

where the relativistic factor $(1 + v_1^* v_2^*/c^2)^{-1}$ has been left out.

For each binary collision both values are calculated and the smallest one is used later. The particle density is just the sum of all particle weightings from one grid cell divided by cell volume

$$n = \frac{1}{V_{\text{cell}}} \sum_i w_i \,. \tag{25.9}$$

Note: It is not checked if the collision is really non-relativistic. If the low temp limit is smaller than s_{01} due to some other reason, e.g. an overflow in s_{01} calculation, the code will use this limit regardless of the particle being relativistic or not which could be physically incorrect.

25.3.5 3.5 Details on the scattering angle distribution

The distribution for the cumulative angle χ as a function of s was introduced in *[4]*

$$F(\chi) = \frac{A(s)\sin\chi}{2\sinh A(s)} e^{A(s)\cos\chi} \,. \tag{25.10}$$

We obtain a random value for the cosine from F with

$$\cos\chi = A^{-1} \ln(e^{-A} + 2U \sinh A) \,, \tag{25.11}$$

where U is a random float between 0 and 1. The parameter A is obtained by solving

$$\coth A - A^{-1} = e^{-s} \quad . \tag{25.12}$$

Previously the algorithm was approximating A with a polynomial fit from *[1]*. Now the $\cos\chi$ is obtained from a new fit that was introduced in smilei:

If s < 4 then:

$$\alpha = 0.37s - 0.005s^2 - 0.0064s^3 \; . \tag{25.13}$$

$$\sin^2(x/2) = \frac{\alpha U}{\sqrt{(1-U) + \alpha^2 U}} \tag{25.14}$$

$$\cos(x) = 1 - 2\sin^2(x/2) \tag{25.15}$$

In the $s \to \infty$ limit scattering becomes isotropic *[4]* so that we can take $\cos\chi = 2U - 1$ for $s > 4$.

25.3.6 3.6 Details on final momentum calculation

The final particle momenta in the COMS frame are calculated with the following formula from *[1]*

$$\mathbf{p}_{1f}^* = -\mathbf{p}_{2f}^* = \begin{pmatrix} \frac{p_{1x}^* p_{1z}^*}{p_{1\perp}^*} & \frac{p_{1y}^* p_1^*}{p_{1\perp}^*} & p_{1x}^* \\ \frac{p_{1y}^* p_{1z}^*}{p_{1\perp}^*} & \frac{p_{1x}^* p_1^*}{p_{1\perp}^*} & p_{1y}^* \\ -p_{1\perp}^* & 0 & p_{1z}^* \end{pmatrix} \cdot \begin{pmatrix} \sin\theta^* \cos\varphi^* \\ \sin\theta^* \sin\varphi^* \\ \cos\theta^* \end{pmatrix} \quad . \tag{25.16}$$

25.3.7 3.7 Dynamic Coulomb logarithm calculation

With the `RelativisticCollisionDynamicLog` functor the Coulomb logarithm is calculated individually for each collision following a formula from *[1]*:

$$\ln\Lambda = \max[2, \frac{1}{2}\ln(1 + \frac{\lambda_D^2}{b_{\text{min}}^2})] \; ,$$

where b_{min} is a minimal impact parameter that depends on particle momenta, charges, and masses; and λ_D is the Debye length.

Please note, according to the `smilei` documentation, in the equation (22) in *[1]* for b_{min} the last factor should not be squared; we drop the square also in PIConGPU.

The Debye length is calculated once per time-step for each simulation cell using the formula:

$$\lambda_D^{-2} = \frac{1}{\epsilon_0}\sum_\alpha n_\alpha \left\langle q_\alpha \right\rangle^2 / T_\alpha \; ,$$

where the sum goes over all charged particle species, n is the number density, $\langle q \rangle$ is the average charge, and T is the temperature. The temperature is assumed to be equal to $\frac{2}{3}\left\langle E_{\text{kin}} \right\rangle$.

In PIConGPU the contributions from each species are calculated as

$$\frac{2}{3\epsilon_0}\rho^2\varepsilon^{-1} \; ,$$

where ρ is the charge density and ε is the energy density. It can be shown that this is equal to $\frac{1}{\epsilon_0} n < q >^2 /T$.

Additionally λ_D is cut-off at the mean interatomic distance of the species with the highest density:

$$\lambda_D \geq (4\pi n_{\text{max}}/3)^{-1/3}$$

25.4 4 Tests

For testing we plan to reproduce all the test cases from `smilei`'s documentation(https://smileipic.github.io/Smilei/collisions.html). For now we have done the thermalization and the beam relaxation tests. The simulations that we used are available under `share/picongpu/tests`.

25.4.1 4.1 Thermalization

In this example there are two particle populations — electrons and ions. They are thermally initialized with different temperatures and their temperatures get closer to each other with time. The usual PIC steps are disabled (there is no field solver and no pusher). The thermalization happens solely due to the binary collisions. We enable inter-collisions for ions and electrons as well as collisions between the two species. Simulation parameters are listed in table Table 25.1. The species temperatures are calculated in post processing from an `openPMD` output, a python script used for the analysis can be found in the tests directory. We extended the tests by running the simulation for longer and also by including runs using the automatic coulomb log calculation instead of a constant value.

The results from the original setup with constant logarithm are shown in fig. Fig. 25.5, Fig. 25.6, Fig. 25.7 for three different macro particle weight ratios. The figures Fig. 25.8, Fig. 25.9, Fig. 25.10 show the results from a set-up with dynamic coulomb logarithm calculation for the electron-ion collisions, and fig. Fig. 25.11, Fig. 25.12, and Fig. 25.13 come from a set-up with automatic logarithm for all three collision pairs. The theoretical curves are obtained from the same formula that was used by `smilei`'s developers and originates from the NRL plasma formulary *[5]*.

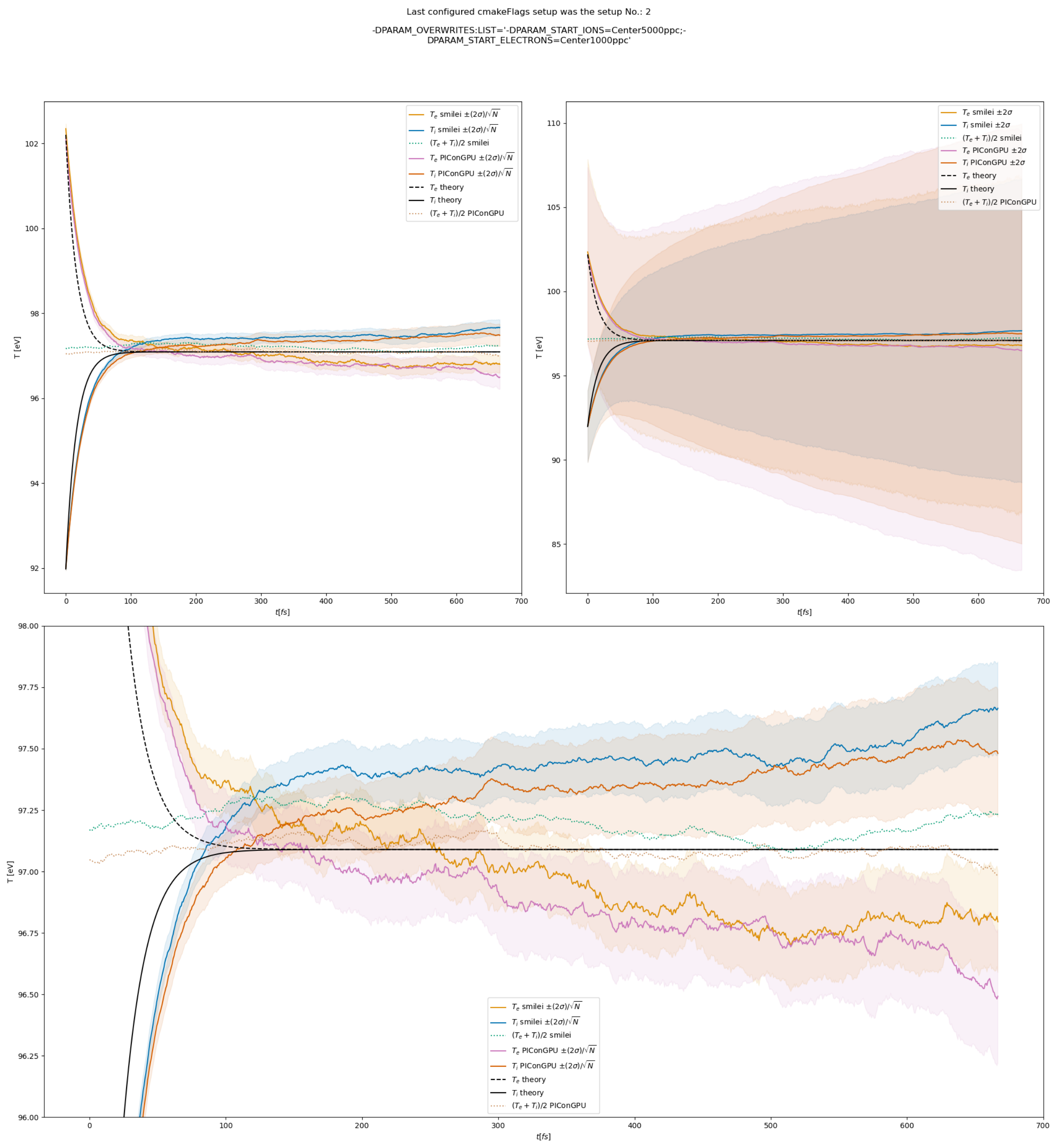
$$\frac{\mathrm{d}T_\alpha}{\mathrm{d}t} = \nu_\epsilon (T_\beta - T_\alpha)$$

$$\nu_\epsilon = \frac{2}{3}\sqrt{\frac{2}{\pi}}\frac{e^4\, Z^{\star 2}\sqrt{m_e m_i}\, n_i\, \ln\Lambda}{4\pi\varepsilon_0^2\,(m_e T_e + m_i T_i)^{3/2}}$$

Since the collisions in different cells are independent of each other, one can treat each cells as an individual randomized run. The simulation values are obtained by averaging over the individual simulation cells. The upper right panel shows the average values together with ± 2 standard deviation of the distribution, while the left panel is showing the same values with ± 2 standard deviation of the mean. The bottom panel is just a zoom in on the upper left panel. The inputs for the `smilei` runs can be found in `smilei_thermalization_inputs.zip`.

Additionally, figures Fig. 25.5, Fig. 25.6, Fig. 25.7 show the average values of the coulomb logarithm, the s paramter, and the debye length. The coulomb logarithm theoretical value (blue curve) is calculated with the following formula from *[5]*:

$$\lambda_{ei} = 24 - \ln(n_e^{1/2} T_e^{-1})$$


Last configured cmakeFlags setup was the setup No.: 2

-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center5000ppc;-DPARAM_START_ELECTRONS=Center1000ppc'

Fig. 25.5: Electron (blue) and ion (red) temperature over time in the thermalization test with both $\ln \Lambda$ constant. The electron to ion weight ratio in the simulation is 1:5. Black lines are the the theoretical curves.

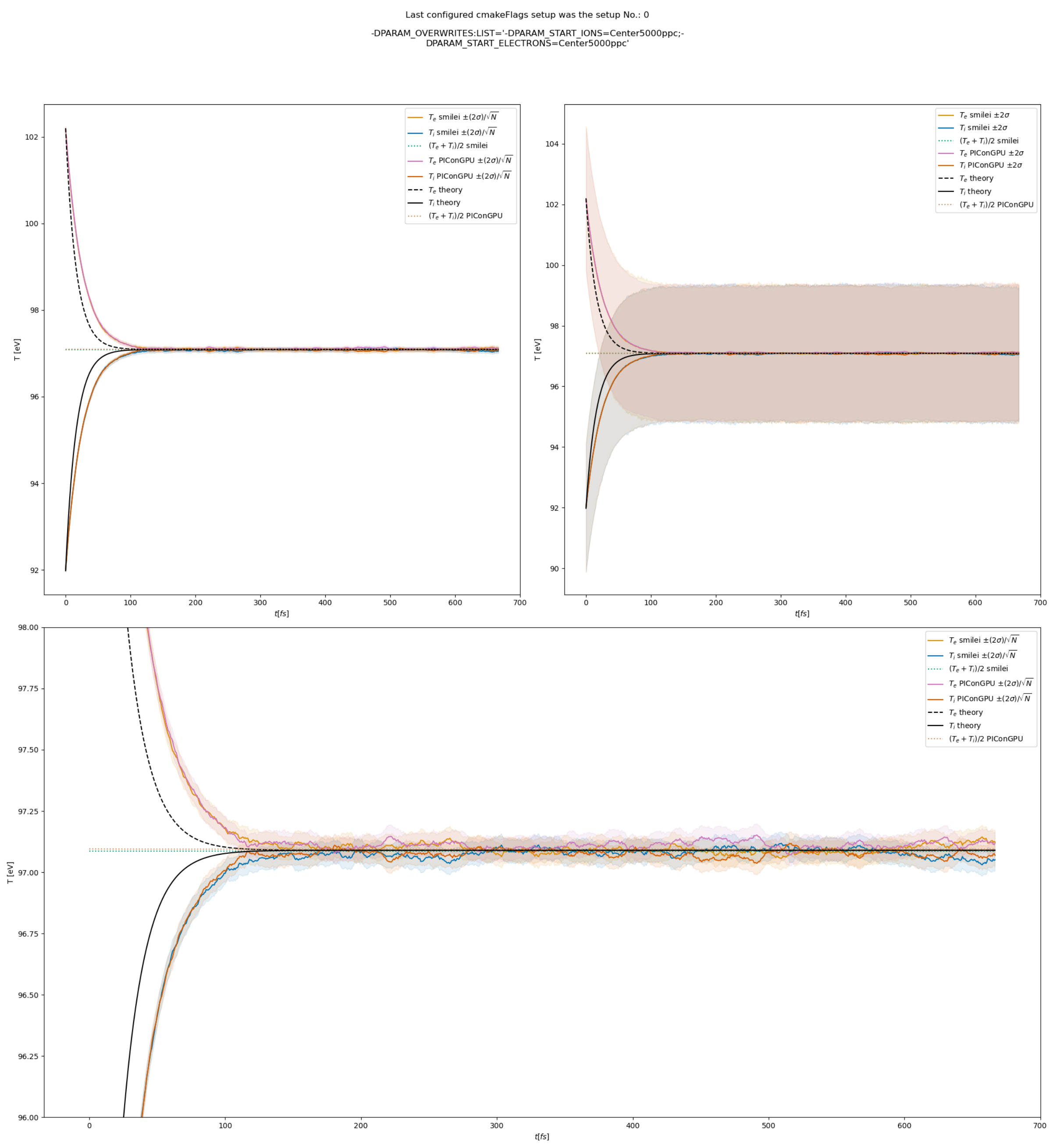
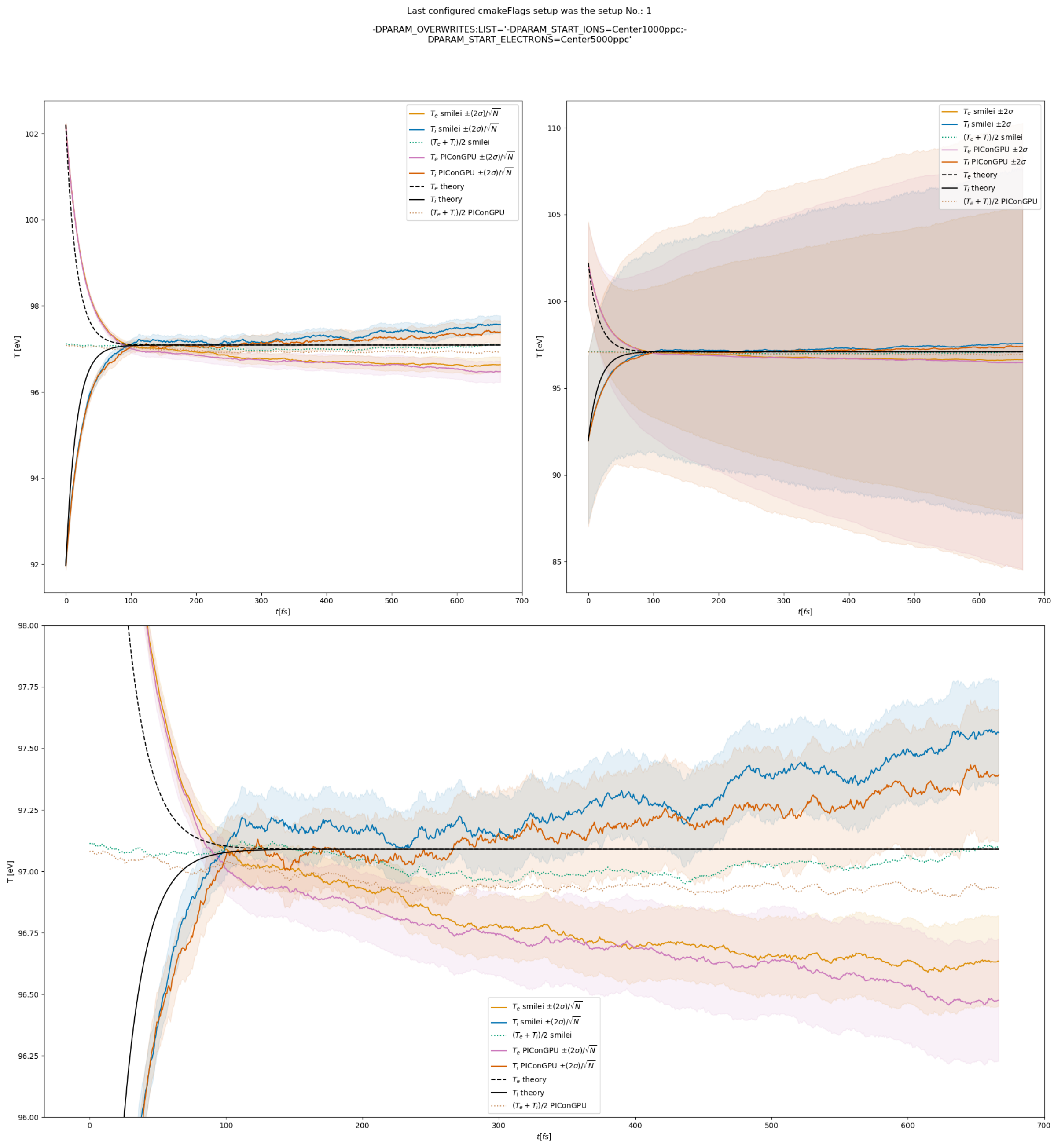


Fig. 25.6: Electron (blue) and ion (red) temperature over time in the thermalization test with both $\ln\Lambda$ constant. The electron to ion weight ratio in the simulation is 1:1. Black lines are the the theoretical curves.


Last configured cmakeFlags setup was the setup No.: 1

-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center1000ppc;-DPARAM_START_ELECTRONS=Center5000ppc'

Fig. 25.7: Electron (blue) and ion (red) temperature over time in the thermalization test with both $\ln \Lambda$ constant. The electron to ion weight ratio in the simulation is 5:1. Black lines are the the theoretical curves.

Last configured cmakeFlags setup was the setup No.: 8

-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center5000ppc;-DPARAM_START_ELECTRONS=Center1000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-DPARAM_DYNAMIC_LOG_INTRA=0'

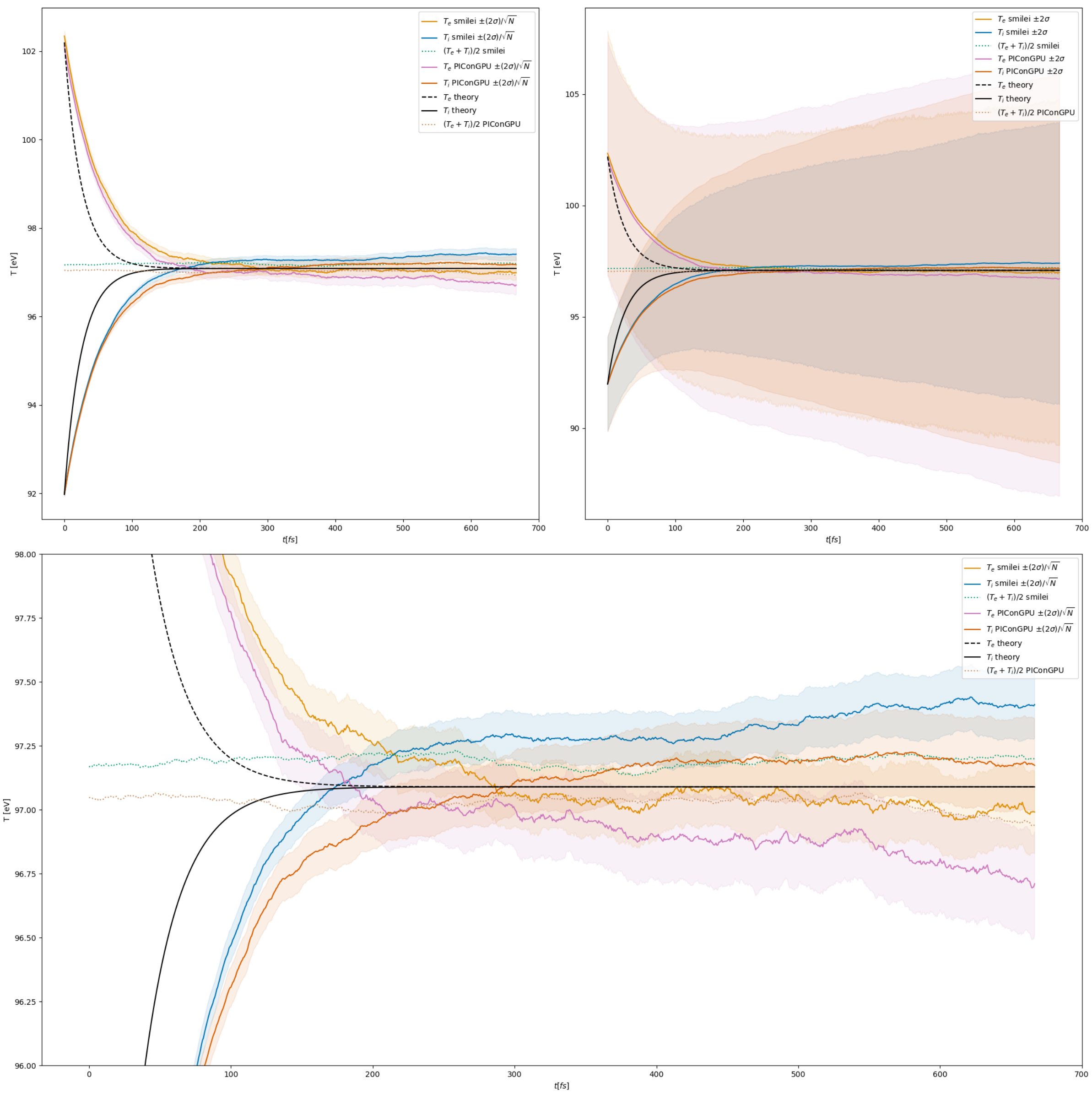


Fig. 25.8: Electron (blue) and ion (red) temperature over time in the thermalization test with a dynamic $\ln\Lambda$ for the electron-ion collisions The electron to ion weight ratio in the simulation is 1:5. Black lines are the the theoretical curves.

Last configured cmakeFlags setup was the setup No.: 6

```
-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center5000ppc;-
DPARAM_START_ELECTRONS=Center5000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-
DPARAM_DYNAMIC_LOG_INTRA=0'
```

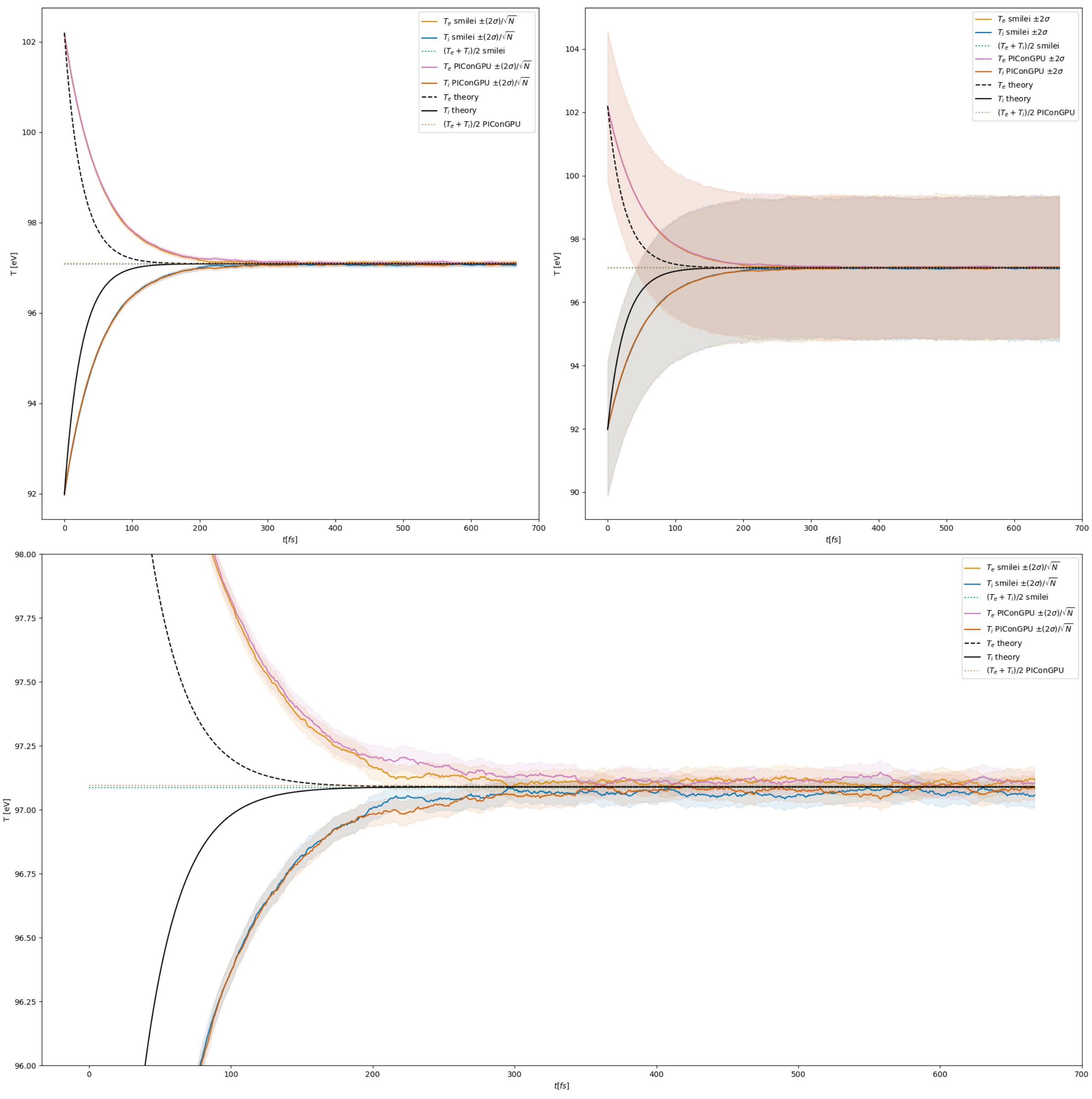
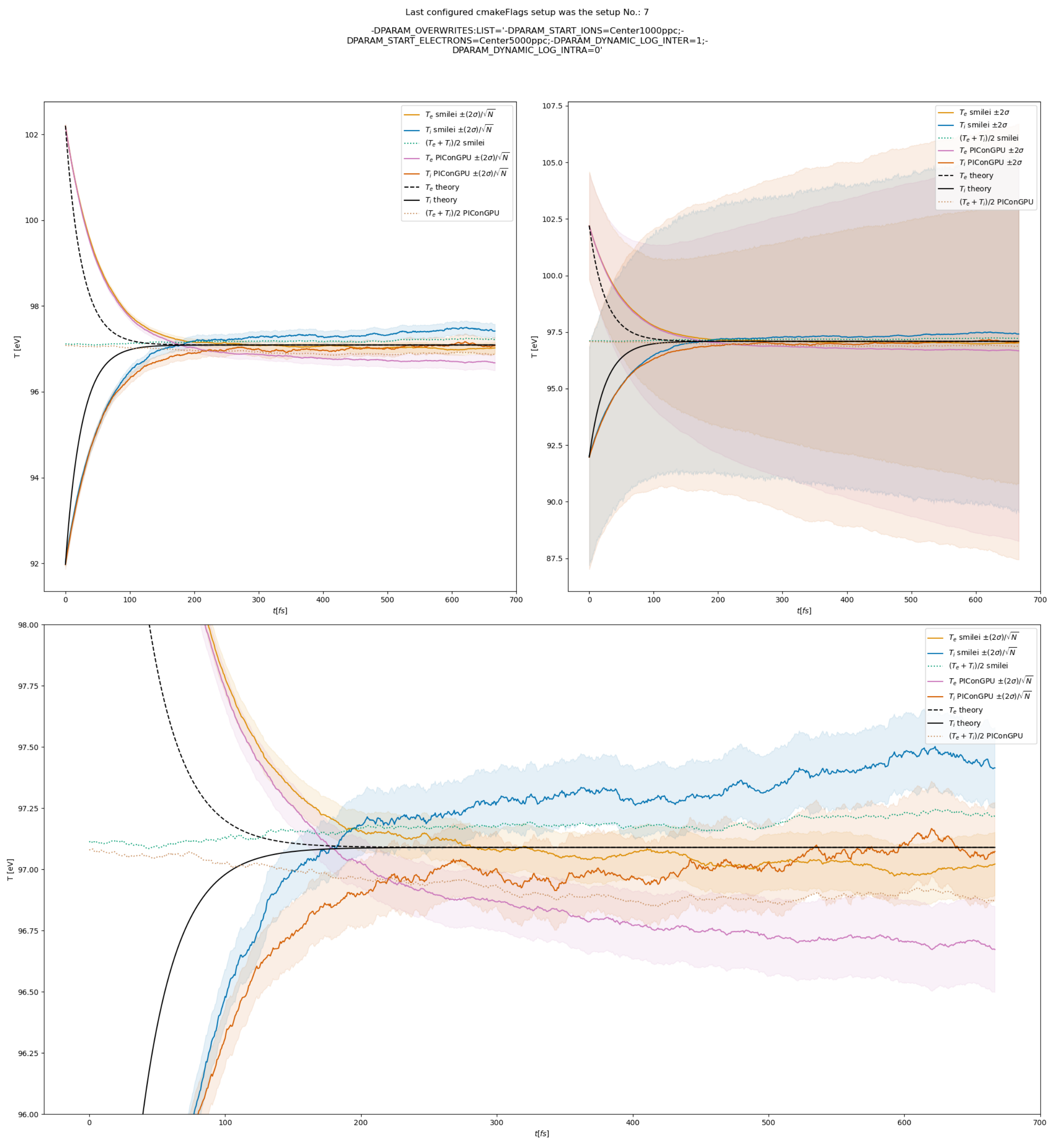
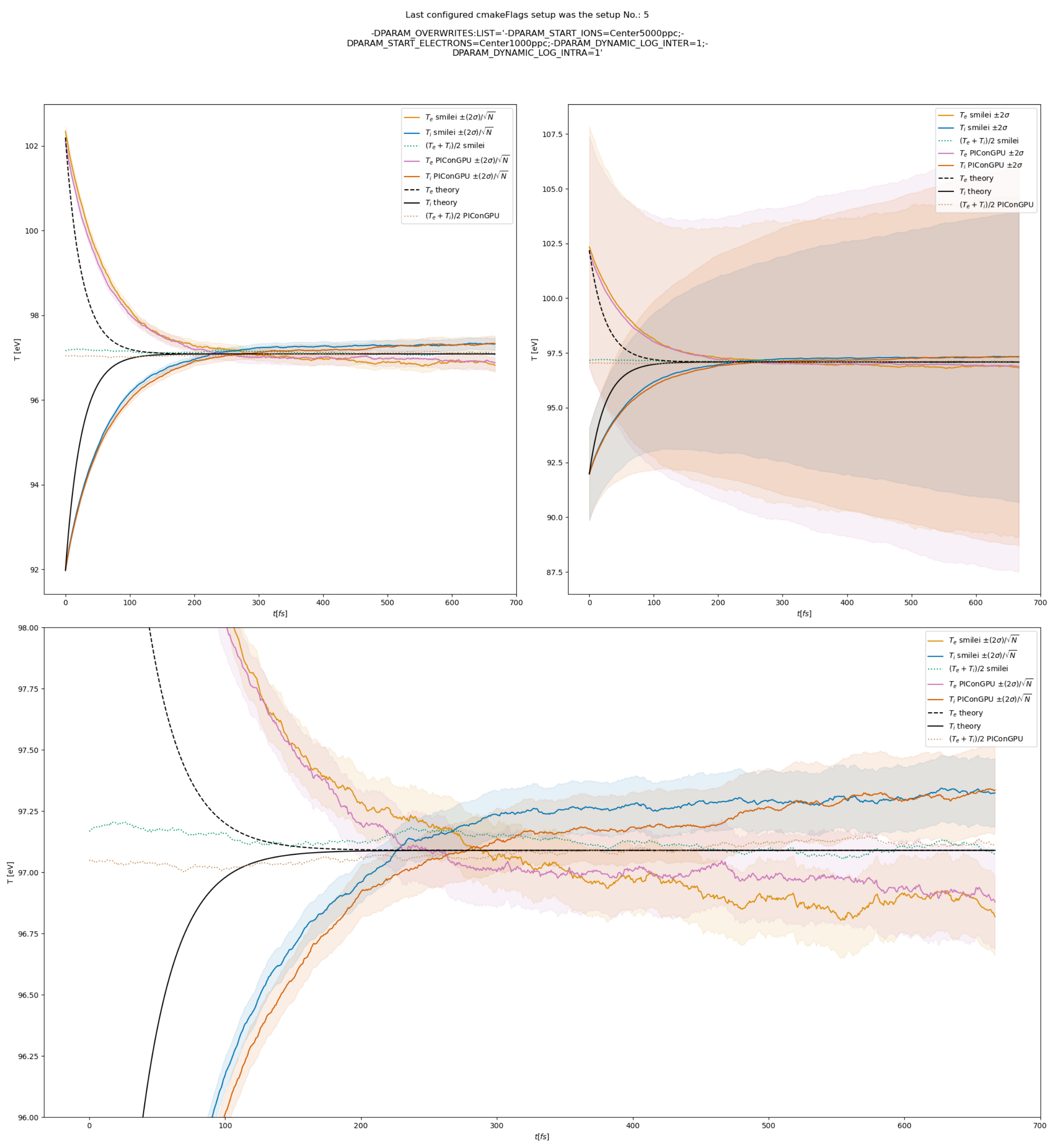


Fig. 25.9: Electron (blue) and ion (red) temperature over time in the thermalization test with a dynamic $\ln \Lambda$ for the electron-ion collisions. The electron to ion weight ratio in the simulation is 1:1. Black lines are the the theoretical curves.


Last configured cmakeFlags setup was the setup No.: 7

-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center1000ppc;-DPARAM_START_ELECTRONS=Center5000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-DPARAM_DYNAMIC_LOG_INTRA=0'

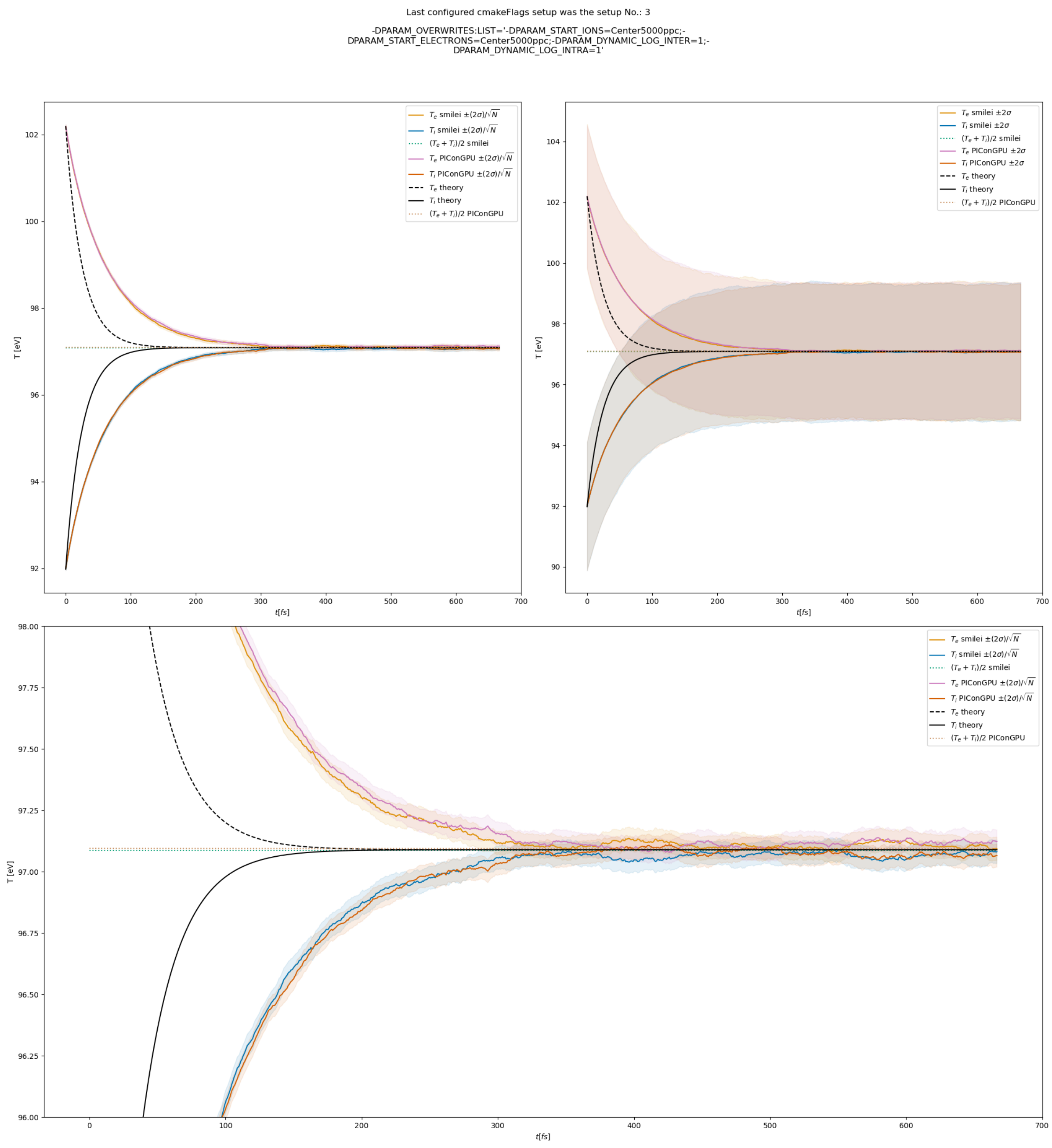
Fig. 25.10: Electron (blue) and ion (red) temperature over time in the thermalization test with a dynamic $\ln\Lambda$ for the electron-ion collisions. The electron to ion weight ratio in the simulation is 5:1. Black lines are the the theoretical curves.



Last configured cmakeFlags setup was the setup No.: 5

```
-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center5000ppc;-
DPARAM_START_ELECTRONS=Center1000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-
DPARAM_DYNAMIC_LOG_INTRA=1'
```

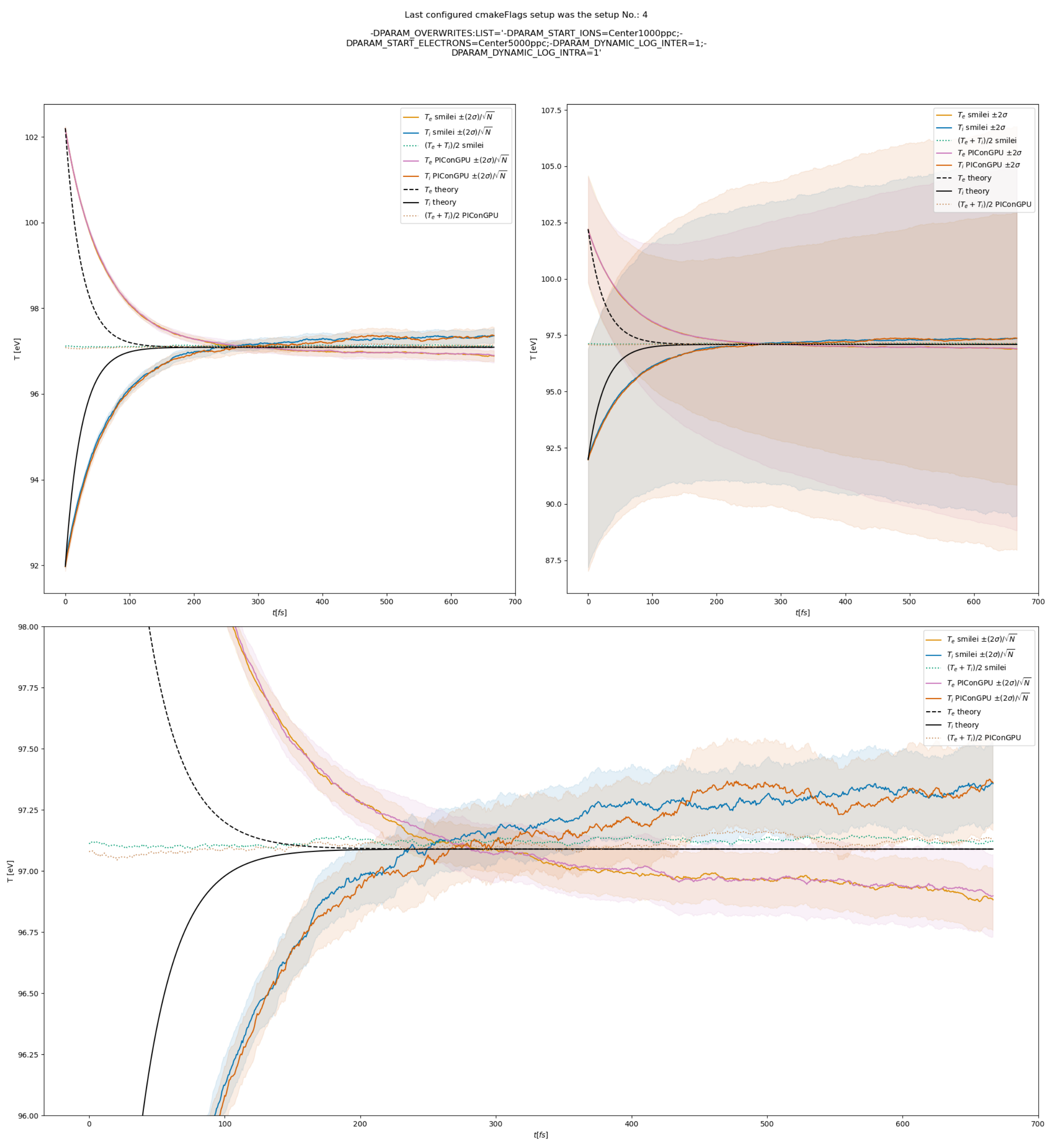
Fig. 25.11: Electron (blue) and ion (red) temperature over time in the thermalization test with a dynamic $\ln \Lambda$ for all collisions. The electron to ion weight ratio in the simulation is 1:5. Black lines are the the theoretical curves.



Last configured cmakeFlags setup was the setup No.: 3

```
-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center5000ppc;-
DPARAM_START_ELECTRONS=Center5000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-
DPARAM_DYNAMIC_LOG_INTRA=1'
```

Fig. 25.12: Electron (blue) and ion (red) temperature over time in the thermalization test with a dynamic $\ln \Lambda$ for all collisions. The electron to ion weight ratio in the simulation is 1:1. Black lines are the the theoretical curves.



Last configured cmakeFlags setup was the setup No.: 4

-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center1000ppc;-DPARAM_START_ELECTRONS=Center5000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-DPARAM_DYNAMIC_LOG_INTRA=1'

Fig. 25.13: Electron (blue) and ion (red) temperature over time in the thermalization test with a dynamic $\ln \Lambda$ for all collisions. The electron to ion weight ratio in the simulation is 5:1. Black lines are the the theoretical curves.

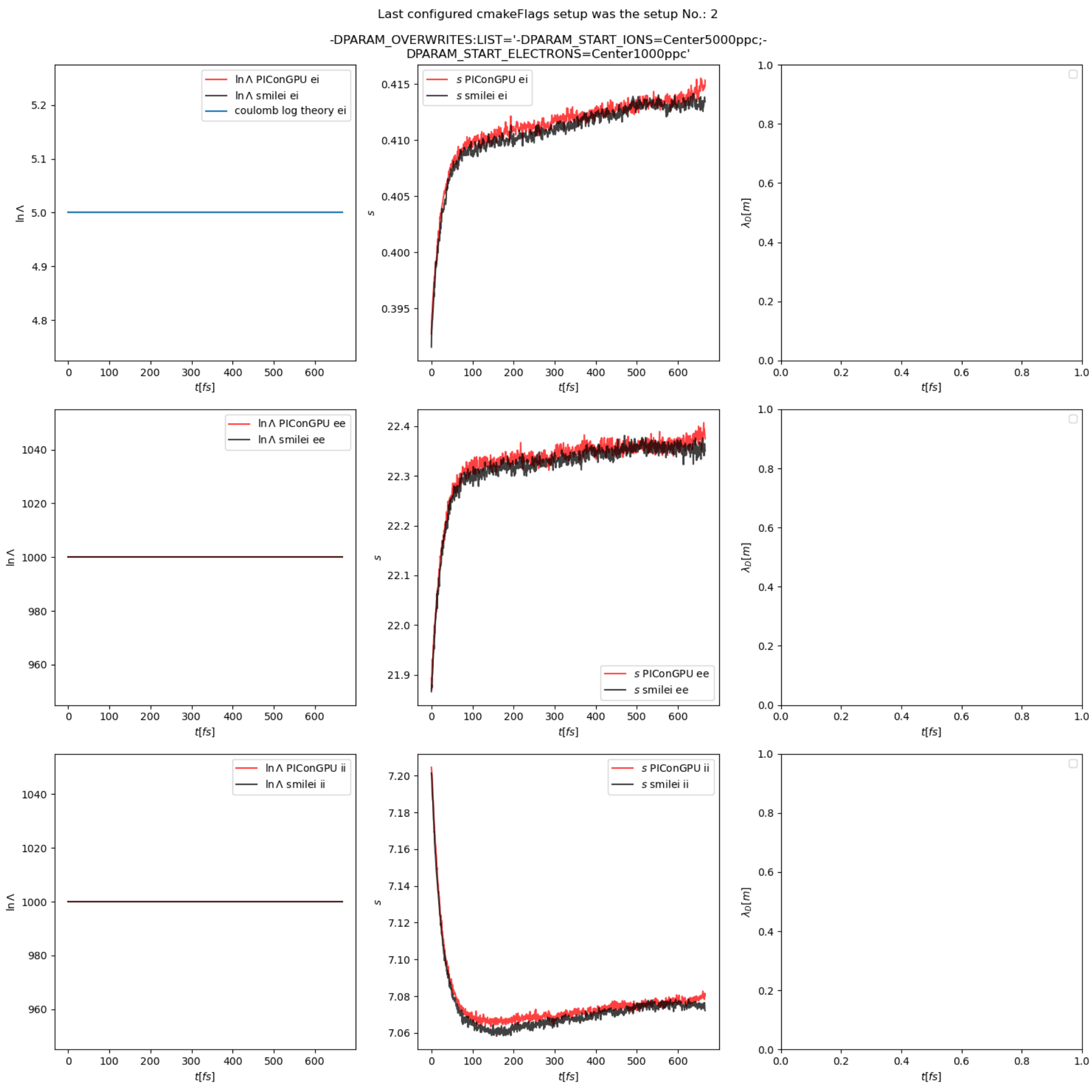


Fig. 25.14: Collision debug values from the thermalization test with both $\ln \Lambda$ constant. The electron to ion weight ratio in the simulation is 1:5.

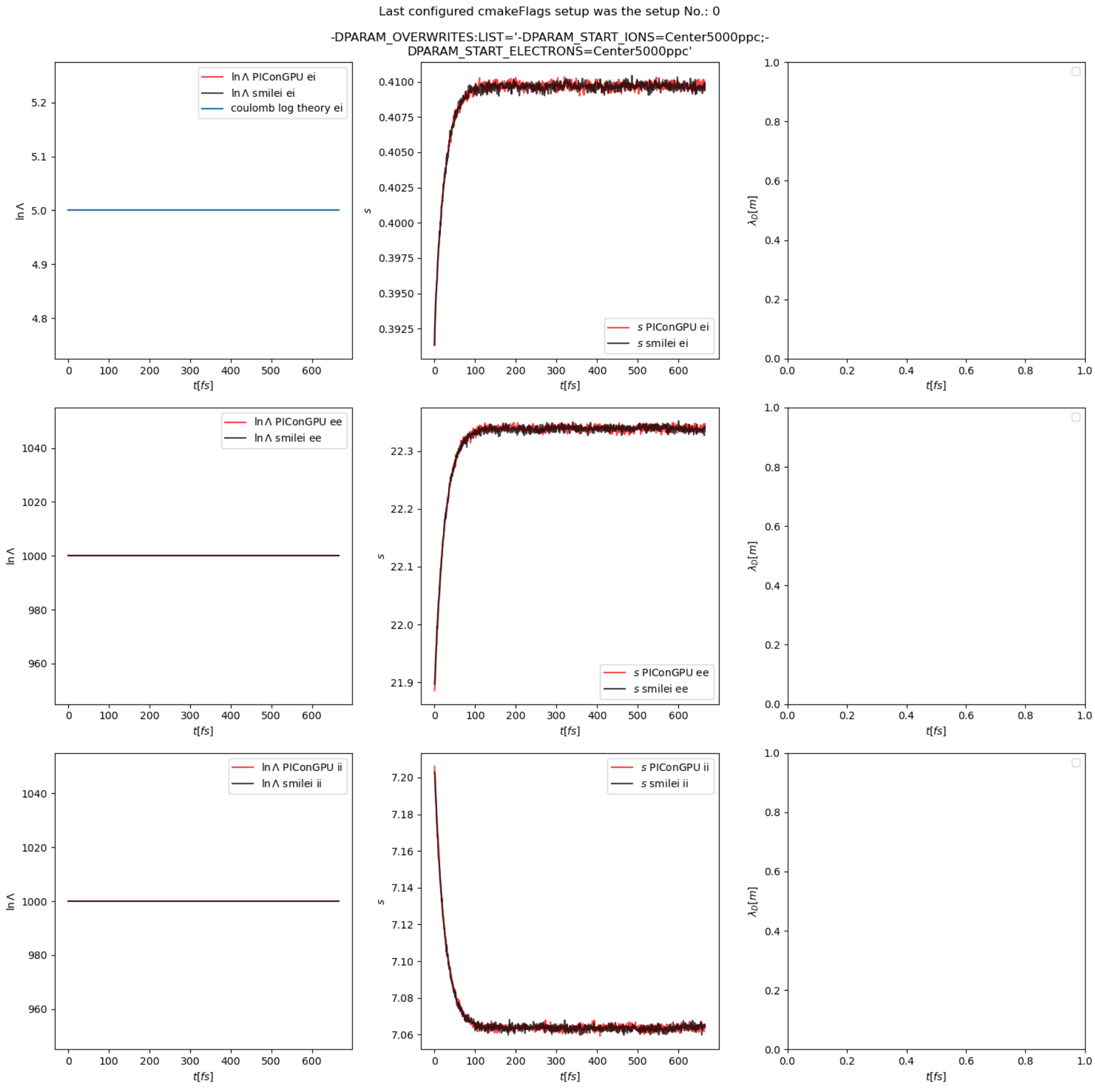


Fig. 25.15: Collision debug values from the thermalization test with both $\ln \Lambda$ constant. The electron to ion weight ratio in the simulation is 1:1.

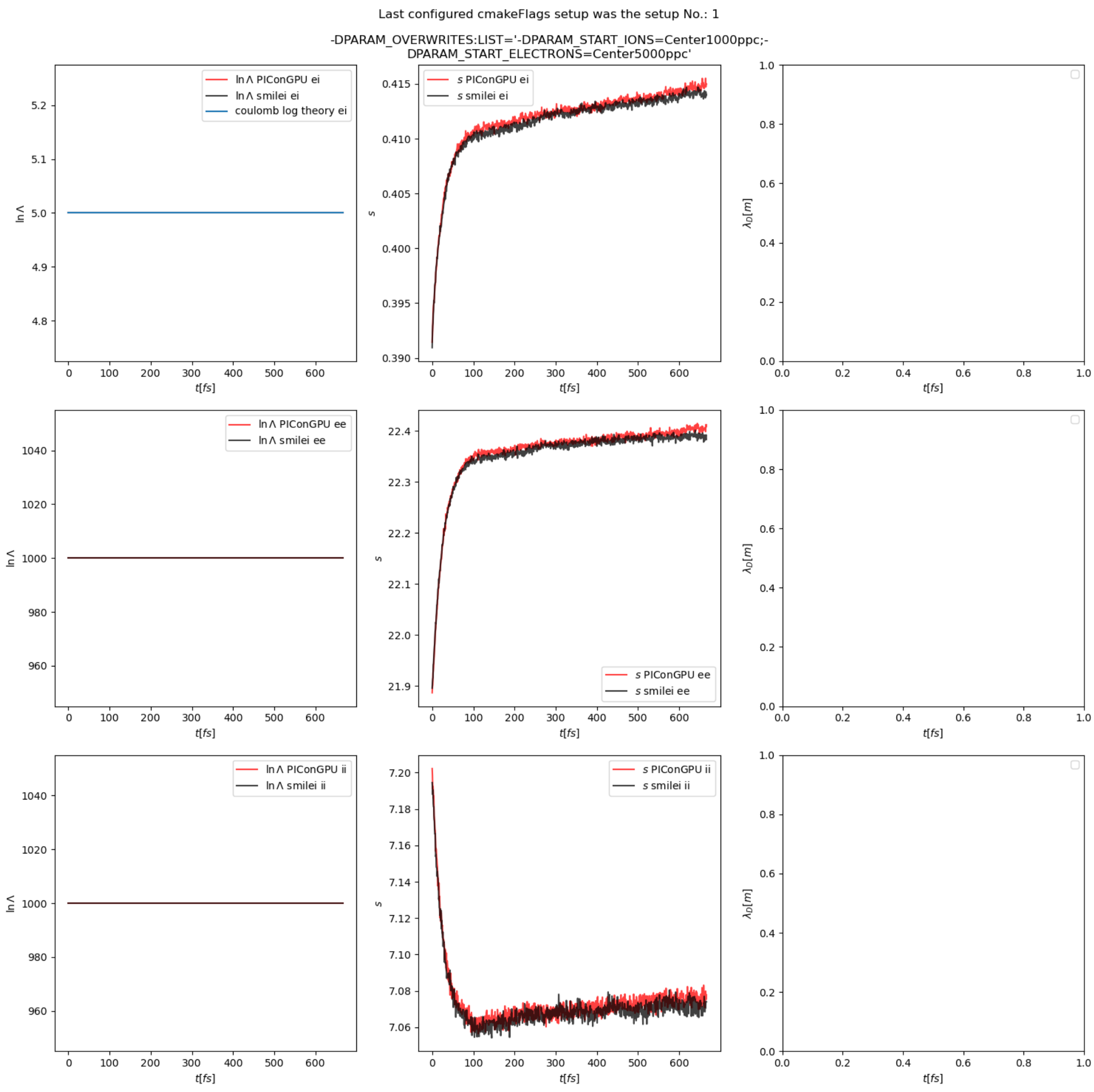
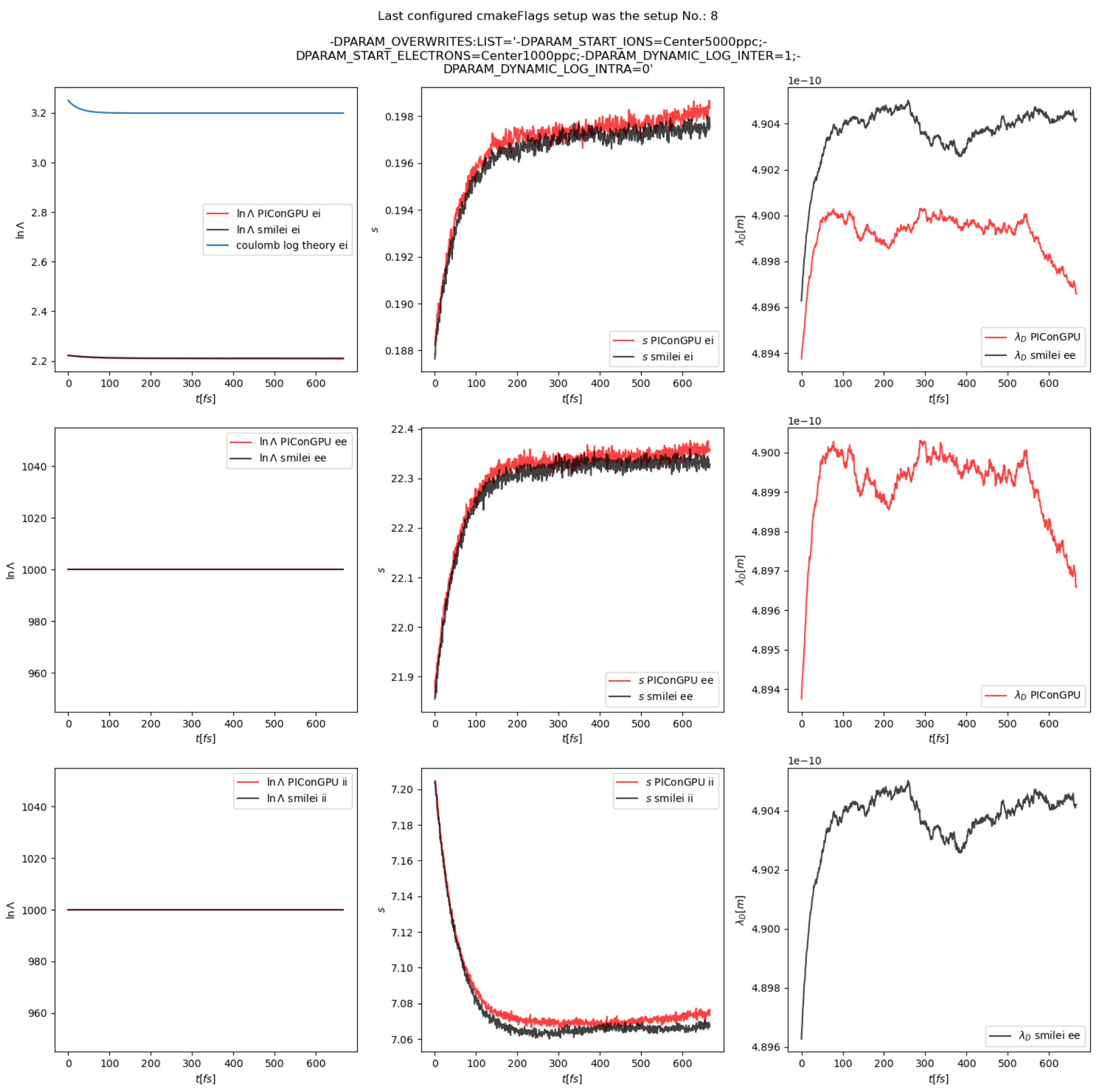


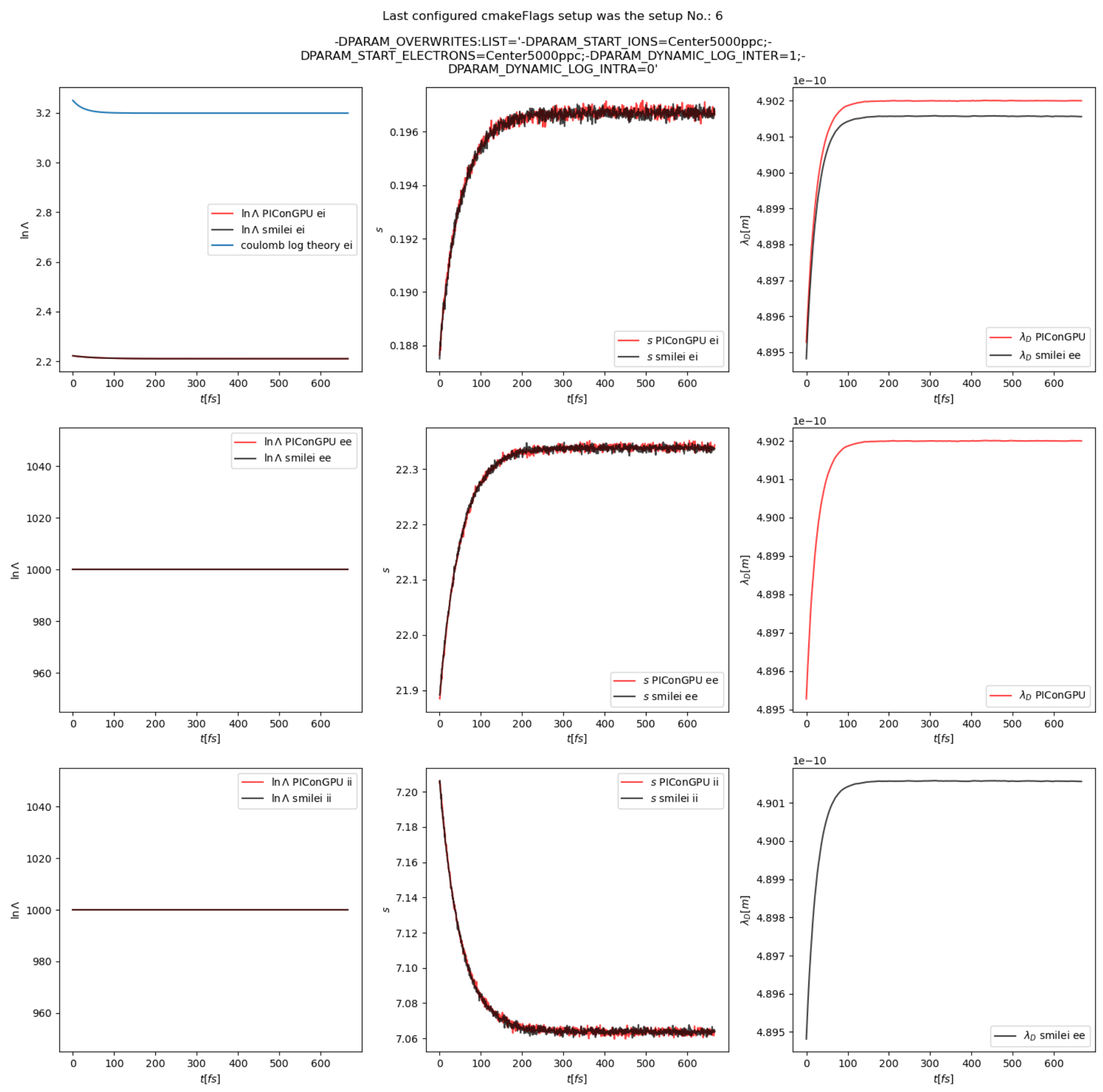
Fig. 25.16: Collision debug values from the thermalization test with both $\ln\Lambda$ constant. The electron to ion weight ratio in the simulation is 5:1.



Last configured cmakeFlags setup was the setup No.: 8

-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center5000ppc;-DPARAM_START_ELECTRONS=Center1000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-DPARAM_DYNAMIC_LOG_INTRA=0'

Fig. 25.17: Collision debug values from the thermalization test with a dynamic $\ln \Lambda$ for the electron-ion collisions The electron to ion weight ratio in the simulation is 1:5.


Last configured cmakeFlags setup was the setup No.: 6

-DPARAM_OVERWRITES:LIST='-DPARAM_START_IONS=Center5000ppc;-DPARAM_START_ELECTRONS=Center5000ppc;-DPARAM_DYNAMIC_LOG_INTER=1;-DPARAM_DYNAMIC_LOG_INTRA=0'

Fig. 25.18: Collision debug values from the thermalization test with a dynamic $\ln\Lambda$ for the electron-ion collisions. The electron to ion weight ratio in the simulation is 1:1.

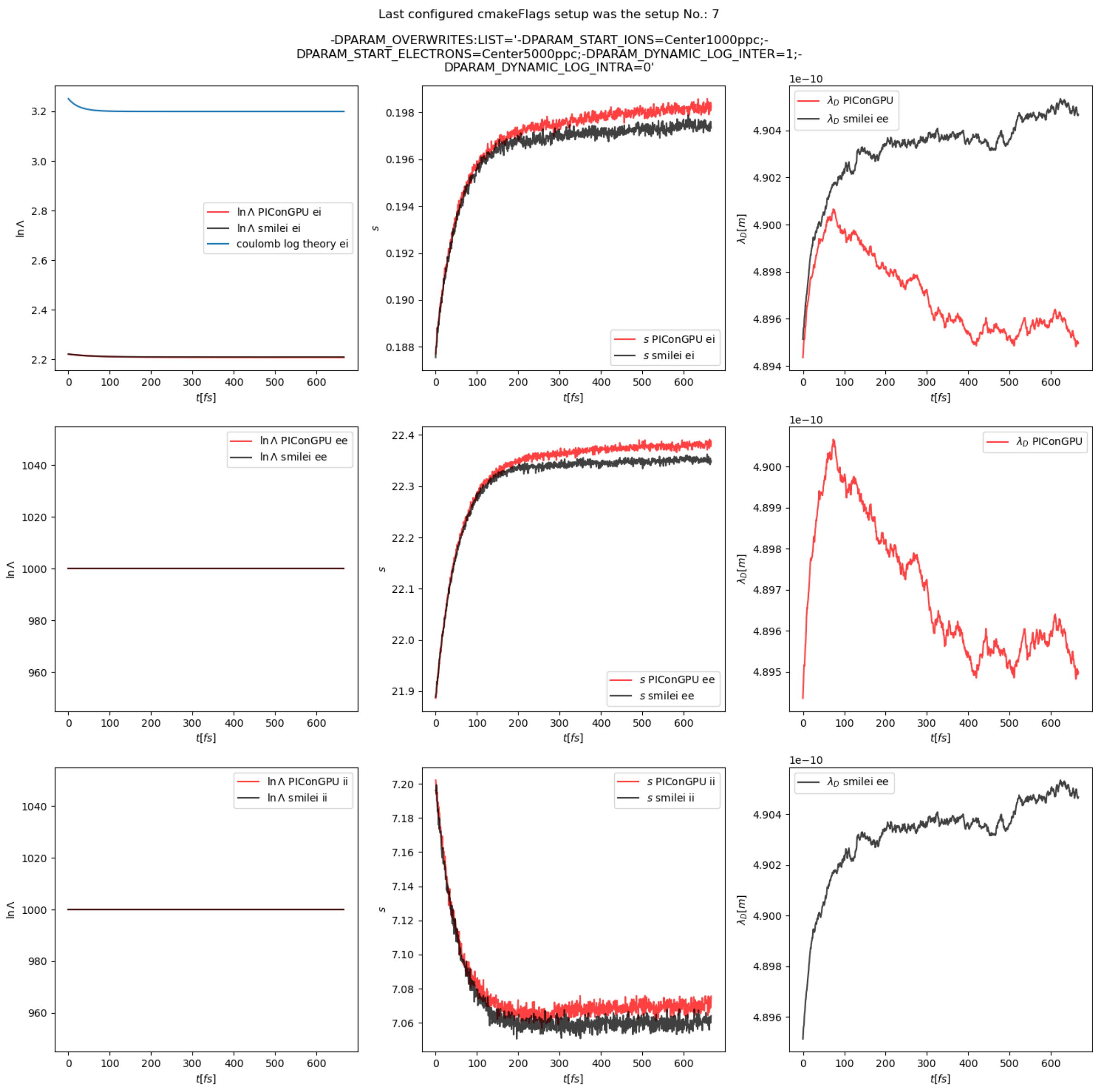


Fig. 25.19: Collision debug values from the thermalization test with a dynamic $\ln\Lambda$ for the electron-ion collisions. The electron to ion weight ratio in the simulation is 5:1.

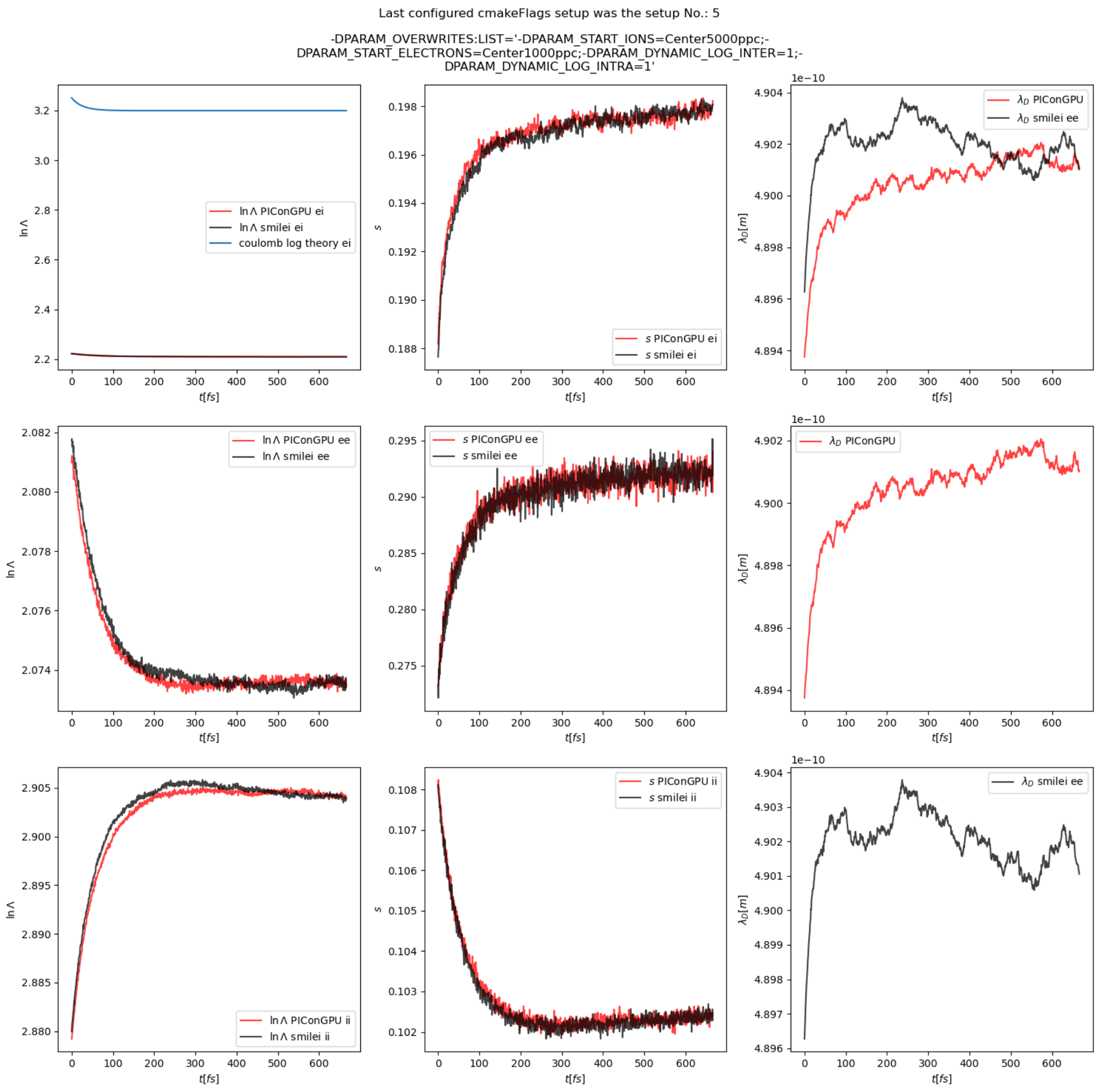


Fig. 25.20: Collision debug values from the thermalization test with a dynamic $\ln \Lambda$ for all collisions. The electron to ion weight ratio in the simulation is 1:5.

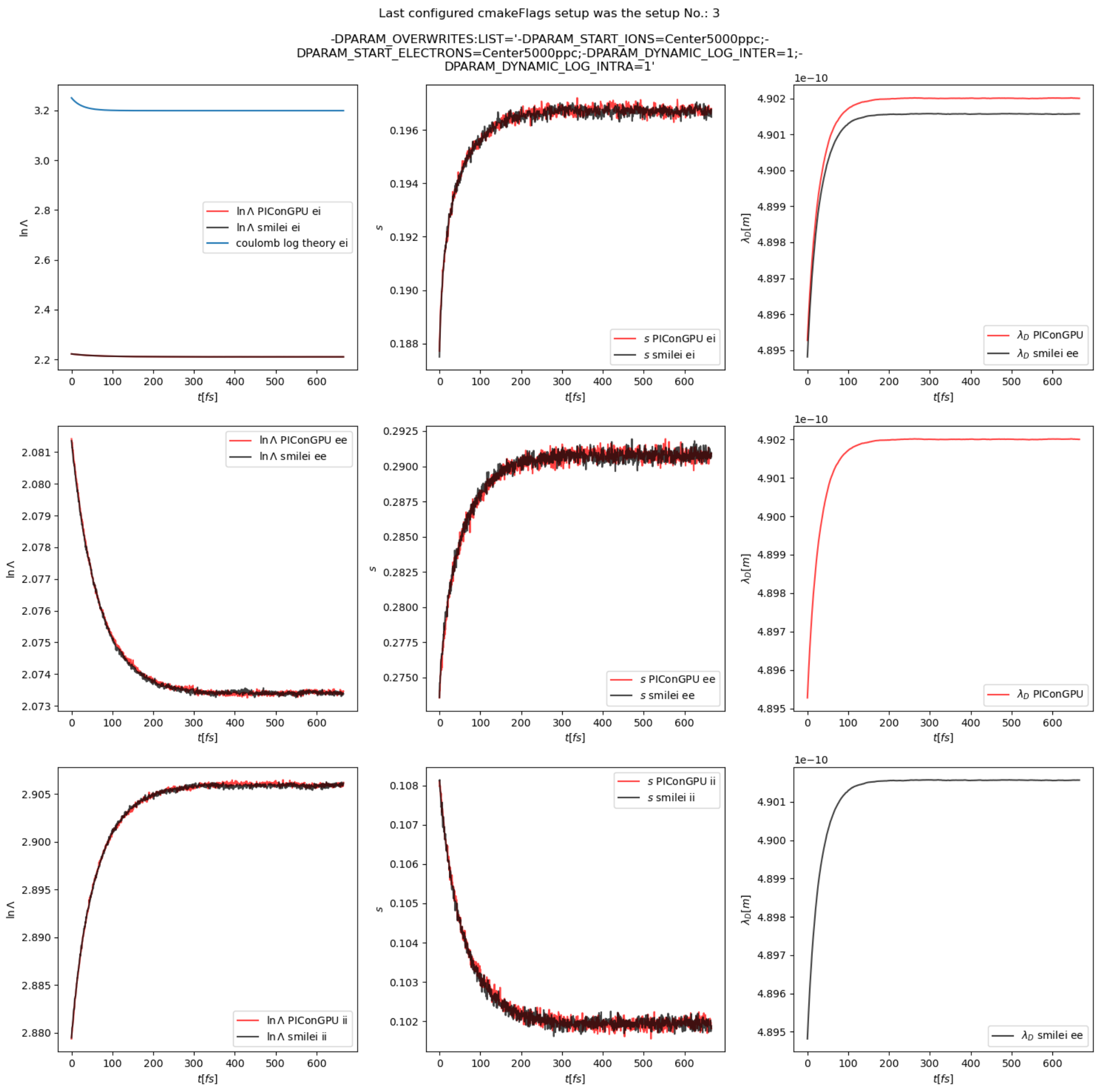


Fig. 25.21: Collision debug values from the thermalization test with a dynamic $\ln \Lambda$ for all collisions. The electron to ion weight ratio in the simulation is 1:1.

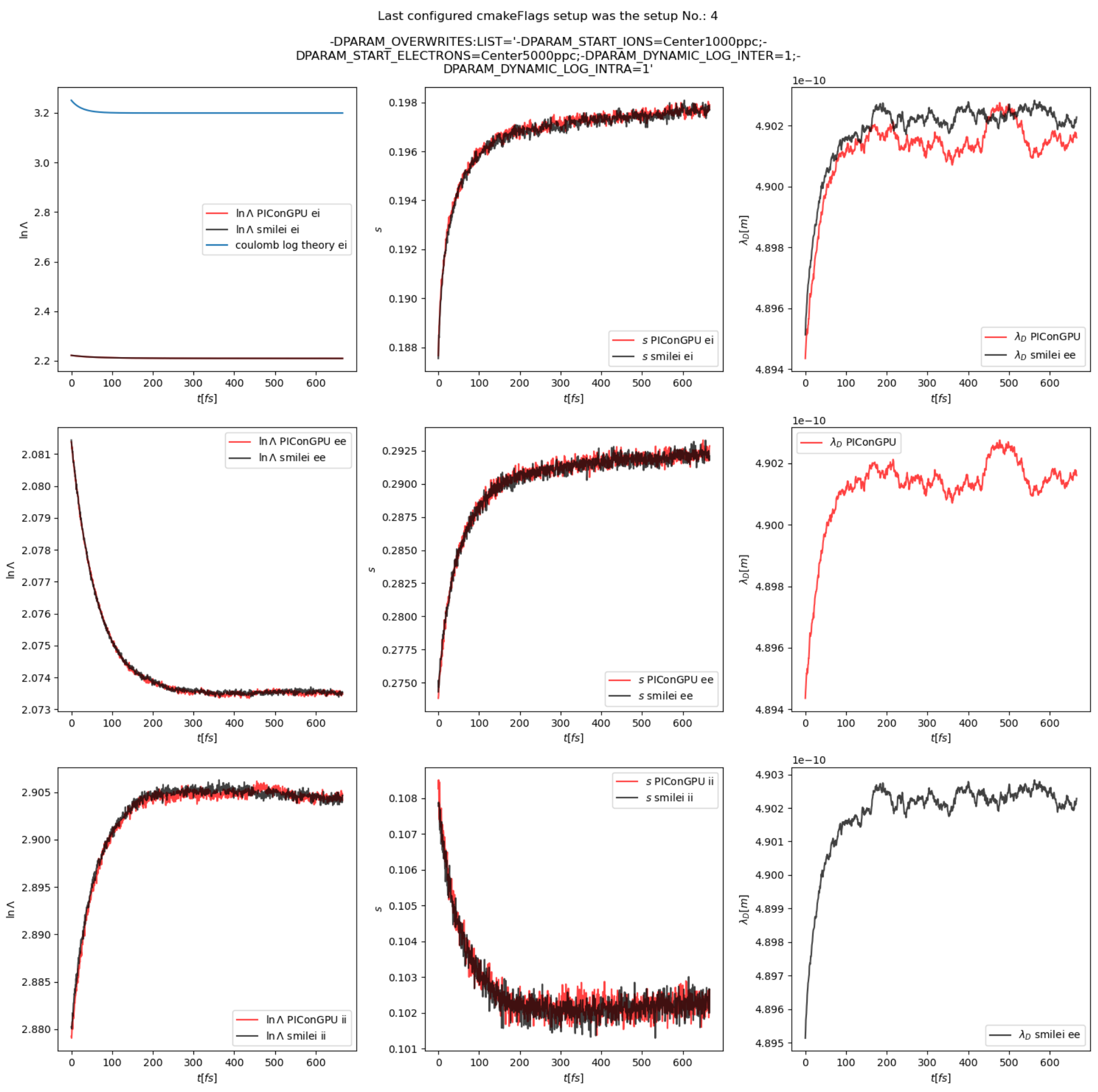


Fig. 25.22: Collision debug values from the thermalization test with a dynamic $\ln \Lambda$ for all collisions. The electron to ion weight ratio in the simulation is 5:1.

Table 25.1: Simulation parameters in the thermalization test

parameter or setting	value
time step duration	2/3 fs
time steps in the simulation	100
density profile	homogeneous
density	1.1 × 10^{28} m^{-3}
cell side length	$\frac{1}{3}c \cdot 10^{-13} \approx 10\mu m$
ion mass	10 m_e
ion charge	+1
initial ion temperature	1.8 × 10^4 $m_e c^2$
initial electron temperature	2.0 × 10^4 $m_e c^2$
Coulomb logarithm (inter–collisions)	5 (if not dynamic)
Coulomb logarithm (intra–collisions)	1000 (if not dynamic)
geometry	2D
grid	12x12
super cell size	4x4
macro particles per cell (ions) setups 1, 2, 3	5000, 1000, 5000
macro pearticles per cell (electrons) setups 1, 2, 3	5000, 5000, 1000

25.4.2 4.2 Beam relaxation

A population of electrons with a very small temperature and a drift velocity (the beam) is colliding with ions. Due to the collisions the velocity distribution of electrons is changing and the drift momentum is transferred into the electron transversal momentum and partially into ion momenta. In this test only the inter-collisions (between ions and electrons) are enabled.

There are three slightly different setups with varying electron drift velocity, ion charge and time step duration. Additionally each setup performs the collisions with three different electron to ion weight ratios: 1:1, 5:1, 1:5. This is achieved by dividing the simulation box into three parts and enabling collisions only for one ratio in each part. All important simulation parameters can be found in tables Table 25.2 and Table 25.3. This test was also extended with runs utilizing the automatic Coulomb logarithm calculation.

The following figures show the electron and ion drift velocities $\langle v_x \rangle$, electron transversal velocity $\sqrt{\langle v_\perp^2 \rangle}$, as well as the ion drift velocity, developing over time. All figures include a comparison values from `smilei` simulations, the `smilei` inputs can be found in `smilei_beam_relaxation_inputs.zip`.

The debug average quantities for $\ln \Lambda, s, \lambda_D$ are shown in figures … .

Table 25.2: Collisions in the 3 parts of the simulation box in the beam relaxation example

parameter	upper part	middle part	lower part
macro particles per cell (ions)	1000	1000	100
macro particles per cell (electrons)	1000	100	1000

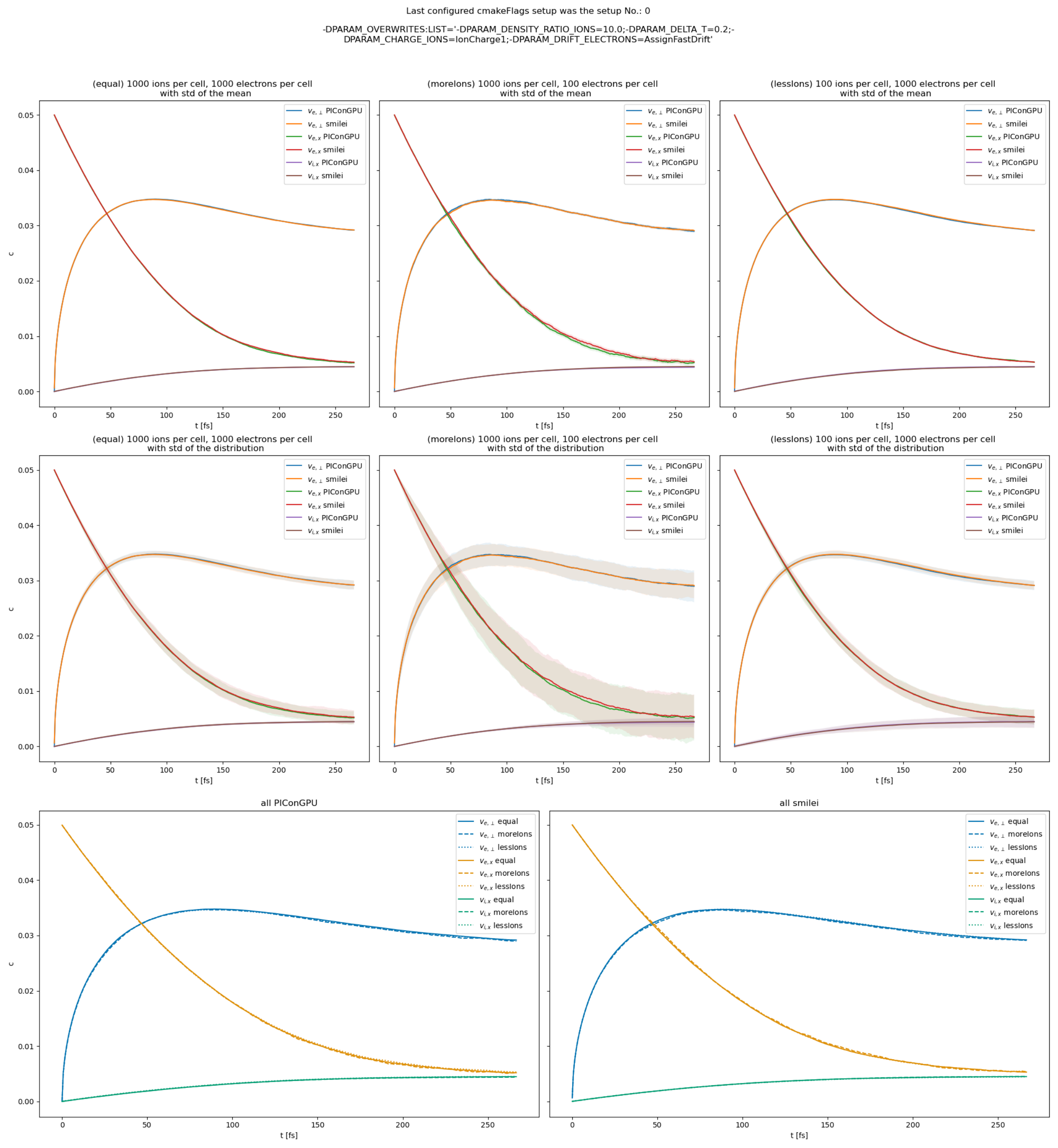
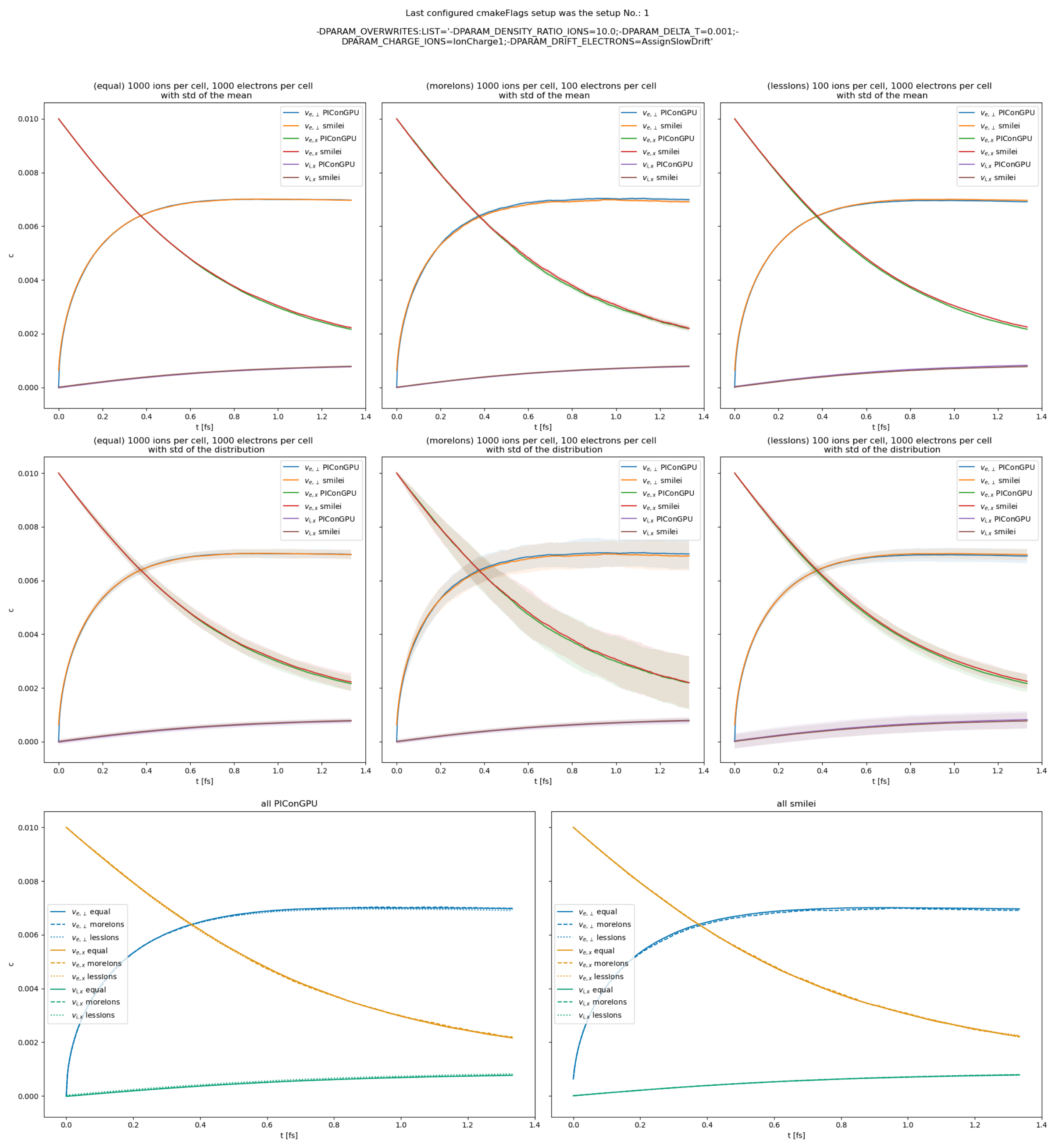


Fig. 25.23: Electron drift velocity $\langle v_x \rangle$, electron transversal velocity $\sqrt{\langle v_\perp^2 \rangle}$, and ion drift velocities from the beam equilibration example setup 1 (with a constant Coulomb logarithm).



Last configured cmakeFlags setup was the setup No.: 1

-DPARAM_OVERWRITES:LIST='-DPARAM_DENSITY_RATIO_IONS=10.0;-DPARAM_DELTA_T=0.001;-DPARAM_CHARGE_IONS=IonCharge1;-DPARAM_DRIFT_ELECTRONS=AssignSlowDrift'

Fig. 25.24: Electron drift velocity $\langle v_x \rangle$, electron transversal velocity $\sqrt{\langle v_\perp^2 \rangle}$, and ion drift velocities from the beam equilibration example setup 2 (with a constant Coulomb logarithm).

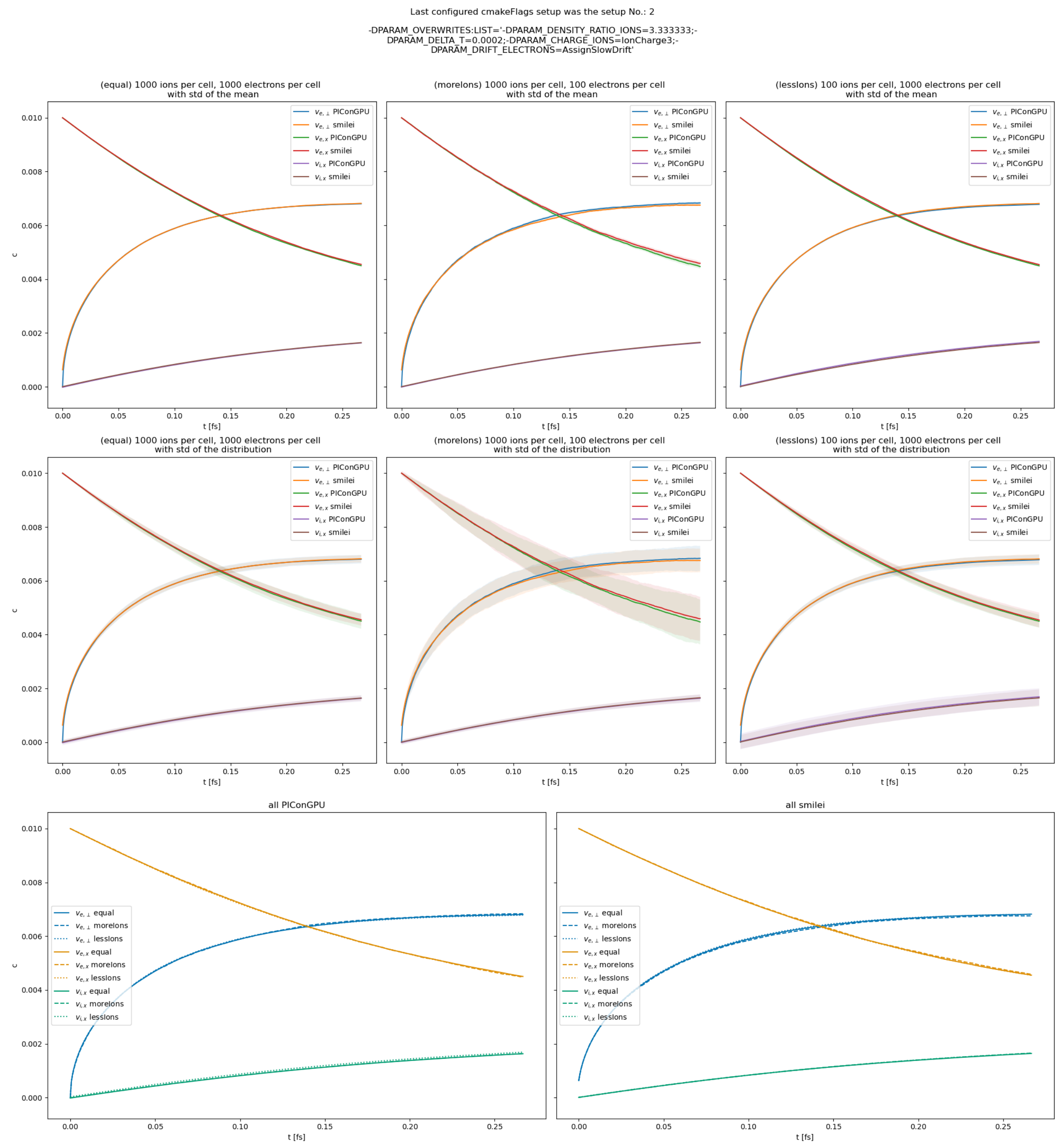


Fig. 25.25: Electron drift velocity $\langle v_x \rangle$, electron transversal velocity $\sqrt{\langle v_\perp^2 \rangle}$, and ion drift velocities from the beam equilibration example setup 3 (with a constant Coulomb logarithm).

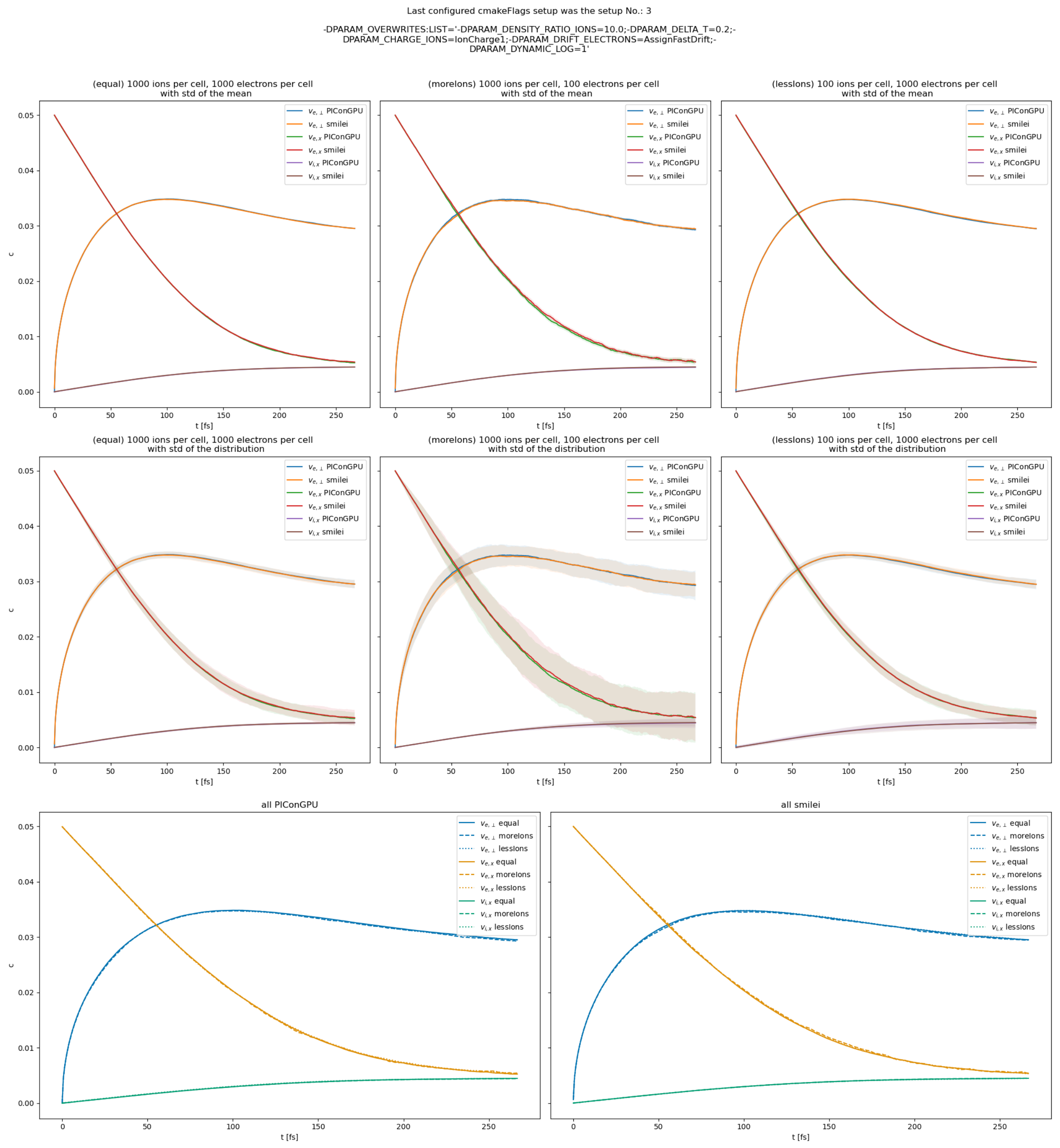
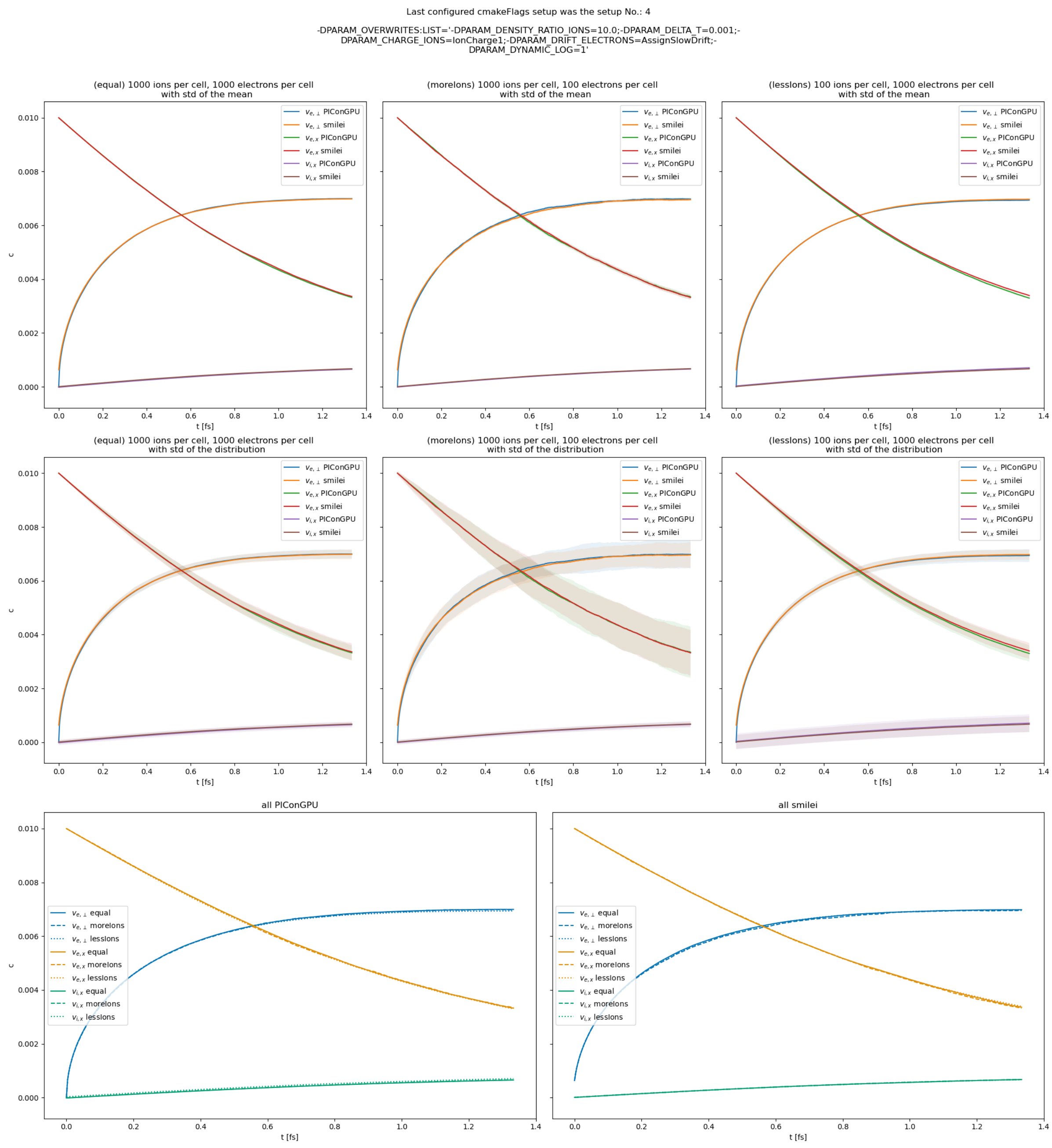


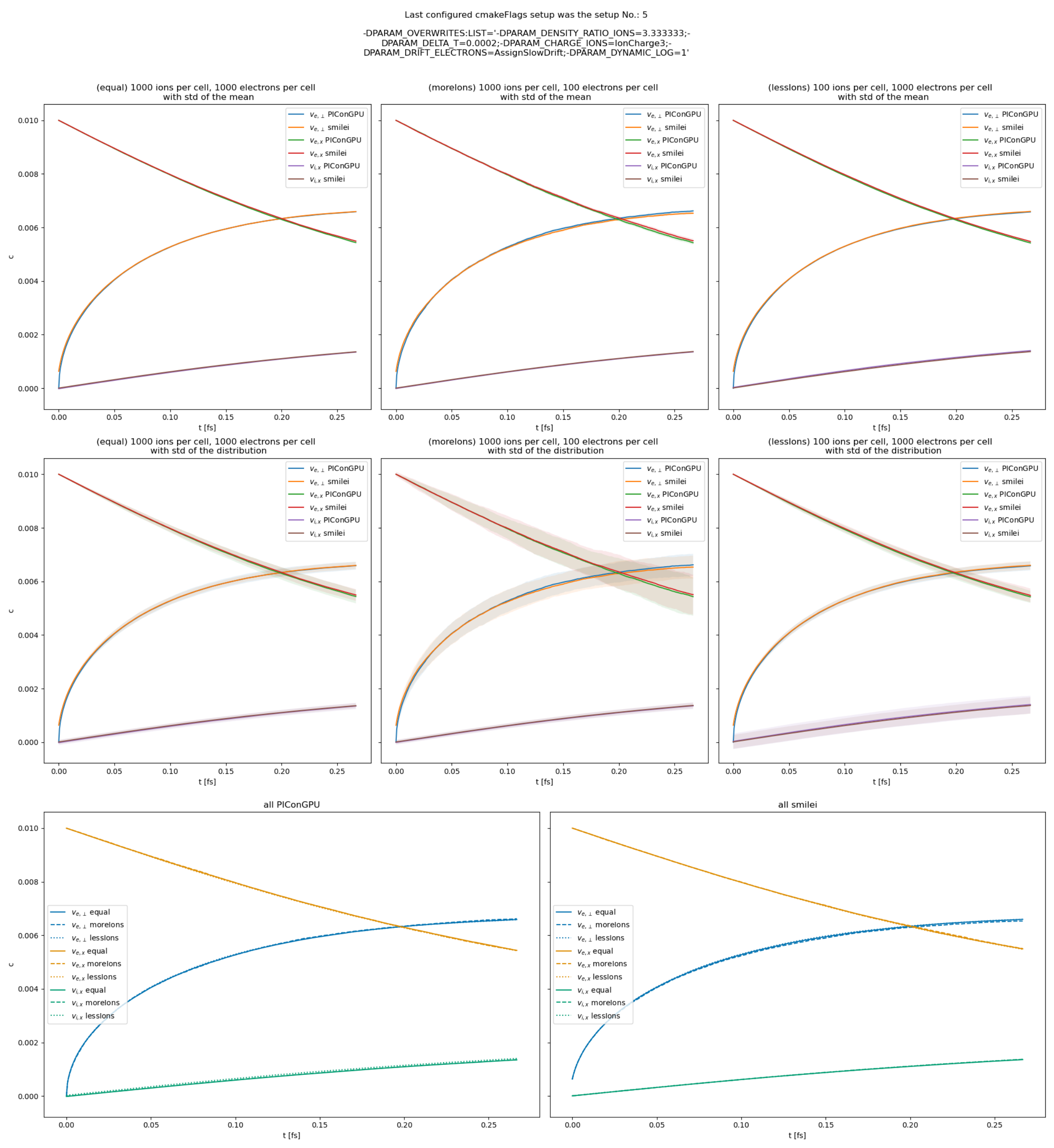
Fig. 25.26: Electron drift velocity $\langle v_x \rangle$, electron transversal velocity $\sqrt{\langle v_\perp^2 \rangle}$, and ion drift velocities from the beam equilibration example setup 1 (with dynamic Coulomb logarithm).



Last configured cmakeFlags setup was the setup No.: 4

-DPARAM_OVERWRITES:LIST='-DPARAM_DENSITY_RATIO_IONS=10.0;-DPARAM_DELTA_T=0.001;-DPARAM_CHARGE_IONS=IonCharge1;-DPARAM_DRIFT_ELECTRONS=AssignSlowDrift;-DPARAM_DYNAMIC_LOG=1'

Fig. 25.27: Electron drift velocity $\langle v_x \rangle$, electron transversal velocity $\sqrt{\langle v_\perp^2 \rangle}$, and ion drift velocities from the beam equilibration example setup 2 (with dynamic Coulomb logarithm).



Last configured cmakeFlags setup was the setup No.: 5

```
-DPARAM_OVERWRITES:LIST='-DPARAM_DENSITY_RATIO_IONS=3.333333;-
DPARAM_DELTA_T=0.0002;-DPARAM_CHARGE_IONS=IonCharge3;-
DPARAM_DRIFT_ELECTRONS=AssignSlowDrift;-DPARAM_DYNAMIC_LOG=1'
```

Fig. 25.28: Electron drift velocity $\langle v_x \rangle$, electron transversal velocity $\sqrt{\langle v_\perp^2 \rangle}$, and ion drift velocities from the beam equilibration example setup 3 (with dynamic Coulomb logarithm).

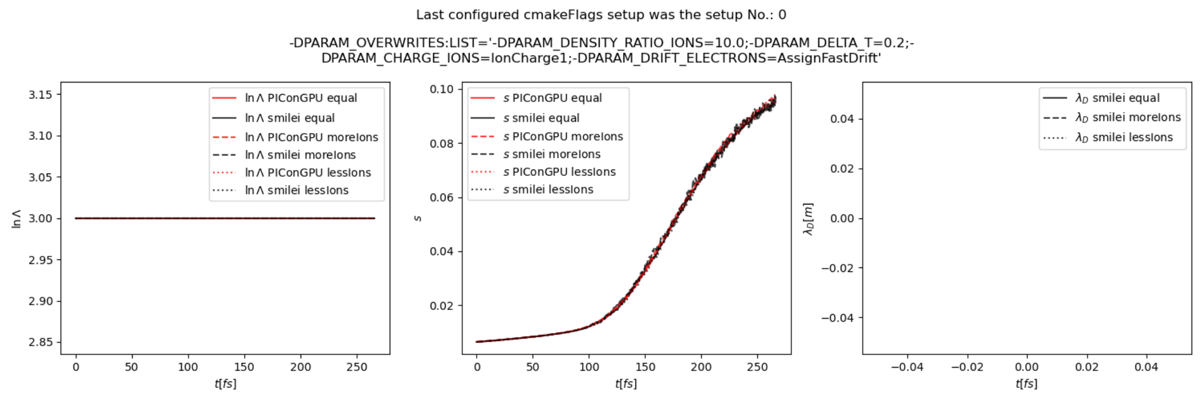


Fig. 25.29: Average Coulomb logarithm $\ln \Lambda$, s (proportional to collision frequency), Debye length λ_D from the beam equilibration example setup 1 (with a constant Coulomb logarithm).

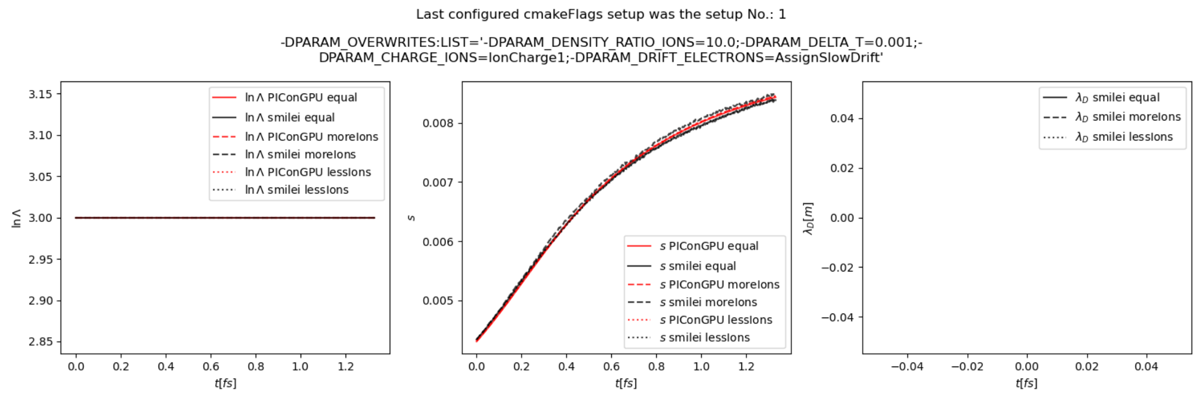


Fig. 25.30: Average Coulomb logarithm $\ln \Lambda$, s (proportional to collision frequency), Debye length λ_D from the beam equilibration example setup 2 (with a constant Coulomb logarithm).

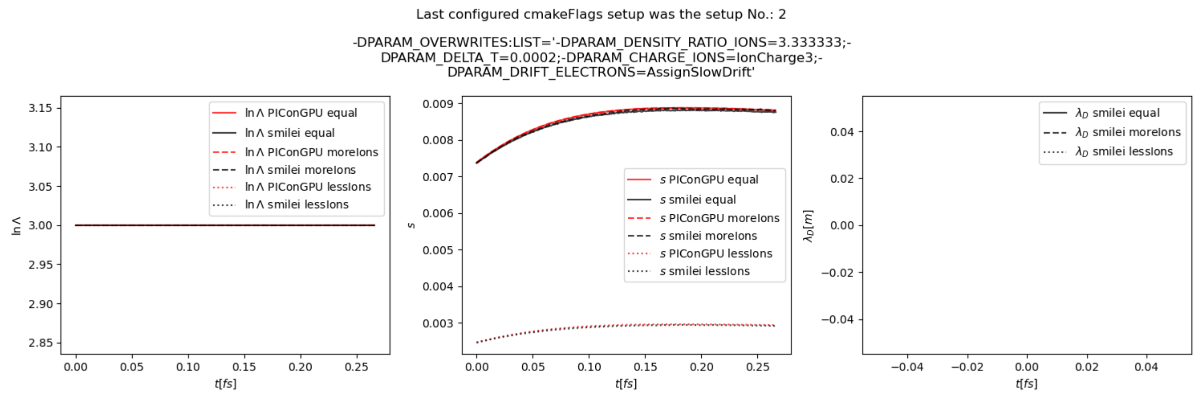
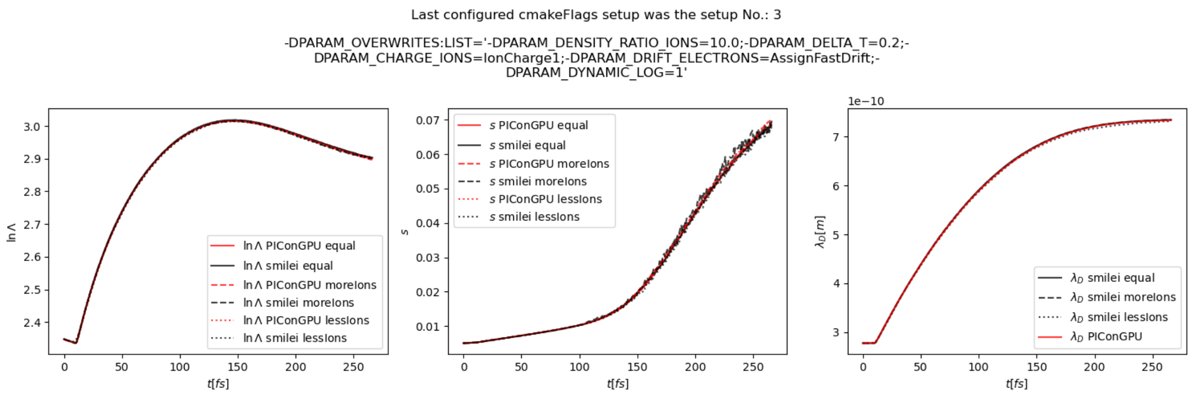


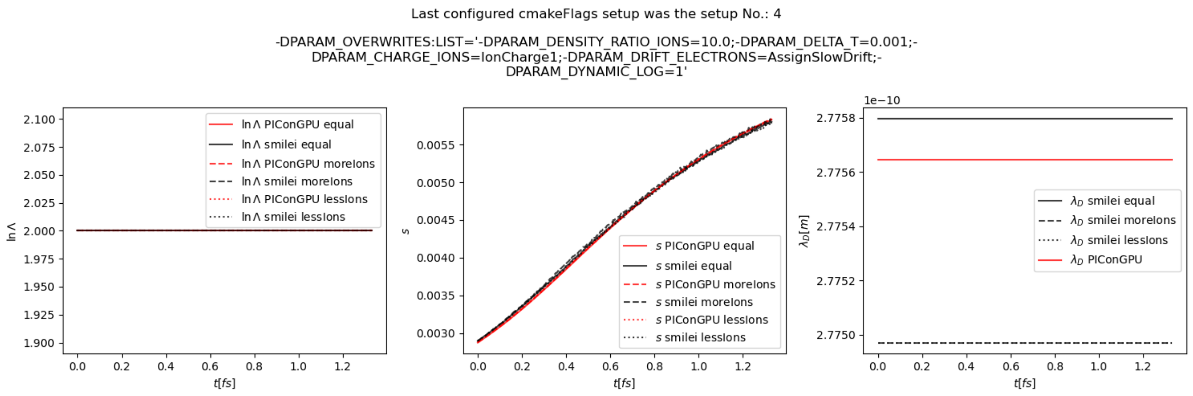
Fig. 25.31: Average Coulomb logarithm $\ln \Lambda$, s (proportional to collision frequency), Debye length λ_D from the beam equilibration example setup 3 (with a constant Coulomb logarithm).



Last configured cmakeFlags setup was the setup No.: 3

-DPARAM_OVERWRITES:LIST='-DPARAM_DENSITY_RATIO_IONS=10.0;-DPARAM_DELTA_T=0.2;-DPARAM_CHARGE_IONS=IonCharge1;-DPARAM_DRIFT_ELECTRONS=AssignFastDrift;-DPARAM_DYNAMIC_LOG=1'

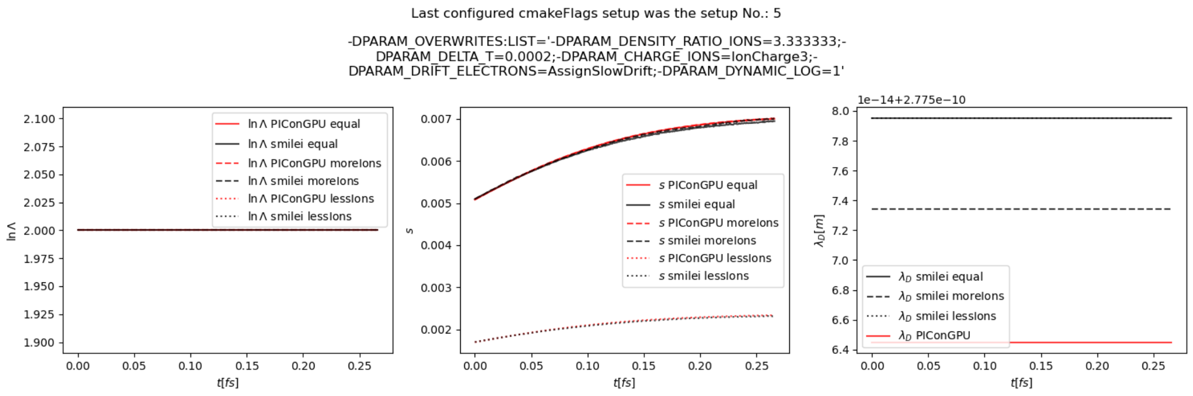
Fig. 25.32: Average Coulomb logarithm $\ln \Lambda$, s (proportional to collision frequency), Debye length λ_D from the beam equilibration example setup 1 (with dynamic Coulomb logarithm).



Last configured cmakeFlags setup was the setup No.: 4

-DPARAM_OVERWRITES:LIST='-DPARAM_DENSITY_RATIO_IONS=10.0;-DPARAM_DELTA_T=0.001;-DPARAM_CHARGE_IONS=IonCharge1;-DPARAM_DRIFT_ELECTRONS=AssignSlowDrift;-DPARAM_DYNAMIC_LOG=1'

Fig. 25.33: Average Coulomb logarithm $\ln \Lambda$, s (proportional to collision frequency), Debye length λ_D from the beam equilibration example setup 2 (with dynamic Coulomb logarithm).



Last configured cmakeFlags setup was the setup No.: 5

-DPARAM_OVERWRITES:LIST='-DPARAM_DENSITY_RATIO_IONS=3.333333;-DPARAM_DELTA_T=0.0002;-DPARAM_CHARGE_IONS=IonCharge3;-DPARAM_DRIFT_ELECTRONS=AssignSlowDrift;-DPARAM_DYNAMIC_LOG=1'

Fig. 25.34: Average Coulomb logarithm $\ln \Lambda$, s (proportional to collision frequency), Debye length λ_D from the beam equilibration example setup 3 (with dynamic Coulomb logarithm).

Table 25.3: Simulation parameters in beam the relaxation test

parameter or setting	value setup 1	setup 2	setup 3
time step duration	$\frac{2}{3}$ fs	$\frac{0.01}{3}$ fs	$\frac{0.002}{3}$ fs
time steps in the simulation	200		
density profile	homogeneous		
density electrons	1.1 × 10^{28} m^{-3}		
density ions	1.1 × 10^{28} m^{-3}	1.1 × 10^{28} m^{-3}	3.7 × 10^{27} m^{-3}
cell side length	$\frac{1}{15}c \cdot 10^{-13} \approx 2\mu m$		
ion mass	10 m_e		
ion charge	+1	+1	+3
initial electron drift	0.05c	0.01c	0.01c
initial ion temperature	0.00002 $m_e c^2$		
initial electron temperature	0.0000002 $m_e c^2$		
Coulomb logarithm	5		
geometry	2D		
grid	12x12		
super cell size	4x4		

25.5 References

[1]F. Pérez, L. Gremillet, A. Decoster, M. Drouin, and E. Lefebvre, Improved modeling of relativistic collisions and collisional ionization in particle-in-cell codes, Physics of Plasmas 19, 083104 (2012).

[2]D. P. Higginson, I. Holod, and A. Link, A corrected method for Coulomb scattering in arbitrarily weighted particle-in-cell plasma simulations, Journal of Computational Physics 413, 109450 (2020).

[3]J. Derouillat, A. Beck, F. Pérez, T. Vinci, M. Chiaramello, A. Grassi, M. Flé, G. Bouchard, I. Plotnikov, N. Aunai, J. Dargent, C. Riconda, and M. Grech, SMILEI: A collaborative, open-source, multi-purpose particle-in-cell code for plasma simulation, Computer Physics Communications 222, 351 (2018).

[4]K. Nanbu, Theory of cumulative small-angle collisions in plasmas, Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics 55, 4642 (1997).

[5]A. S. Richardson, NRL Plasma Formulary, (2019).

CHAPTER

TWENTYSIX

PYTHON

Section author: Axel Huebl, Klaus Steiniger

If you are new to python, get your hands on the tutorials of the following important libraries to get started.

- https://www.python.org/about/gettingstarted/
- https://docs.python.org/3/tutorial/index.html

An easy way to get a Python environment for analysis of PIConGPU data up and running is to download and install Miniconda

https://docs.conda.io/en/latest/miniconda.html

and set up a conda environment with the help of this conda environment file

https://gist.github.com/steindev/d19263d41b0964bcecdfb1f47e18a86e

(see documentation within the file).

26.1 Numpy

Numpy is the universal swiss army knife for working on ND arrays in python.

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

26.2 Matplotlib

One common way to visualize plots:

- http://matplotlib.org/faq/usage_faq.html#usage
- https://gist.github.com/ax3l/fc123cb94f59d440f952

26.3 Jupyter

Access, share, modify, run and interact with your python scripts from your browser:

https://jupyter.readthedocs.io

26.4 openPMD-viewer

An exploratory framework that visualizes and analyzes data in our HDF5 files thanks to their *openPMD markup*. Automatically converts units to SI, interprets iteration steps as time series, annotates axes and provides some domain specific analysis, e.g. for LWFA. Also provides an interactive GUI for fast exploration via Jupyter notebooks.

- Project Homepage
- Tutorial

26.5 openPMD-api

A data library that reads (and writes) data in our openPMD files (ADIOS2 and HDF5) to and from Numpy data structures. Provides an API to correctly convert units to SI, interprets iteration steps correctly, etc.

- Manual
- Examples

26.6 yt-project

With yt 3.4 or newer, our HDF5 output, which uses the *openPMD markup*, can be read, processed and visualized with yt.

- Project Homepage
- Data Loading
- Data Tutorial

CHAPTER

TWENTYSEVEN

OPENPMD

Section author: Axel Huebl

Module author: Axel Huebl

If you hear about *openPMD** for the first time you can find a quick online tutorial on it here.

As a user of PIConGPU, you will be mainly interested in our *python tools* and readers, that can read openPMD, e.g. into:

- read & write data: openPMD-api (manual)
- visualization and analysis, including an exploratory Jupyter notebook GUI: openPMD-viewer (tutorial)
- yt-project (tutorial)
- *ParaView*
- VisIt
- converter tools: openPMD-converter
- full list of projects using openPMD

If you intend to write your own post-processing routines, make sure to check out our example files, the formal, open standard on openPMD and a list of projects that already support openPMD.

CHAPTER

TWENTYEIGHT

PARAVIEW

Section author: Axel Huebl

Module author: Axel Huebl

Please see https://github.com/ComputationalRadiationPhysics/picongpu/wiki/ParaView for now.

CHAPTER

TWENTYNINE

DEVICE OVERSUBSCRIPTION

Module author: René Widera

By default the strategy to execute PIConGPU is that one MPI rank is using a single device e.g. a GPU. In some situation it could be beneficial to use multiple MPI ranks per device to get a better load balancing or better overlap communications with computation.

29.1 Usage

Follow *the description to pass command line parameter to PIConGPU*. PIConGPU provides the command line parameter `--numRanksPerDevice` or short `-r` to allow sharing a compute device between multiple MPI ranks. If you change the default value `1` to `2` PIConGPU is supporting two MPI processes per device.

Note: Using device oversubscription will limit the maximal memory footprint per PIConGPU MPI rank on the device too `<total available memory on device>/<number of ranks per device>`.

29.2 NVIDIA

29.2.1 Compute Mode

On NVIDIA GPUs there are different point which can influence the oversubscription of a device/GPU. NVIDIA Compute Mode must be ``Default` to allow multiple processes to use a single GPU. If you use device oversubscription with NVIDIA GPUs the kernel executed from different processes will be serialized by the driver, this is mostly describing the performance of PIConGPU because the device is under utilized.

29.2.2 Multi-Process Service (MPS)

If you use NVIDIA MPS and split one device into `4` you need to use `--numRanksPerDevice 4` for PIConGPU even if MPS is providing you with 4 virtual gpus. MPS can be used to workaround the kernel serialization when using multiple processes per GPU.

29.3 CPU

If you *compiled PIConGPU with a CPU accelerator* e.g. *omp2b*, *serial*, *tbb*, or *threads* device oversubscribing will have no effect. For CPU accelerators PIConGPU is not using a pre allocated device memory heap therefore you can freely choose the number of MPI ranks per CPU.

CHAPTER

THIRTY

PICONGPU SIGNALS

Sending signals to PIConGPU allows creating checkpoints during the run and a clean shutdown before the simulation arrived the end time step. Signal support is not available on WINDOWS operating systems.

Triggering a checkpoint with signals is only possible if you enabled a *checkpointing plugin*.

30.1 Overview

30.1.1 SIGNALS handled by PIConGPU on POSIX systems

- HUP (1): Triggers USR2. Controlling process/terminal hangup.
- INT (2): Triggers USR2. This SIGNAL gets triggert while hitting ^C.
- QUIT (3): Triggers USR2. This is the terminal quit SIGNAL.
- ABRT (6): Triggers USR2. Can only be called from within the code.
- USR1 (10): Create a checkpoint for the next time step.
- USR2 (12): Finish current loop and exit normally by setting time step `n_max = n`.
- ALRM (14): Trigger USR1 and USR2.
- TERM (15): Trigger USR1.

30.1.2 Default SIGNALS

These can not be handled:

- KILL (9)
- CONT (18)
- STOP (19)

30.2 Batch Systems

30.2.1 Slurm

Documenation: https://slurm.schedmd.com/scancel.html

```
scancel --signal=USR1 --batch <jobid>
```

30.2.2 IBM LSF

Documentation: https://www.ibm.com/docs/hu/spectrum-lsf/10.1.0?topic=job-send-signal `bkill -s USR1 <jobid>`

30.3 Reference to SIGNALS

LINUX SIGNALS:

- https://man7.org/linux/man-pages/man7/signal.7.html
- http://en.wikipedia.org/wiki/Unix_signal#POSIX_signals

CHAPTER

THIRTYONE

HOW TO PARTICIPATE AS A DEVELOPER

31.1 Contents

31.2 Code - Version Control

If you are familiar with git, feel free to jump to our *github workflow* section.

31.2.1 install git

Debian/Ubuntu:

- `sudo apt-get install git`
- make sure `git --version` is at least at version 1.7.10

Optional *one* of these. There are nice GUI tools available to get an overview on your repository.

- `gitk git-gui qgit gitg`

Mac:

- see here

Windows:

- see here

- just kidding, it's this link
- please use ASCII for your files and take care of line endings

Configure your global git settings:

- `git config --global user.name NAME`
- `git config --global user.email EMAIL@EXAMPLE.com`
- `git config --global color.ui "auto"` (if you like colors)
- `git config --global pack.threads "0"` (improved performance for multi cores)

You may even improve your level of awesomeness by:

- `git config --global alias.pr "pull --rebase"` (see how to *avoide merge commits*)
- `git config --global alias.pm "pull --rebase mainline"` (to sync with the mainline by `git pm dev`)
- `git config --global alias.st "status -sb"` (short status version)
- `git config --global alias.l "log --oneline --graph --decorate --first-parent"` (single branch history)
- `git config --global alias.la "log --oneline --graph --decorate --all"` (full branch history)
- `git config --global rerere.enable 1` (see git rerere)
- More `alias` tricks:
  - `git config --get-regexp alias` (show all aliases)
  - `git config --global --unset alias.<Name>` (unset alias `<Name>`)

31.2.2 git

Git is a *distributed* **version control system**. It helps you to keep your software development work organized, because it keeps track of *changes* in your project. It also helps to come along in **teams**, crunching on the *same project*. Examples:

- Arrr, dare you other guys! Why did you change my precious *main.cpp*, too!?
- Who introduced that awesome block of code? I would like to pay for a beer as a reward.
- Everything is wrong now, why did this happen and when?
- What parts of the code changed since I went on vacation (to a conference, phd seminar, mate fridge, …)?

If *version control* is totally **new** to you (that's good, because you are not spoiled) - please refer to a beginners guide first.

- git - the simple guide
- 15 minutes guide at try.github.io

Since git is *distributed*, no one really needs a server or services like github.com to *use git*. Actually, there are even very good reasons why one should use git even for **local** data, e.g. a master thesis (or your collection of ascii art dwarf hamster pictures).

Btw, **fun fact warning**: Linus Torvalds, yes the nice guy with the pinguin stuff and all that, developed git to maintain the **Linux kernel**. So that's cool, by definition.

A nice overview about the *humongous* number of tutorials can be found at stackoverflow.com … but we may like to start with a git **cheat sheet** (is there anyone out there who knows more than 1% of all git commands available?)

- git-tower.com (print the 1st page)
- github.com - "cheat git" gem (a cheat sheet for the console)

- kernel.org *Everyday GIT with 20 commands or so*
- an other interactive, huge cheat sheet (nice overview about stash - workspace - index - local/remote repositories)

Please spend a minute to learn how to write **useful** git commit **messages** (caption-style, maximum characters per line, use blank lines, present tense). Read our *commit rules* and use keywords.

If you like, you can **credit** someone else for your **next commit** with:

- `git commit --author "John Doe <johns-github-mail@example.com>"`

31.2.3 git for svn users

If you already used version control systems before, you may enjoy the git for svn users crash course.

Anyway, please keep in mind to use git *not* like a centralized version control system (e.g. *not* like svn). Imagine git as your *own private* svn server waiting for your commits. For example *Github.com* is only **one out of many** *sources for updates*. (But of course, we agree to share our *finished*, new features there.)

31.3 GitHub Workflow

Welcome to github! We will try to explain our coordination strategy (I am out of here!) and our development workflow in this section.

31.3.1 In a Nutshell

Create a *GitHub* account and prepare your *basic git config*.

Prepare your *forked* copy of our repository:

- fork picongpu on *GitHub*
- `git clone git@github.com:<YourUserName>/picongpu.git` (create local copy)
- `git remote add mainline git@github.com:ComputationalRadiationPhysics/picongpu.git` (add our main repository for updates)
- `git checkout dev` (switch to our, its now *your*, dev branch to start from)

Start a *topic/feature branch*:

- `git checkout -b <newFeatureName>` (start a new branch from dev and check it out)
- *hack hack*
- `git add <yourChangedFiles>` (add changed and new files to index)
- `git clang-format` (format all changed files with the clang-format utility, needs to be loaded or installed separately)
- `git add <yourChangedFiles>` (add the formating changes)
- `git commit` (commit your changes to your *local* repository)
- `git pull --rebase mainline dev` (update with our *remote dev* updates and avoid a merge commit)

Optional, *clean up* your feature branch. That can be *dangerous*:

- `git pull` (if you pushed your branch already to your public repository)
- `git pull --rebase mainline dev` (apply the mainline updates to your feature branch)
- `git log ..mainline/dev`, `git log --oneline --graph --decorate --all` (check for related commits and ugly merge commits)

- `git rebase mainline/dev` (re-apply your changes after a fresh update to the `mainline/dev`, see here)
- `git rebase -i mainline/dev` (squash related commits to reduce the complexity of the features history during a pull request)

Publish your feature and start a *pull request*:

- `git push -u origin <newFeatureName>` (push your local branch to your github profile)
- Go to your *GitHub* page and open a *pull request*, e.g. by clicking on *compare & review*
- Add additional updates (if requested to do so) by `push`-ing to your branch again. This will update the *pull request*.

31.3.2 How to fork from us

To keep our development fast and conflict free, we recomment you to fork our repository and start your work from our **dev** (development) branch in your private repository. Simply click the *Fork* button above to do so.

Afterwards, `git clone` **your** repository to your local machine. But that is not it! To keep track of the original **dev** repository, add it as another remote.

- `git remote add mainline https://github.com/ComputationalRadiationPhysics/picongpu.git`
- `git checkout dev` (go to branch **dev**)

Well done so far! Just start developing. Just like this? No! As always in git, start a *new branch* with `git checkout -b topic-<yourFeatureName>` and apply your changes there.

31.3.3 Keep track of updates

We consider it a **best practice** *not to modify* your **dev** branch at all.
Instead you can use it to `pull --ff-only` new updates from the original repository. Take care to **switch to dev** by `git checkout dev` to start **new feature branches** from **dev**.

So, if you like to do so, you can even keep track of the *original dev* branch that way. Just start your new branch with `git branch --track <yourFeatureName> mainline/dev` instead. This allows you to immediatly pull or fetch from **our dev** and avoids typing (during `git pull --rebase`). Nevertheless, if you like to `push` to *your* forked (== `origin`) repository, you have to say e.g. `git push origin <branchName>` explicitly.

You should **add updates** from the original repository on a **regular basis** or *at least* when you *finished your feature*.

- commit your local changes in your *feature branch*: `git commit`

Now you *could* do a normal *merge* of the latest `mainline/dev` changes into your feature branch. That is indeed possible, but will create an ugly merge commit. Instead try to first update *the point where you branched from* and apply your changes *again*. That is called a **rebase** and is indeed less harmful as reading the sentence before:

- `git checkout <yourFeatureName>`
- `git pull --rebase mainline dev` (in case of an emergency, hit `git rebase --abort`)

Now solve your conflicts, if there are any, and you got it! Well done!

31.3.4 Pull requests or *being social*

How to propose that **your awesome feature** (we know it will be awesome!) should be included in the **mainline PIConGPU** version?

Due to the so called pull requests in *GitHub*, this quite easy (yeah, sure). We start again with a *forked repository* of our own. You already created a **new feature branch** starting from our **dev** branch and commited your changes. Finally, you publish you local branch via a *push* to *your* GitHub repository: `git push -u origin <yourLocalBranchName>`

Now let's start a *review*. Open the *GitHub* homepage, go to your repository and switch to your *pushed feature branch*. Select the green **compare & review** button. Now compare the changes between **your feature branch** and **our dev**.

Everything looks good? Submit it as a **pull request** (link in the header). Please take the time to write an **extensive description**.

- What did you implement and why?
- Is there an open issue that you try to address (please link it)?
- Do not be afraid to add images!

The description of the pull request is essential and will be referred to in the change log of the next release.

Please consider to change only **one aspect per pull request** (do not be afraid of follow-up pull requests!). For example, submit a pull request with a bug fix, another one with new math implementations and the last one with a new awesome implementation that needs both of them. You will see, that speeds up *review time* a lot!

Speaking of those, a fruitful (*wuhu, we love you - don't be scared*) *discussion* about your **submitted change set** will start at this point. If we find some things you could *improve* (*That looks awesome, all right!*), simply change your *local feature branch* and *push the changes back* to your GitHub repository, to **update the pull request**. (You can now rebase follow-up branches, too.)

One of our *maintainers* will pick up the pull request to coordinate the review. Other regular developers that are competent in the topic might assist.

Sharing is caring! Thank you for participating, **you are great**!

31.3.5 maintainer notes

- do not *push* to the main repository on a regular basis, use **pull request** for your features like everyone else
- **never** do a *rebase* on the mainline repositories (this causes heavy problems for everyone who pulls them)
- on the other hand try to use pull –rebase to **avoid merge commits** (in your *local/topic branches* **only**)
- do not vote on your *own pull requests*, wait for the other maintainers
- we try to follow the strategy of a-successful-git-branching-model

Last but not least, help.github.com has a very nice FAQ section.

More best practices.

31.4 Commit Rules

See our *commit rules page*

31.5 Test Suite Examples

You know a useful setting to validate our provided methods? Tell us about it or add it to our test sets in the examples/ folder!

CHAPTER

THIRTYTWO

PICONGPU COMMIT RULEZ

We agree on the following simple rules to make our lives easier :)

- Stick to the **style** below for **commit messages**
- **Commit compiling patches** for the `dev` branch, you can be less strict for (unshared) *topic branches*
- Follow the code style and formatting which is democratically evolved in Contributing.

32.1 Pre-commit

Compliance with the coding style can be ensured (as much as automatically possible) by using pre-commit hooks (based on the more general but harder to use git hooks). After the following installation this little tool will run a number of checks prior to every commit, reject the commit, if they don't succeed, and potentially apply fixes.

`pre-commit` is a Python tool, so you need a working version of `python3`, e.g. from conda, your favorite package manager or directly from the website. For installation, we provide `requirements_pre-commit.txt` and you can use

```
# set up a virtual environment if you prefer, then:
$ python3 -m pip install -r requirements_pre-commit.txt

# run this inside of your local clone of the repo:
$ pre-commit install
```

From now on, each commit in this clone of the repo will be checked. See pre-commit for usage details. Some hints:

- You can run all hooks on all files via `pre-commit run --all-files [--hook-stage manual]`. The last argument `--hook-stage manual` includes rather slow additional tests that are run by the CI but are considered too heavy-weight to run before each commit.
- If a check fails, the fixes are often applied automatically (e.g., run `clang-format`). If there are no unstaged files, these changes will appear as unstaged changes in your working tree. To make the commit pass, you have to `git add` all changed files.
- In urgent cases, you can skip the checks via `git commit [...] --no-verify`. Be aware that similar things will be checked in CI during your PR and fail then at latest.

32.2 Manually Formatting Code

For C++ code, we provide `.clang-format` file in the root directory. Python code must adhere to PEP 8 guidelines. Following both of these is automated in `pre-commit`. If you are not able or willing to use `pre-commit`, you can instead do the following manually to get close to the same result:

For Python code, install `black` and run

```
black -l 120
```

The following describes formatting of C++ code.

- Install *ClangFormat 12* from LLVM 12.0.1
- To format all files in your working copy, you can run this command in bash from the root folder of PIConGPU:

  ```
  find include/ share/picongpu/ share/pmacc -iname "*.def" \
  -o -iname "*.h" -o -iname "*.cpp" -o -iname "*.cu" \
  -o -iname "*.hpp" -o -iname "*.tpp" -o -iname "*.kernel" \
  -o -iname "*.loader" -o -iname "*.param" -o -iname "*.unitless" \
  | xargs clang-format-12 -i
  ```

Instead of using the bash command above you can use *Git* together with *ClangFormat* to format your patched code only. Before applying this command, you must extend your local git configuration **once** with all file endings used in *PIConGPU*:

```
git config --local clangFormat.extensions def,h,cpp,cu,hpp,tpp,kernel,loader,param,
↪unitless
```

For only formatting lines you added using `git add`, call `git clang-format-12` before you create a commit. Please be aware that un-staged changes will not be formatted.

32.3 Commit Messages

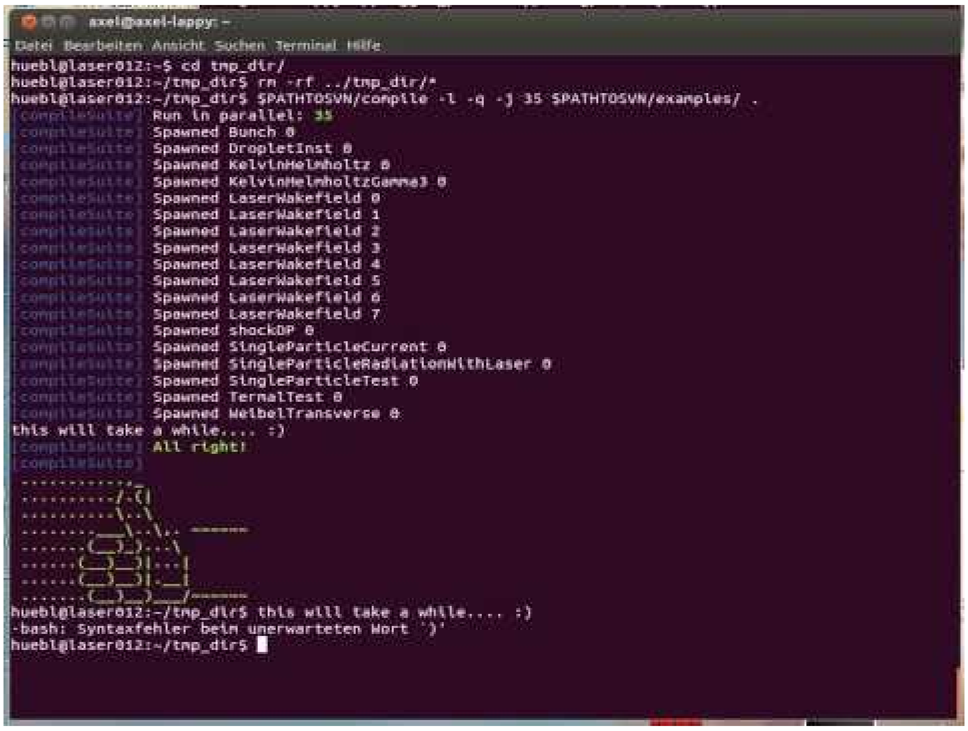
Let's go for an example:

> Use the 1st line as a topic, stay <= 50 chars
>
> - the blank line between the "topic" and this "body" is MANDATORY
> - use several key points with - or * for additional information
> - stay <= 72 characters in this "body" section
> - avoid blank lines in the body
> - Why? Pls refer to http://stopwritingramblingcommitmessages.com/

32.4 Compile Tests

We provide an (interactive/automated) **script** that **compiles all examples** within the `examples/` directory in your branch.

This helps a lot to **maintain various combinations** of options in the code (like different solvers, boundary conditions, . . .).



```
huebl@laser012:~$ cd tmp_dir/
huebl@laser012:~/tmp_dir$ rm -rf ../tmp_dir/*
huebl@laser012:~/tmp_dir$ $PATHTOSVN/compile -l -q -j 35 $PATHTOSVN/examples/ .
[compileSuite] Run in parallel: 35
[compileSuite] Spawned Bunch 0
[compileSuite] Spawned DropletInst 0
[compileSuite] Spawned KelvinHelmholtz 0
[compileSuite] Spawned KelvinHelmholtzGamma3 0
[compileSuite] Spawned LaserWakefield 0
[compileSuite] Spawned LaserWakefield 1
[compileSuite] Spawned LaserWakefield 2
[compileSuite] Spawned LaserWakefield 3
[compileSuite] Spawned LaserWakefield 4
[compileSuite] Spawned LaserWakefield 5
[compileSuite] Spawned LaserWakefield 6
[compileSuite] Spawned LaserWakefield 7
[compileSuite] Spawned shockDP 0
[compileSuite] Spawned SingleParticleCurrent 0
[compileSuite] Spawned SingleParticleRadiationWithLaser 0
[compileSuite] Spawned SingleParticleTest 0
[compileSuite] Spawned TermalTest 0
[compileSuite] Spawned WeibelTransverse 0
this will take a while.... :)
[compileSuite] All right!
[compileSuite]
 ...........,_
 ..........\/.(|
 ..........\..\
 ........___\..\.. ~~~~~~
 .......(___)...\
 ......(___)|...|
 ......(___)|.__|
 .......(__)__)___/~~~~~~
huebl@laser012:~/tmp_dir$ this will take a while.... :)
-bash: Syntaxfehler beim unerwarteten Wort `)'
huebl@laser012:~/tmp_dir$ █
```

Assume

- `repo=<pathToYourPIConGPUgitDirectory>`
- `tmpPath=<tmpFolder>`

Now run the tests with

- `$repo/compile -l $repo/examples/ $tmpPath`

Further options are:

- `-q :  continue on errors`
- `-j <N> :  run <N> tests in parallel (note:  do NOT omit the number <N>)`

If you ran your test with, let's say `-l -q -j 4`, and you got errors like

> [compileSuite] [error] In PIC_EXTENSION_PATH:PATH=.../params/KelvinHelmholtz/cmakePreset_0: CMAKE_INSTALL_PREFIX:PATH=.../params/KelvinHelmholtz/cmakePreset_0 (.../build) make install

check the specific test's output (in this case `examples/KelvinHelmholtz` with *CMake preset #0*) with:

- `less -R $tmpPath/build/build_ThermalTest_cmakePreset_0/compile.log`

32.4.1 Compile Tests - Single Example

Compile **all CMake presets** of a *single example* with:

- `$repo/compile $repo/examples/ $tmpPath`

32.4.2 Compile Tests - Cluster Example:

- Request an interactive job (to release some load from the head node) `qsub -I -q laser -lwalltime=03:00:00 -lnodes=1:ppn=64`
- Use a non-home directory, e.g. `tmpPath=/net/cns/projects/HPL/<yourTeam>/<yourName>/tmp_tests/`
- Compile like a boss! `<pathToYourPIConGPUgitDirectory>/compile -l -q -j 60 <pathToYourPIConGPUgitDirectory>/examples/ $tmpPath`
- Wait for the **thumbs up/down** :)

CHAPTER

THIRTYTHREE

REPOSITORY STRUCTURE

Section author: Axel Huebl

33.1 Branches

- `dev`: the development branch where all features start from and are merged to
- `release-X.Y.Z`: release candiate for version `X.Y.Z` with an upcoming release, receives updates for bug fixes and documentation such as change logs but usually no new features

33.2 Directory Structure

- `include/`
  - C++ header *and* source files
  - set `-I` here
  - prefixed with project name
- `lib/`
  - pre-compiled libraries
  - `python/`
    - python modules
    - set `PYTHONPATH` here
    - `extra/`
      - modules, e.g. for RT interfaces, pre* & post-processing
    - `picmi/`
      - user-facing python interface
    - `pypicongpu/`
      - internal interface for `.param` & `.cfg`-file generation
      - used by PICMI implementation
- `etc/`
  - (runtime) configuration files
  - `picongpu/`
    - `tbg` templates (as long as PIConGPU specific, later on to `share/tbg/`)
    - network configurations (e.g. infiniband)

        * score-p and vampir-trace filters

- `share/`
  - examples, documentation
  - `picongpu/`
    * `benchmarks/`: setups used for benchmarking
    * `completions/`: bash completions
    * `examples/`: each with same structure as `/`
    * `pypicongpu/`: required files for code generation
      · `schema/`: code generation JSON schemas
      · `template/`: base template for code generation
    * `tests/`: self-validating examples for different scenarios
    * `unit/`: small (hopefully expanding) selection of unit tests
- `bin/`
  - core tools for the "PIConGPU framework"
  - set `PATH` here
- `docs/`
  - currently for the documentation files
  - might move, e.g. to `lib/picongpu/docs/` and its build artifacts to `share/{doc,man}/`,

CHAPTER

THIRTYFOUR

CODING GUIDE LINES

Section author: Axel Huebl

See also:

Our coding guide lines are documented in this repository.

34.1 Source Style

For contributions, *an ideal patch blends in the existing coding style around it* without being noticed as an addition when applied. Nevertheless, please make sure *new files* follow the styles linked above as strict as possible from the beginning.

clang-format-12 (version 12.0.1) should be used to format the code. There are different ways to format the code.

34.1.1 Format All Files

To format all files in your working copy, you can run this command in bash from the root folder of PIConGPU:

```
find include/ share/picongpu/ share/pmacc -iname "*.def" \
  -o -iname "*.h" -o -iname "*.cpp" -o -iname "*.cu" \
  -o -iname "*.hpp" -o -iname "*.tpp" -o -iname "*.kernel" \
  -o -iname "*.loader" -o -iname "*.param" -o -iname "*.unitless" \
  | xargs clang-format-12 -i
```

34.1.2 Format Only Changes, Using Git

Instead of using the bash command above you can use *Git* together with *ClangFormat* to format your patched code only.

> *ClangFormat* is an external tool for code formating that can be called by *Git* on changed files only and is part of clang tools.

Before applying this command, you must extend your local git configuration **once** with all file endings used in *PIConGPU*:

```
git config --local clangFormat.extensions def,h,cpp,cu,hpp,tpp,kernel,loader,param,
↪unitless
```

After installing, or on a cluster loading the module(see introduction), clangFormat can be called by git on all **staged files** using the command:

```
git clangFormat
```

Warning: The binary for *ClangFormat* is called *clang-format* on some operating systems. If *clangFormat* is not recognized, try *clang-format* instead, in addition please check that *clang-format –version* returns version *11.X.X* in this case.

The Typical workflow using git clangFormat is the following,

1. make your patch
2. stage the changed files in git

```
git add <files you changed>/ -A
```

3. format them according to guidelines

```
git clangFormat
```

4. stage the now changed(formated) files again

```
git add <files you changed>
```

5. commit changed files

```
git commit -m <commit message>
```

Please be aware that un-staged changes will not be formatted. Formatting all changes of the previous commit can be achieved by executing the command *git clang-format-12 HEAD~1*.

34.2 License Header

Please **add the according license header** snippet to your *new files*:

- for PIConGPU (GPLv3+): `src/tools/bin/addLicense <FileName>`
- for libraries (LGPLv3+ & GPLv3+): `export PROJECT_NAME=PMacc && src/tools/bin/addLicense <FileName>`
- delete other headers: `src/tools/bin/deleteHeadComment <FileName>`
- add license to all `.hpp` files within a directory (recursive): `export PROJECT_NAME=PIConGPU && src/tools/bin/findAndDo <PATH> "*.hpp" src/tools/bin/addLicense`
- the default project name is `PIConGPU` (case sensitive!) and add the GPLv3+ only

Files in the directory `thirdParty/` are only imported from remote repositories. If you want to improve them, submit your pull requests there and open an issue for our **maintainers** to update to a new version of the according software.

CHAPTER

THIRTYFIVE

SPHINX

Section author: Axel Huebl, Marco Garten

In the following section we explain how to contribute to this documentation.

If you are reading the HTML version on http://picongpu.readthedocs.io and want to improve or correct existing pages, check the "Edit on GitHub" link on the right upper corner of each document.

Alternatively, go to *docs/source* in our source code and follow the directory structure of reStructuredText (`.rst`) files there. For intrusive changes, like structural changes to chapters, please open an issue to discuss them beforehand.

35.1 Build Locally

This document is build based on free open-source software, namely Sphinx, Doxygen (C++ APIs as XML) and Breathe (to include doxygen XML in Sphinx). A web-version is hosted on ReadTheDocs.

The following requirements need to be installed (once) to build our documentation successfully:

```
cd docs/

# doxygen is not shipped via pip, install it externally,
# from the homepage, your package manager, conda, etc.
# example:
sudo apt-get install doxygen

# python tools & style theme
pip install -r requirements.txt # --user
```

In order to not break any of your existing Python configurations, you can also create a new environment that you only use for building the documentation. Since it is possible to install doxygen with conda, the following demonstrates this.

```
cd docs/

# create a bare conda environment containing just all the requirements
# for building the picongpu documentation
# note: also installs doxygen inside this environment
conda env create --file picongpu-docs-env.yml

# start up the environment as suggested during its creation e.g.
conda activate picongpu-docs-env
# or
source activate picongpu-docs-env
```

With all documentation-related software successfully installed, just run the following commands to build your docs locally. Please check your documentation build is successful and renders as you expected before opening a pull request!

```
# skip this if you are still in docs/
cd docs/

# parse the C++ API documentation,
#   enjoy the doxygen warnings!
doxygen
# render the `.rst` files and replace their macros within
#   enjoy the breathe errors on things it does not understand from doxygen :)
make html

# open it, e.g. with firefox :)
firefox build/html/index.html

# now again for the pdf :)
make latexpdf

# open it, e.g. with okular
build/latex/PIConGPU.pdf
```

35.2 Useful Links

- A primer on writing restFUL files for sphinx
- Why You Shouldn't Use "Markdown" for Documentation
- Markdown Limitations in Sphinx

CHAPTER

THIRTYSIX

DOXYGEN

Section author: Axel Huebl

PIConGPU uses *Doxygen* for API documentation. Please provide the corresponding annotations in new and updated code as needed. To build this documentation do the following:

36.1 Requirements

Install Doxygen and its dependencies for graph generation.

```
# install requirements (Debian/Ubuntu)
sudo apt-get install doxygen graphviz
```

36.2 Activate HTML Output

Activate the generation of html files in the doxygen config file

```
# enable HTML output in our Doxyfile
sed -i 's/GENERATE_HTML.*=.*NO/GENERATE_HTML     = YES/' docs/Doxyfile
```

36.3 Build

Now run the following commands to build the Doxygen HTML documentation locally.

```
cd docs/

# build the doxygen HTML documentation
doxygen

# open the generated HTML pages, e.g. with firefox
firefox html/index.html
```

CHAPTER

THIRTYSEVEN

CLANG TOOLS

Section author: Axel Huebl

We are currently integrating support for Clang Tools [ClangTools] such as clang-tidy and clang-format. Clang Tools are fantastic for static source code analysis, e.g. to find defects, automate style formatting or modernize code.

37.1 Install

At least LLVM/Clang 3.9 or newer is required. On Debian/Ubuntu, install them via:

```
sudo apt-get install clang-tidy-3.9
```

37.2 Usage

Currently, those tools work only with CPU backends of PIConGPU. For example, enable the *OpenMP* backend via:

```
# in an example
mkdir build
cd build

pic-configure -c"-DALPAKA_ACC_CPU_B_OMP2_T_SEQ_ENABLE=ON" ..
```

We try to auto-detect clang-tidy. If that fails, you can set a manual hint to an adequate version via -DCLANG_TIDY_BIN in CMake:

```
pic-configure -c"-DALPAKA_ACC_CPU_B_OMP2_T_SEQ_ENABLE=ON -DCLANG_TIDY_BIN=$(which␣
↪clang-tidy-3.9)" ..
```

If a proper version of clang-tidy is found, we add a new clang-tidy build target:

```
# enable verbose output to see all warnings and errors
make VERBOSE=true clang-tidy
```

CHAPTER

THIRTYEIGHT

EXTENDING PICONGPU

Section author: Sergei Bastrakov

Note: A number of places in *.param files* allow providing user-defined functors. The processing logic can largely be customized on that level, without modifying the source code. Such an external way is recommended when applicable. This section covers the case of extending the internal implementation.

38.1 General Simulation Loop Structure

A PIConGPU simulation effectively boils down to performing initialization and then executing a simulation loop. The simulation loop iterates over the requested number of time steps. For each time step, first all enabled plugins are called. Then all core computational stages are executed. Details of creating a new plugin or a new stage are presented below.

38.2 Adding a Plugin

PIConGPU plugins can perform computations or output, and often do both. Since all plugins are called at iteration start, they normally implement operations that are independent of the place in the computational loop. For operations requiring a specific ordering, please refer to the next section.

To add a new plugin, make a new file or subdirectory inside `include/picongpu/plugins`. Each plugin class must inherit our base class `ISimulationPlugin`. In turn, it largely uses its own base class `pmacc::ISimulationPlugin`. These classes define the interface of all PIConGPU plugins. The methods that most plugins want to override are:

- `pluginRegisterHelp()` adds command-line options for the plugin. In case a plugin introduces some new compile-time parameters, they are normally put to a new `.param` file.
- `pluginGetName()` sets a text name for the plugin, used to report errors currently.
- `pluginLoad()` initializes internal data of the plugin after the command-line arguments are submitted. Note that a constructor of the plugin class would be called before that and so often cannot do a full initialization. Is called once upon simulation start.
- `pluginUnload()` finalizes internal data if necessary, is called once at the end of the simulation.
- `setMappingDescription()` is used to pass simulation data to be used in kernels
- `notify()` runs the plugin for the given time iteration. This method implements the computational logic of the plugin. It often involves calling an internal algorithm or writing an own kernel. Those are described in the following sections.
- `checkpoint()` saves plugin internal data for checkpointing if necessary
- `restart()` loads the internal data from a checkpoint (necessary if `checkpoint()` writes some data)

Most plugins are run with a certain period in time steps. In this case, `Environment<>::get().PluginConnector().setNotificationPeriod(this, notifyPeriod)` can be used inside `pluginLoad()` to set this period. Then `notify()` will only be called for the active time steps.

For plugins (and most PIConGPU code) dealing with particles, it is common to template-parametrize based on species. Such plugins should use base class `plugins::multi::IInstance`. There is also a helper class `plugins::multi::IHelp` for command-line parameters prefixed for species. To match a plugin to applicable species, partially specialize trait `particles::traits::SpeciesEligibleForSolver`.

Regardless of the base class used, the new plugin class must be instantiated at `picongpu::PluginController` and the new headers included there. In case the plugin is conditionally available (e.g. requires an optional dependency), guards must also be placed there, around the include and instantiation.

When adding a plugin, do not forget to extend the documentation with plugin description and parameters. At least, extend the list of *plugins* and *command-line parameters* (the latter via `TBG_macros.cfg`). Other welcome additions for a new plugin include a dedicated documentation, a new example demonstrating usage in a realistic scenario, and a Python postprocessing script.

38.3 Adding a Simulation Stage

The currently executed simulation stages and their order are defined inside `Simulation::runOneStep()`. All stage classes are in `namespace picongpu::simulation::stage` and their implementations are located in respective directory. The existing stages share compatible interface and thus follow a pseudo-concept, not formally defined currently. The interface is also compatible to functors passed to `InitPipeline` and `IterationStartPipeline`. Note that stage classes are just wrappers around the internal implementations. Their purpose is to offer a high-level view of the steps and data dependencies in the computational loop, and add command-line parameters if necessary.

To add a new simulation stage:

- create a new `.hpp` file in `picongpu/simulation/stage`.
- write your stage functor class following the interface of other stage functors.
- if needed (e.g. for command-line parameters or static data) store an instance of it as a member of `Simulation` class.
- add a call to it inside `Simulation::runOneStep()`.

As mentioned above, a simulation stage should merely be calling an actual implementation located at an appropriate place inside PIConGPU or pmacc. When possible, it is recommended to use existing generic algorithms like `particles::Manipulate<>` or `FieldTmp::computeValue()`. Otherwise, one often has to implement an own kernel.

38.4 Writing a Kernel

Computational kernels are written using library alpaka. Most kernel functors are templates parametrized with the number of threads per block, often called `numWorkers`. Kernel invocations are wrapped into a helper macro `PMACC_KERNEL` or `PMACC_LOCKSTEP_KERNEL`.

A vast majority of PIConGPU kernels operate on two levels of parallelism: between supercells and inside each supercell. This parallel pattern is covered by the mapper concept.

template<uint32_t **T_areaType**, template<unsigned, typename> class **T_MappingDescription**, unsigned **T_dim**, typename **T_SupercellSize**>
class **MapperConcept** : public *T_MappingDescription*<*T_dim*, *T_SupercellSize*>

Concept for mapper from block indices to supercells in the given area for alpaka kernels.

The mapping covers the case of supercell-level parallelism between alpaka blocks. The supercells can be processed concurrently in any order. This class is not used directly, but defines a concept for such mappers.

Mapper provides a 1:1 mapping from supercells in the area to alpaka blocks. (Since it is 1:1, the mapping is invertible, but only this direction is provided.) Dimensionality of the area indices and block indices is the same. A kernel must be launched with exactly getGridDim() blocks. Each block must process a single supercell with index getSuperCellIndex(blockIndex). Implementation is optimized for the standard areas in AreaType.

In-block parallelism is independent of this mapping and is done by a kernel. Naturally, this parallelism should also define block size, again independent of this mapping.

This pattern is used in most kernels in particle-mesh codes. Two most common parallel patterns are:

- for particle or particle-grid operations: alpaka block per supercell with this mapping, thread-level parallelism between particles of a frame
- for grid operations: alpaka block per supercell with this mapping, thread-level parallelism between cells of a supercell

Template Parameters

- `T_areaType` – parameter describing area to be mapped (depends on mapper type)
- `T_MappingDescription` – mapping description type, base class for *MapperConcept*
- `T_dim` – dimensionality of area and block indices
- `T_SupercellSize` – compile-time supercell size

For this parallel pattern, a mapper object provides the number of blocks to use for a kernel. On the device side, the object provides a mapping between alpaka blocks and supercells to be processed. Parallelism for threads between blocks is done inside the kernel. It is often over cells in a supercell or particles in a frame using *lockstep programming*.

A kernel often takes one or several data boxes from the host side. The data boxes allow array-like access to data. A more detailed description of boxes and other widely used classes is given in the following sections.

CHAPTER

THIRTYNINE

IMPORTANT PICONGPU CLASSES

This is very, very small selection of classes of interest to get you started.

39.1 Simulation

class **Simulation** : public *pmacc*::*SimulationHelper*<*simDim*>

Global simulation controller class.

Initialises simulation data and defines the simulation steps for each iteration.

Template Parameters

DIM – the dimension (2-3) for the simulation

Public Functions

Simulation() = default

Constructor.

inline virtual void **pluginRegisterHelp**(po::options_description &desc) override

Register command line parameters for this plugin.

Parameters are parsed and set prior to plugin load.

Parameters

desc – boost::program_options description

inline virtual void **startSimulation**() override

Begin the simulation.

inline nlohmann::json **metadata**() const

inline virtual std::string **pluginGetName**() const override

Return the name of this plugin for status messages.

Returns

plugin name

inline virtual void **pluginLoad**() override

inline virtual void **pluginUnload**() override

inline virtual void **notify**(uint32_t) override

Notification callback.

For example Plugins can set their requested notification frequency at the PluginConnector

Parameters

currentStep – current simulation iteration step inline virtual void **`init`**`()` override

Initialize simulation.

Does hardware selections/reservations, memory allocations and initializes data structures as empty.

inline virtual uint32_t **`fillSimulation`**`()` override

Fills simulation with initial data after *init()*

Returns

returns the first step of the simulation (can be >0 for, e.g., restarts from checkpoints)

inline virtual void **`runOneStep`**`(`uint32_t currentStep`)` override

Run one simulation step.

Parameters

`currentStep` – iteration number of the current step

inline virtual void **`movingWindowCheck`**`(`uint32_t currentStep`)` override

Check if moving window work must do.

If no moving window is needed the implementation of this function can be empty

Parameters

`currentStep` – simulation step

inline virtual void **`resetAll`**`(`uint32_t currentStep`)` override

Reset the simulation to a state such as it was after *init()* but for a specific time step.

Can be used to call *fillSimulation()* again.

inline void **`slide`**`(`uint32_t currentStep`)`

inline virtual void **`setInitController`**`(`IInitPlugin *initController`)`

inline *MappingDesc* ***`getMappingDescription`**`()`

39.2 FieldE

class **`FieldE`** : public *picongpu*::*fields*::EMFieldBase<*FieldE*>

Representation of the electric field.

Stores field values on host and device and provides data synchronization between them.

Implements interfaces defined by *SimulationFieldHelper< MappingDesc >* and ISimulationData.

39.3 FieldB

class **`FieldB`** : public *picongpu*::*fields*::EMFieldBase<*FieldB*>

Representation of the magnetic field.

Stores field values on host and device and provides data synchronization between them.

Implements interfaces defined by *SimulationFieldHelper< MappingDesc >* and ISimulationData.

39.4 FieldJ

class **FieldJ** : public *pmacc*::*SimulationFieldHelper*<*MappingDesc*>, public *pmacc*::ISimulationData

Representation of the current density field.

Stores field values on host and device and provides data synchronization between them.

Implements interfaces defined by *SimulationFieldHelper< MappingDesc >* and ISimulationData.

39.5 FieldTmp

class **FieldTmp** : public *pmacc*::*SimulationFieldHelper*<*MappingDesc*>, public *pmacc*::ISimulationData

Representation of the temporary scalar field for plugins and temporary particle data mapped to grid (charge density, energy density, etc.)

Stores field values on host and device and provides data synchronization between them.

Implements interfaces defined by *SimulationFieldHelper< MappingDesc >* and ISimulationData.

39.6 Particles

template<typename **T_Name**, typename **T_Flags**, typename **T_Attributes**>

class **Particles** : public *pmacc*::*ParticlesBase*<ParticleDescription<*T_Name*, std::integral_constant<uint32_t, *numFrameSlots*>, *SuperCellSize*, *T_Attributes*, *T_Flags*, *pmacc*::mp_if<*pmacc*::mp_contains<*T_Flags*, GetKeyFromAlias<*T_Flags*, boundaryCondition<>>::type>, *pmacc*::traits::Resolve<GetKeyFromAlias<*T_Flags*, boundaryCondition<>>::type>::type, *pmacc*::HandleGuardRegion<*pmacc*::particles::policies::ExchangeParticles, *pmacc*::particles::policies::DoNothing>>>, *MappingDesc*, *DeviceHeap*>, public *pmacc*::ISimulationData

particle species

Template Parameters

- **T_Name** – name of the species [type PMACC_CSTRING]
- **T_Attributes** – sequence with attributes [type boost::mp11 list]
- **T_Flags** – sequence with flags e.g. solver [type boost::mp11 list]

Public Types

using **SpeciesParticleDescription** = *pmacc*::*ParticleDescription*<*T_Name*, std::integral_constant<uint32_t, *numFrameSlots*>, *SuperCellSize*, *T_Attributes*, *T_Flags*, *pmacc*::mp_if<*pmacc*::mp_contains<*T_Flags*, typename GetKeyFromAlias<*T_Flags*, boundaryCondition<>>::type>, typename *pmacc*::traits::Resolve<typename GetKeyFromAlias<*T_Flags*, boundaryCondition<>>::type>::type, *pmacc*::HandleGuardRegion<*pmacc*::particles::policies::ExchangeParticles, *pmacc*::particles::policies::DoNothing>>>

using **ParticlesBaseType** = ParticlesBase<*SpeciesParticleDescription*, *picongpu*::*MappingDesc*, *DeviceHeap*>

using **FrameType** = typename *ParticlesBaseType*::FrameType using **FrameTypeBorder** = typename *ParticlesBaseType*::FrameTypeBorder

using **ParticlesBoxType** = typename *ParticlesBaseType*::ParticlesBoxType

Public Functions

Particles(const std::shared_ptr<*DeviceHeap*> &heap, *picongpu*::*MappingDesc* cellDescription, SimulationDataId datasetID)

void **createParticleBuffer**()

void **update**(uint32_t const currentStep)

Push all particles.

template<typename **T_MapperFactory**>
inline void **shiftBetweenSupercells**(*T_MapperFactory* const &mapperFactory, bool onlyProcessMustShiftSupercells)

Update the supercell storage for particles in the area according to particle attributes.

Template Parameters

T_MapperFactory – factory type to construct a mapper that defines the area to process

Parameters

- **mapperFactory** – factory instance
- **onlyProcessMustShiftSupercells** – whether to process only supercells with mustShift set to true (optimization to be used with particle pusher) or process all supercells

void **applyBoundary**(uint32_t const currentStep)

Apply all boundary conditions.

template<typename **T_DensityFunctor**, typename **T_PositionFunctor**>
void **initDensityProfile**(*T_DensityFunctor* &densityFunctor, *T_PositionFunctor* &positionFunctor, const uint32_t currentStep)

template<typename **T_SrcName**, typename **T_SrcAttributes**, typename **T_SrcFlags**, typename **T_ManipulateFunctor**, typename **T_SrcFilterFunctor**>
void **deviceDeriveFrom**(*Particles*<*T_SrcName*, *T_SrcAttributes*, *T_SrcFlags*> &src, *T_ManipulateFunctor* &manipulateFunctor, *T_SrcFilterFunctor* &srcFilterFunctor)

virtual SimulationDataId **getUniqueId**() override

Return the globally unique identifier for this simulation data.

Returns

globally unique identifier

virtual void **synchronize**() override

Synchronizes simulation data, meaning accessing (host side) data will return up-to-date values.

virtual void **syncToDevice**() override

Synchronize data from host to device.

template<typename **T_Pusher**>

void **push**(uint32_t const currentStep)

Do the particle push stage using the given pusher.

Template Parameters

T_Pusher – non-composite pusher type

Parameters

currentStep – current time iteration

Public Static Functions

static inline std::array<*particles*::boundary::Description, *simDim*> &**boundaryDescription**()

Get boundary descriptions for the species.

For both sides along the same axis, both boundaries have the same description. Must not be modified outside of the ParticleBoundaries simulation stage.

This method is static as it is used by static getStringProperties().

static inline *pmacc*::traits::StringProperty **getStringProperties**()

39.7 ComputeGridValuePerFrame

template<class **T_ParticleShape**, class **T_DerivedAttribute**>

class **ComputeGridValuePerFrame**

Public Types

using **AssignmentFunction** = typename *T_ParticleShape*::ChargeAssignment

using **LowerMargin** = typename *pmacc*::math::CT::make_Int<*simDim*, *lowerMargin*>::type

using **UpperMargin** = typename *pmacc*::math::CT::make_Int<*simDim*, *upperMargin*>::type

Public Functions

HDINLINE **ComputeGridValuePerFrame**() = default

HDINLINE float1_64 getUnit () const

return unit for this solver

Returns

solver unit

HINLINE std::vector< float_64 > getUnitDimension () const

return powers of the 7 base measures for this solver

characterizing the unit of the result of the solver in SI (length L, mass M, time T, electric current I, thermodynamic temperature theta, amount of substance N, luminous intensity J)

template<typename T_Particle, typename TVecSuperCell, typename BoxTmp, typename T_Worker, typename T_AccFilter> DINLINE void operator() (T_Worker const &worker, T_Particle &particle, const TVecSuperCell superCell, T_AccFilter &accFilter, BoxTmp &tmpBox)

Public Static Functions

static HINLINE std::string getName ()

return name of the this solver

Returns

name of solver

Public Static Attributes

static constexpr int **supp** = *AssignmentFunction*::support

static constexpr int **lowerMargin** = *supp* / 2

static constexpr int **upperMargin** = (*supp* + 1) / 2

CHAPTER

FORTY

IMPORTANT PMACC CLASSES

This is very, very small selection of classes of interest to get you started.

Note: Please help adding more Doxygen doc strings to the classes described below. As an example, here is a listing of possible extensive docs that new developers find are missing: https://github.com/ComputationalRadiationPhysics/picongpu/issues/776

40.1 Environment

template<uint32_t **T_dim**>

class **Environment** : public *pmacc*::detail::Environment

Global *Environment* singleton for PMacc.

Public Functions

HINLINE void enableMpiDirect ()

HINLINE bool isMpiDirectEnabled () const

HINLINE pmacc::GridController< T_dim > & GridController ()

get the singleton GridController

Returns

instance of GridController

HINLINE pmacc::SubGrid< T_dim > & SubGrid ()

get the singleton SubGrid

Returns

instance of SubGrid

HINLINE pmacc::Filesystem< T_dim > & Filesystem ()

get the singleton Filesystem

Returns

instance of Filesystem

HINLINE void initDevices (DataSpace< T_dim > devices, DataSpace< T_dim > periodic)

create and initialize the environment of PMacc

Usage of MPI or device(accelerator) function calls before this method are not allowed.

Parameters

- **devices** – number of devices per simulation dimension
- **periodic** – periodicity each simulation dimension (0 == not periodic, 1 == periodic)

HINLINE void initGrids (DataSpace< T_dim > globalDomainSize, DataSpace< T_dim > localDomainSize, DataSpace< T_dim > localDomainOffset)

initialize the computing domain information of PMacc

Parameters

- **globalDomainSize** – size of the global simulation domain [cells]
- **localDomainSize** – size of the local simulation domain [cells]
- **localDomainOffset** – local domain offset [cells]

Environment(const *Environment*&) = delete

Environment &**operator=**(const *Environment*&) = delete

Public Static Functions

static inline *Environment*<*T_dim*> &**get**()

get the singleton Environment< DIM >

Returns

instance of Environment<DIM >

40.2 DataConnector

class **DataConnector**

Singleton class which collects and shares simulation data.

All members are kept as shared pointers, which allows their factories to be destroyed after sharing ownership with our *DataConnector*.

Public Functions

bool **hasId**(SimulationDataId id)

Returns if data with identifier id is shared.

Parameters

id – id of the Dataset to query

Returns

if dataset with id is registered

void **`initialise`**(AbstractInitialiser &initialiser, uint32_t currentStep)

Initialises all Datasets using initialiser.

After initialising, the Datasets will be invalid.

Parameters

- **`initialiser`** – class used for initialising Datasets
- **`currentStep`** – current simulation step

void **`share`**(const std::shared_ptr<ISimulationData> &data)

Register a new Dataset and share its ownership.

If a Dataset with the same id already exists, a runtime_error is thrown. (Check with *DataConnector::hasId* when necessary.)

Parameters
`data` – simulation data to share ownership

void **`consume`**(std::unique_ptr<ISimulationData> data)

Register a new Dataset and transfer its ownership.

If a Dataset with the same id already exists, a runtime_error is thrown. (Check with *DataConnector::hasId* when necessary.) The only difference from *share()* is transfer of ownership.

Parameters
`data` – simulation data to transfer ownership

void **`deregister`**(SimulationDataId id)

End sharing a dataset with identifier id.

Parameters
`id` – id of the dataset to remove

void **`clean`**()

Unshare all associated datasets.

template<class **`TYPE`**>
inline std::shared_ptr<*TYPE*> **`get`**(SimulationDataId id)

Returns shared pointer to managed data.

Reference to data in Dataset with identifier id and type TYPE is returned. Increments the reference counter to the dataset specified by id.

Template Parameters
`TYPE` – if of the data to load

Parameters
`id` – id of the Dataset to load from

Returns
returns a reference to the data of type TYPE

Friends

`friend struct` detail::Environment

40.3 DataSpace

template<unsigned **T_dim**>

class **DataSpace** : public *pmacc*::math::*Vector*<int, *T_dim*>

A T_dim-dimensional vector of integers.

The object is used to describe indices of threads in the kernel. *DataSpace* describes a T_dim-dimensional data space with a specific size for each dimension. It only describes the space and does not hold any actual data.

Attention

Currently this object is also used to describe memory extents. This should be avoided and is a technical dept from the past and limits the index space to 2 giga elements.

Template Parameters

T_dim – dimension N of the dataspace

Public Types

using **BaseType** = math::*Vector*<int, *T_dim*>

Public Functions

inline HDINLINE **DataSpace**()

default constructor.

Sets size of all dimensions to 0.

constexpr HDINLINE **DataSpace**(const *DataSpace*&) = default

constexpr HDINLINE DataSpace & operator= (const DataSpace &)=default

template<typename **T_MemberType**>
inline explicit HDINLINE **DataSpace**(alpaka::Vec<::alpaka::DimInt<*T_dim*>, *T_MemberType*> const &value)

constructor.

Sets size of all dimensions from alpaka.

template<typename ...**T_Args**, typename = std::enable_if_t<(std::is_convertible_v<*T_Args*, int> && ...)>>
inline constexpr HDINLINE **DataSpace**(*T_Args*&&... args)

Constructor for N-dimensional *DataSpace*.

Attention

This constructor allows implicit casts.

Parameters

args – size of each dimension, x,y,z,…

inline constexpr HDINLINE **DataSpace**(const *BaseType* &vec)

inline HDINLINE **DataSpace**(const math::Size_t<*T_dim*> &vec)

`inline HDINLINE int getDim () const`

Returns number of dimensions (T_dim) of this *DataSpace*.

Returns
number of dimensions

`inline HINLINE bool isOneDimensionGreaterThan (const MemSpace< T_dim > &other) const`

Evaluates if one dimension is greater than the respective dimension of other.

Parameters
`other` – *DataSpace* to compare with

Returns
true if one dimension is greater, false otherwise

inline HDINLINE **`operator math::Size_t<`*`T_dim`*`>`**() const

Public Static Functions

`static inline HDINLINE DataSpace< T_dim > create (int value=1)`

Give *DataSpace* where all dimensions set to init value.

Parameters
`value` – value which is setfor all dimensions

Returns
the new *DataSpace*

Public Static Attributes

static constexpr uint32_t **`Dim`** = *T_dim*

40.4 Vector

template<typename **`T_Type`**, uint32_t **`T_dim`**, typename **`T_Navigator`** = IdentityNavigator, typename **`T_Storage`** = detail::Vector_components<*T_Type*, *T_dim*>>
struct **`Vector`** : private *pmacc*::math::detail::Vector_components<*T_Type*, *T_dim*>, protected *pmacc*::math::IdentityNavigator

Unnamed Group

`template<uint32_t T_numElements> inline constexpr HDINLINE Vector< type, T_numElements, Navigator > shrink ()`

Shrink the number of elements of a vector.

Template Parameters
`T_numElements` – New dimension of the vector.

Returns
First T_numElements elements of the origin vector

`template<uint32_t T_numElements> inline HDINLINE Vector< type, T_numElements, Navigator > shrink () const`

Public Types

using **`Storage`** = *T_Storage*

using **`type`** = typename *Storage*::type

using **`tag`** = *tag*::Vector

using **`Navigator`** = *T_Navigator*

using **`ParamType`** = typename boost::call_traits<*type*>::param_type

Public Functions

`PMACC_CASSERT_MSG (math_Vector__with_DIM_0_is_not_allowed, dim > 0u)`

constexpr **`Vector`**`()` = default

template<typename ...**`T_Args`**, typename = std::enable_if_t<(std::is_convertible_v<*T_Args*, *T_Type*> && ...)>>
inline constexpr HDINLINE **`Vector`**`(`*T_Args*... args`)`

Constructor for N-dimensional vector.

Attention

This constructor allows implicit casts.

Parameters

`args` – value of each dimension, x,y,z,. . .

constexpr HDINLINE **`Vector`**`(`const *Vector* &other`)` = default

template<typename **`T_OtherNavigator`**, typename **`T_OtherStorage`**>
inline HDINLINE **`Vector`**`(`const *Vector*<*T_Type*, *dim*, *T_OtherNavigator*, *T_OtherStorage*> &other`)`

template<typename **`T_OtherType`**, typename **`T_OtherNavigator`**, typename **`T_OtherStorage`**>
inline explicit HDINLINE **`Vector`**`(`const *Vector*<*T_OtherType*, *dim*, *T_OtherNavigator*, *T_OtherStorage*> &other`)`

template<typename **`T_MemberType`**>
inline explicit HDINLINE **`Vector`**`(`alpaka::Vec<::alpaka::DimInt<*T_dim*>, *T_MemberType*> const &value`)`

Transforms an alpaka vector into the corresponding PMacc vector.

The order of members is automatically permuted from z,y,x to x,y,z.

template<uint32_t **`T_deferDim`** = *T_dim*, typename = typename std::enable_if<*T_deferDim* == 1u>::type>
inline explicit HDINLINE **`operator type`**`()`

Allow static_cast / explicit cast to member type for 1D vector.

`inline HDINLINE const Vector & toRT () const`

`inline HDINLINE Vector & toRT ()`

`inline HDINLINE Vector revert ()`

`HDINLINE Vector & operator= (const Vector &)=default`

`template<typename T_OtherNavigator, typename T_OtherStorage> inline HDINLINE Vector & operator= (const Vector< type, dim, T_OtherNavigator, T_OtherStorage > &rhs)`

`inline HDINLINE type & operator[] (const uint32_t idx)`

`inline HDINLINE const type & operator[] (const uint32_t idx) const`

`inline HDINLINE type & x ()`

`inline HDINLINE type & y ()`

`inline HDINLINE type & z ()`

`inline HDINLINE const type & x () const`

`inline HDINLINE const type & y () const`

`inline HDINLINE const type & z () const`

`template<uint32_t T_numElements> inline HDINLINE Vector< type, T_numElements, Navigator > shrink (const int startIdx) const`

Shrink the number of elements of a vector.

Template Parameters

`T_numElements` – New dimension of the vector.

Parameters

`startIdx` – Index within the origin vector which will be the first element in the result.

Returns

T_numElements elements of the origin vector starting with the index startIdx. Indexing will wrapp around when the end of the origin vector is reached.

`template<uint32_t dimToRemove> inline HDINLINE Vector< type, dim - 1, Navigator > remove () const`

Removes a component.

It is not allowed to call this method on a vector with the dimensionality of one.

Template Parameters

`dimToRemove` – index which shall be removed; range: [0; dim - 1]

Returns

vector with `dim - 1` elements

`inline HDINLINE type productOfComponents () const`

Returns product of all components.

Returns
product of components

`inline HDINLINE type sumOfComponents () const`

Returns sum of all components.

Returns
sum of components

`template<typename T_OtherNavigator, typename T_OtherStorage> inline HDINLINE Vector & operator+= (const Vector< type, dim, T_OtherNavigator, T_OtherStorage > &other)`

+= operator

Parameters
`other` – instance with same type and dimension like the left instance

Returns
reference to manipulated left instance

`template<typename T_OtherNavigator, typename T_OtherStorage> inline HDINLINE Vector & operator-= (const Vector< type, dim, T_OtherNavigator, T_OtherStorage > &other)`

-= operator

Parameters
`other` – instance with same type and dimension like the left instance

Returns
reference to manipulated left instance

`template<typename T_OtherNavigator, typename T_OtherStorage> inline HDINLINE Vector & operator*= (const Vector< type, dim, T_OtherNavigator, T_OtherStorage > &other)`

*= operator

Parameters
`other` – instance with same type and dimension like the left instance

Returns
reference to manipulated left instance

`template<typename T_OtherNavigator, typename T_OtherStorage> inline HDINLINE Vector & operator/= (const Vector< type, dim, T_OtherNavigator, T_OtherStorage > &other)`

/= operator

Parameters
`other` – instance with same type and dimension like the left instance

Returns
reference to manipulated left instance

`inline HDINLINE Vector & operator+= (ParamType other)`

`inline HDINLINE Vector & operator-= (ParamType other)`

`inline HDINLINE Vector & operator*= (ParamType other)`

`inline HDINLINE Vector & operator/= (ParamType other)`

`inline HDINLINE bool operator== (const Vector &rhs) const`

== comparison operator.

Compares sizes of two DataSpaces.

Parameters
`other` – *Vector* to compare to

Returns
true if all components in both vectors are equal, else false

`inline HDINLINE bool operator!= (const Vector &rhs) const`

!= comparison operator.

Compares sizes of two DataSpaces.

Parameters
`other` – *Vector* to compare to

Returns
true if one component in both vectors are not equal, else false

inline std::string **`toString`**(const std::string separator = ",", const std::string enclosings = "{}") const

create string out of the vector

example: .toString(";","|") -> |x;...;z| .toString(",","[]") -> [x,...,z]

Parameters

- `separator` – string to separate components of the vector
- `enclosings` – string with size 2 to enclose vector size == 0 ? no enclose symbols size == 1 ? means enclose symbol begin and end are equal size >= 2 ? letter[0] = begin enclose symbol letter[1] = end enclose symbol

`inline HDINLINE alpaka::Vec<::alpaka::DimInt< T_dim >, MemIdxType > toAlpakaMemVec () const`

Transforms a *Vector* into alpaka vector used for memory extent descriptions.

The order of members is automatically permuted from x,y,z to z,y,x. The member data type will be MemIdxType.

Only integral types are supported. The method is performing an static cast to MemIdxType.

`inline HDINLINE alpaka::Vec<::alpaka::DimInt< T_dim >, IdxType > toAlpakaKernelVec () const`

Transforms a *Vector* into alpaka vector used for kernel extent descriptions.

The order of members is automatically permuted from x,y,z to z,y,x. The member data type will be IdxType to fit with the accelerator index type.

Only integral types are supported. The method is performing an static cast to IdxType.

Public Static Functions

static inline HDINLINE Vector create (ParamType value)

Creates a *Vector* where all dimensions are set to the same value.

Parameters

value – Value which is set for all dimensions

Returns

new Vector<…>

Public Static Attributes

static constexpr uint32_t **dim** = *Storage*::dim

40.5 SuperCell

template<typename **T_FrameType**, typename **T_SuperCellSize**>

class **SuperCell**

Public Types

using **SuperCellSize** = *T_SuperCellSize*

Public Functions

inline HDINLINE **SuperCell**()

inline HDINLINE T_FrameType * FirstFramePtr ()

inline HDINLINE T_FrameType * LastFramePtr ()

inline HDINLINE T_FrameType * FirstFramePtr () const

inline HDINLINE T_FrameType * LastFramePtr () const

inline HDINLINE bool mustShift () const

inline HDINLINE void setMustShift (bool const value)

inline HDINLINE uint32_t getSizeLastFrame () const

inline HDINLINE uint32_t getNumParticles () const

inline HDINLINE void setNumParticles (uint32_t const size)

PMACC_ALIGN(firstFramePtr, *T_FrameType**)

PMACC_ALIGN(lastFramePtr, *T_FrameType**)

40.6 GridBuffer

template<class **TYPE**, unsigned **DIM**, class **BORDERTYPE** = *TYPE*>

class **GridBuffer** : public *pmacc*::HostDeviceBuffer<*TYPE*, *DIM*>

GridBuffer represents a DIM-dimensional buffer which exists on the host as well as on the device.

GridBuffer combines a HostBuffer and a DeviceBuffer with equal sizes. Additionally, it allows sending data from and receiving data to these buffers. Buffers consist of core data which may be surrounded by border data.

Template Parameters

- **TYPE** – datatype for internal Host- and DeviceBuffer
- **DIM** – dimension of the buffers
- **BORDERTYPE** – optional type for border data in the buffers. TYPE is used by default.

Public Types

using **DataBoxType** = typename Parent::DataBoxType

Public Functions

inline **GridBuffer**(const GridLayout<*DIM*> &gridLayout, bool sizeOnDevice = false)

Constructor.

Parameters

- **gridLayout** – layout of the buffers, including border-cells
- **sizeOnDevice** – if true, size information exists on device, too.

inline **GridBuffer**(const *DataSpace*<*DIM*> &dataSpace, bool sizeOnDevice = false)

Constructor.

Parameters

- **dataSpace** – *DataSpace* representing buffer size without border-cells
- **sizeOnDevice** – if true, internal buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)

inline **GridBuffer**(DeviceBuffer<*TYPE*, *DIM*> &otherDeviceBuffer, const GridLayout<*DIM*> &gridLayout, bool sizeOnDevice = false)

Constructor.

Parameters

- **otherDeviceBuffer** – DeviceBuffer which should be used instead of creating own DeviceBuffer
- **gridLayout** – layout of the buffers, including border-cells

- **sizeOnDevice** – if true, internal buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)

inline **GridBuffer**(HostBuffer<*TYPE*, *DIM*> &otherHostBuffer, const *DataSpace*<*DIM*> &offsetHost, DeviceBuffer<*TYPE*, *DIM*> &otherDeviceBuffer, const *DataSpace*<*DIM*> &offsetDevice, const GridLayout<*DIM*> &gridLayout, bool sizeOnDevice = false)

inline void **addExchange**(uint32_t dataPlace, const Mask &receive, *DataSpace*<*DIM*> guardingCells, uint32_t communicationTag, bool sizeOnDeviceSend, bool sizeOnDeviceReceive)

Add Exchange in *GridBuffer* memory space.

An Exchange is added to this *GridBuffer*. The exchange buffers use the same memory as this *GridBuffer*.

Parameters

- **dataPlace** – place where received data is stored [GUARD | BORDER] if dataPlace=GUARD than copy other BORDER to my GUARD if dataPlace=BORDER than copy other GUARD to my BORDER
- **receive** – a Mask which describes the directions for the exchange
- **guardingCells** – number of guarding cells in each dimension
- **communicationTag** – unique tag/id for communication has to be the same when this method is called multiple times for the same object (with non-overlapping masks)
- **sizeOnDeviceSend** – if true, internal send buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)
- **sizeOnDeviceReceive** – if true, internal receive buffers must store their size additionally on the device

inline void **addExchange**(uint32_t dataPlace, const Mask &receive, *DataSpace*<*DIM*> guardingCells, uint32_t communicationTag, bool sizeOnDevice = false)

Add Exchange in *GridBuffer* memory space.

An Exchange is added to this *GridBuffer*. The exchange buffers use the same memory as this *GridBuffer*.

Parameters

- **dataPlace** – place where received data is stored [GUARD | BORDER] if dataPlace=GUARD than copy other BORDER to my GUARD if dataPlace=BORDER than copy other GUARD to my BORDER
- **receive** – a Mask which describes the directions for the exchange
- **guardingCells** – number of guarding cells in each dimension
- **communicationTag** – unique tag/id for communication
- **sizeOnDevice** – if true, internal buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)

inline void **addExchangeBuffer**(const Mask &receive, const *DataSpace*<*DIM*> &dataSpace, uint32_t communicationTag, bool sizeOnDeviceSend, bool sizeOnDeviceReceive)

Add Exchange in dedicated memory space.

An Exchange is added to this *GridBuffer*. The exchange buffers use the their own memory instead of using the *GridBuffer*'s memory space.

Parameters

- `receive` – a Mask which describes the directions for the exchange
- `dataSpace` – size of the newly created exchange buffer in each dimension
- `communicationTag` – unique tag/id for communication
- `sizeOnDeviceSend` – if true, internal send buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)
- `sizeOnDeviceReceive` – if true, internal receive buffers must store their size additionally on the device

inline void **`addExchangeBuffer`**(const Mask &receive, const *DataSpace*<*DIM*> &dataSpace, uint32_t communicationTag, bool sizeOnDevice = false)

Add Exchange in dedicated memory space.

An Exchange is added to this *GridBuffer*. The exchange buffers use the their own memory instead of using the *GridBuffer*'s memory space.

Parameters

- `receive` – a Mask which describes the directions for the exchange
- `dataSpace` – size of the newly created exchange buffer in each dimension
- `communicationTag` – unique tag/id for communication
- `sizeOnDevice` – if true, internal buffers must store their size additionally on the device (as we keep this information coherent with the host, it influences performance on host-device copies, but some algorithms on the device might need to know the size of the buffer)

inline bool **`hasSendExchange`**(uint32_t ex) const

Returns whether this *GridBuffer* has an Exchange for sending in ex direction.

Parameters

`ex` – exchange direction to query

Returns

true if send exchanges with ex direction exist, otherwise false

inline bool **`hasReceiveExchange`**(uint32_t ex) const

Returns whether this *GridBuffer* has an Exchange for receiving from ex direction.

Parameters

`ex` – exchange direction to query

Returns

true if receive exchanges with ex direction exist, otherwise false

inline Exchange<*BORDERTYPE*, *DIM*> &**`getSendExchange`**(uint32_t ex) const

Returns the Exchange for sending data in ex direction.

Returns an Exchange which for sending data from this *GridBuffer* in the direction described by ex.

Parameters

`ex` – the direction to query

Returns
the Exchange for sending data

inline Exchange<*BORDERTYPE*, *DIM*> &**getReceiveExchange**(uint32_t ex) const

Returns the Exchange for receiving data from ex direction.

Returns an Exchange which for receiving data to this *GridBuffer* from the direction described by ex.

Parameters
ex – the direction to query

Returns
the Exchange for receiving data

inline Mask **getSendMask**() const

Returns the Mask describing send exchanges.

Returns
Mask for send exchanges

inline Mask **getReceiveMask**() const

Returns the Mask describing receive exchanges.

Returns
Mask for receive exchanges

inline EventTask **communication**()

Starts sync data from own device buffer to neighbor device buffer.

Asynchronously starts synchronization data from internal DeviceBuffer using added Exchange buffers. This operation runs sequential to other code but intern asynchronous

inline EventTask **asyncCommunication**(EventTask serialEvent)

Starts sync data from own device buffer to neighbor device buffer.

Asynchronously starts synchronization data from internal DeviceBuffer using added Exchange buffers.

inline EventTask **asyncSend**(EventTask serialEvent, uint32_t sendEx)

inline EventTask **asyncReceive**(EventTask serialEvent, uint32_t recvEx)

inline GridLayout<*DIM*> **getGridLayout**()

Returns the GridLayout describing this *GridBuffer*.

Returns
the layout of this buffer

Protected Attributes

bool **hasOneExchange**

uint32_t **lastUsedCommunicationTag**

GridLayout<*DIM*> **gridLayout**

Mask **sendMask**

Mask **receiveMask**

std::unique_ptr<Exchange<*BORDERTYPE*, *DIM*>> **sendExchanges**[27]

std::unique_ptr<Exchange<*BORDERTYPE*, *DIM*>> **receiveExchanges**[27]

EventTask **receiveEvents**[27]

EventTask **sendEvents**[27]

uint32_t **maxExchange**

40.7 SimulationFieldHelper

template<class **CellDescription**>

class **SimulationFieldHelper**

Public Types

using **MappingDesc** = *CellDescription*

Public Functions

inline **SimulationFieldHelper**(*CellDescription* description)

virtual **~SimulationFieldHelper**() = default

virtual void **reset**(uint32_t currentStep) = 0

Reset is as well used for init.

virtual void **syncToDevice**() = 0

Synchronize data from host to device.

inline *CellDescription* **getCellDescription**() const

Public Static Attributes

static constexpr uint32_t **dim** = *MappingDesc*::Dim

Protected Attributes

CellDescription **cellDescription**

40.8 ParticlesBase

template<typename **T_ParticleDescription**, class **T_MappingDesc**, typename **T_DeviceHeap**>

class **ParticlesBase** : public *pmacc*::*SimulationFieldHelper*<*T_MappingDesc*>

Public Types

using **MappingDesc** = *T_MappingDesc*

using **BufferType** = ParticlesBuffer<*ParticleDescription*, typename *MappingDesc*::SuperCellSize, *T_DeviceHeap*, *MappingDesc*::Dim>

using **FrameType** = typename *BufferType*::FrameType

using **FrameTypeBorder** = typename *BufferType*::FrameTypeBorder

using **ParticlesBoxType** = typename *BufferType*::ParticlesBoxType

using **HandleGuardRegion** = typename *ParticleDescription*::*HandleGuardRegion*

using **SimulationDataTag** = ParticlesTag

Public Functions

template<typename **T_MapperFactory**>
inline void **fillGaps**(*T_MapperFactory* const &mapperFactory)

Fill gaps in an area defined by a mapper factory.

Template Parameters

T_MapperFactory – factory type to construct a mapper that defines the area to process

Parameters

mapperFactory – factory instance

inline void **fillAllGaps**()

inline void **fillBorderGaps**()

void **deleteGuardParticles**(uint32_t exchangeType)

template<uint32_t **T_area**>
void **deleteParticlesInArea**()

void **copyGuardToExchange**(uint32_t exchangeType)

copy guard particles to intermediate exchange buffer

Copy all particles from the guard of a direction to the device exchange buffer.

Warning: This method resets the number of particles in the processed supercells even if there are particles left in the supercell and does not guarantee that the last frame is contiguous filled. Call fillAllGaps afterwards if you need a valid number of particles and a contiguously filled last frame.

void **insertParticles**(uint32_t exchangeType)

inline *ParticlesBoxType* **getDeviceParticlesBox**()

inline *ParticlesBoxType* **getHostParticlesBox**(const int64_t memoryOffset)

inline *BufferType* &**getParticlesBuffer**()

virtual void **reset**(uint32_t currentStep) override

Reset is as well used for init.

Public Static Attributes

static constexpr uint32_t **dim** = *MappingDesc*::Dim

Protected Functions

inline **ParticlesBase**(const std::shared_ptr<*T_DeviceHeap*> &deviceHeap, *MappingDesc* description)

inline ~**ParticlesBase**() override

template<uint32_t **T_area**>
inline void **shiftParticles**(bool onlyProcessMustShiftSupercells)

Shift all particles in an area defined by a mapper factory.

The factory type must be such that StrideMapperFactory<T_MapperFactory, stride> is specialized

Parameters

onlyProcessMustShiftSupercells – whether to process only supercells with mustShift set to true (optimization to be used with particle pusher) or process all supercells

template<typename **T_MapperFactory**>
inline void **shiftParticles**(*T_MapperFactory* const &mapperFactory, bool onlyProcessMustShiftSupercells)

Shift all particles in the area defined by the given factory.

Note that the area itself is not strided, but the factory must produce stride mappers for the area.

Template Parameters

T_strideMapperFactory – factory type to construct a stride mapper, resulting mapper must have stride of at least 3, adheres to the MapperFactory concept

Parameters

- **mapperFactory** – factory instance
- **onlyProcessMustShiftSupercells** – whether to process only supercells with mustShift set to true (optimization to be used with particle pusher) or process all supercells

Protected Attributes

BufferType ***particlesBuffer**

40.9 ParticleDescription

template<typename **T_Name**, typename **T_NumSlots**, typename **T_SuperCellSize**, typename **T_ValueTypeSeq**, typename **T_Flags** = mp_list<>, typename **T_HandleGuardRegion** = *HandleGuardRegion*<particles::policies::ExchangeParticles, particles::policies::DeleteParticles>, typename **T_MethodsList** = mp_list<>, typename **T_FrameExtensionList** = mp_list<>>
struct **ParticleDescription**

ParticleDescription defines attributes, methods and flags of a particle.

This class holds no runtime data. The class holds information about the name, attributes, flags and methods of a particle.

Template Parameters

- **T_Name** – name of described particle (e.g. electron, ion) type must be a PMACC_CSTRING
- **T_NumSlots** – compile time size of a super cell
- **T_ValueTypeSeq** – sequence or single type with value_identifier, must not have duplicates
- **T_Flags** – sequence or single type with identifier to add flags on a frame, must not have duplicates
- **T_MethodsList** – sequence or single class with particle methods (e.g. calculate mass, gamma, . . .) (e.g. useSolverXY, calcRadiation, . . .)
- **T_FrameExtensionList** – sequence or single class with frame extensions
  - extension must be an unary template class that supports boost::mpl::apply1<>
  - type of the final frame is applied to each extension class (this allows pointers and references to a frame itself)
  - the final frame that uses *ParticleDescription* inherits from all extension classes

Public Types

using **Name** = *T_Name*

using **NumSlots** = *T_NumSlots*

using **SuperCellSize** = *T_SuperCellSize*

using **ValueTypeSeq** = ToSeq<*T_ValueTypeSeq*>

using **FlagsList** = ToSeq<*T_Flags*>

using **HandleGuardRegion** = *T_HandleGuardRegion*

using **MethodsList** = ToSeq<*T_MethodsList*>

using **FrameExtensionList** = ToSeq<*T_FrameExtensionList*>

Public Functions

PMACC_CASSERT_MSG(_error_particles_must_not_have_duplicate_attributes____check_your_speciesDefinition_param_file, isUnique<*ValueTypeSeq*>)

PMACC_CASSERT_MSG(_error_particles_must_not_have_duplicate_flags____check_your_speciesDefinition_param_file, isUnique<*FlagsList*>)

40.10 ParticleBox

template<class **T_Frame**, typename **T_DeviceHeapHandle**, typename **T_SuperCellSize**, unsigned **DIM**>

class **ParticlesBox** : protected *pmacc*::DataBox<PitchedBox<*SuperCell*<*T_Frame*, *T_SuperCellSize*>, *DIM*>>

A DIM-dimensional Box holding frames with particle data.

Template Parameters

- **FRAME** – datatype for frames
- **DIM** – dimension of data (1-3)

Public Types

using **FrameType** = *T_Frame*

using **FramePtr** = FramePointer<*FrameType*>

using **SuperCellType** = *SuperCell*<*FrameType*, *T_SuperCellSize*>

using **BaseType** = DataBox<PitchedBox<*SuperCell*<*FrameType*, *T_SuperCellSize*>, *DIM*>>

using **SuperCellSize** = *T_SuperCellSize*

using **DeviceHeapHandle** = *T_DeviceHeapHandle*

Public Functions

HDINLINE **ParticlesBox**() = default

default constructor

Warning: after this call the object is in a invalid state and must be initialized with an assignment of a valid ParticleBox

inline HDINLINE **ParticlesBox**(const DataBox<PitchedBox<*SuperCellType*, *DIM*>> &superCells, const *DeviceHeapHandle* &deviceHeapHandle)

inline HDINLINE **ParticlesBox**(const DataBox<PitchedBox<*SuperCellType*, *DIM*>> &superCells, const *DeviceHeapHandle* &deviceHeapHandle, int64_t memoryOffset)

`template<typename T_Worker> inline DINLINE FramePtr getEmptyFrame (const T_Worker &worker)`

Returns an empty frame from data heap.

Returns
an empty frame

`template<typename T_Worker> inline DINLINE void removeFrame (const T_Worker &worker, FramePtr &frame)`

Removes frame from heap data heap.

Parameters
`frame` – frame to remove

`inline HDINLINE FramePtr mapPtr (FramePtr devPtr) const`

`inline HDINLINE FramePtr getNextFrame (const FramePtr &frame) const`

Returns the next frame in the linked list.

Parameters
`frame` – the active frame

Returns
the next frame in the list

`inline HDINLINE FramePtr getPreviousFrame (const FramePtr &frame) const`

Returns the previous frame in the linked list.

Parameters
`frame` – the active frame

Returns
the previous frame in the list

`inline HDINLINE FramePtr getLastFrame (const DataSpace< DIM > &idx) const`

Returns the last frame of a supercell.

Parameters
`idx` – position of supercell

Returns
the last frame of the linked list from supercell

`inline HDINLINE FramePtr getFirstFrame (const DataSpace< DIM > &idx) const`

Returns the first frame of a supercell.

Parameters
`idx` – position of supercell

Returns
the first frame of the linked list from supercell

`template<typename T_Worker> inline DINLINE void setAsFirstFrame (T_Worker const &worker, FramePtr &frame, DataSpace< DIM > const &idx)`

Sets frame as the first frame of a supercell.

Parameters

- `frame` – frame to set as first frame
- `idx` – position of supercell

`template<typename T_Worker> inline DINLINE void setAsLastFrame (T_Worker const &worker, FramePointer< FrameType > &frame, DataSpace< DIM > const &idx)`

Sets frame as the last frame of a supercell.

Parameters

- `frame` – frame to set as last frame
- `idx` – position of supercell

`template<typename T_Worker> inline DINLINE bool removeLastFrame (const T_Worker &worker, const DataSpace< DIM > &idx)`

Removes the last frame of a supercell.

This call is not threadsave, only one thread from a supercell may call this function.

Parameters
`idx` – position of supercell

Returns
true if more frames in list, else false

`inline decltype(auto) HDINLINE getSuperCell (DataSpace< DIM > idx) const`

`inline decltype(auto) HDINLINE getSuperCell (DataSpace< DIM > idx)`

Public Static Attributes

static constexpr uint32_t **`frameSize`** = *FrameType*::NumSlots::value

static constexpr uint32_t **`Dim`** = *DIM*

40.11 Frame

template<typename **`T_CreatePairOperator`**, typename **`T_ParticleDescription`**>

struct **`Frame`** : protected *pmacc*::math::MapTuple<SeqToMap<*T_ParticleDescription*::ValueTypeSeq, *T_CreatePairOperator*>::type>, public *pmacc*::InheritLinearly<mp_append<*T_ParticleDescription*::MethodsList, OperateOnSeq<*T_ParticleDescription*::FrameExtensionList, boost::mpl::apply1<boost::mpl::_1, *Frame*<*T_CreatePairOperator*, *T_ParticleDescription*>>>::type>>

Frame is a storage for arbitrary number >0 of Particles with attributes.

See also:

MapTupel

Template Parameters

- `T_CreatePairOperator` – unary template operator to create a boost pair from single type (pair<name,dataType>)

- **T_ValueTypeSeq** – sequence with value_identifier
- **T_MethodsList** – sequence of classes with particle methods (e.g. calculate mass, gamma, . . .)
- **T_Flags** – sequence with identifiers to add flags on a frame (e.g. useSolverXY, calcRadiation, . . .)

Public Types

using **ParticleDescription** = *T_ParticleDescription*

using **Name** = typename *ParticleDescription*::Name

using **NumSlots** = typename *ParticleDescription*::NumSlots

Number of particle slots within the frame.

using **ValueTypeSeq** = typename *ParticleDescription*::ValueTypeSeq

using **MethodsList** = typename *ParticleDescription*::MethodsList

using **FlagList** = typename *ParticleDescription*::FlagsList

using **FrameExtensionList** = typename *ParticleDescription*::FrameExtensionList

using **BaseType** = pmath::MapTuple<typename SeqToMap<*ValueTypeSeq*, *T_CreatePairOperator*>::type>

using **ParticleType** = *pmacc*::Particle<*Frame*>

using **SuperCellSize** = typename *ParticleDescription*::SuperCellSize

Public Functions

inline HDINLINE ParticleType operator[] (const uint32_t idx)

access the Nth particle

inline HDINLINE const ParticleType operator[] (const uint32_t idx) const

access the Nth particle

template<typename **T_Key**>
inline HDINLINE auto &**getIdentifier**(const *T_Key*)

access attribute with a identifier

Parameters

T_Key – instance of identifier type (can be an alias, value_identifier or any other class)

Returns

result of operator[] of MapTupel

template<typename T_Key> inline HDINLINE const auto & getIdentifier (const T_Key) const

const version of method *getIdentifier(const T_Key)*

Public Static Functions

static inline HINLINE std::string getName ()

Public Static Attributes

static constexpr uint32_t **frameSize** = *NumSlots*::value

40.12 IPlugin

class **IPlugin** : public *pmacc*::INotify

Subclassed by *pmacc::SimulationHelper< simDim >*, picongpu::ISimulationPlugin, picongpu::ISimulationStarter, picongpu::MetadataAggregator, picongpu::plugins::binning::Binner< TBinningData >, *pmacc::SimulationHelper< DIM >*

Public Functions

IPlugin() = default

~IPlugin() override = default

inline virtual void **load**()

inline virtual void **unload**()

inline bool **isLoaded**()

virtual void **checkpoint**(uint32_t currentStep, const std::string checkpointDirectory) = 0

Notifies plugins that a (restartable) checkpoint should be created for this timestep.

Parameters

- **currentStep** – cuurent simulation iteration step
- **checkpointDirectory** – common directory for checkpoints

virtual void **restart**(uint32_t restartStep, const std::string restartDirectory) = 0

Restart notification callback.

Parameters

- **restartStep** – simulation iteration step to restart from
- **restartDirectory** – common restart directory (contains checkpoints)

virtual void **pluginRegisterHelp**(po::options_description &desc) = 0

Register command line parameters for this plugin.

Parameters are parsed and set prior to plugin load.

Parameters

desc – boost::program_options description virtual std::string **pluginGetName**() const = 0

Return the name of this plugin for status messages.

Returns
plugin name

inline virtual void **onParticleLeave**(const std::string&, const int32_t)

Called each timestep if particles are leaving the global simulation volume.

This method is only called for species which are marked with the `GuardHandlerCallPlugins` policy in their descpription.

The order in which the plugins are called is undefined, so this means read-only access to the particles.

Parameters

- **speciesName** – name of the particle species
- **direction** – the direction the particles are leaving the simulation

inline uint32_t **getLastCheckpoint**() const

When was the plugin checkpointed last?

Returns
last checkpoint's time step

inline void **setLastCheckpoint**(uint32_t currentStep)

Remember last checkpoint call.

Parameters
currentStep – current simulation iteration step

Protected Functions

inline virtual void **pluginLoad**()

inline virtual void **pluginUnload**()

Protected Attributes

bool **loaded** = {false}

uint32_t **lastCheckpoint** = {0}

40.13 PluginConnector

class **PluginConnector**

Plugin registration and management class.

Public Functions

void **registerPlugin**(*IPlugin* *plugin)

Register a plugin for loading/unloading and notifications.

Plugins are loaded in the order they are registered and unloaded in reverse order. To trigger plugin notifications, call

See also:

setNotificationPeriod after registration.

Parameters

plugin – plugin to register

void **loadPlugins**()

Calls load on all registered, not loaded plugins.

void **unloadPlugins**()

Unloads all registered, loaded plugins.

std::list<po::options_description> **registerHelp**()

Publishes command line parameters for registered plugins.

Returns

list of boost program_options command line parameters

void **setNotificationPeriod**(INotify *notifiedObj, std::string const &period)

Set the notification period.

Parameters

- **notifiedObj** – the object to notify, e.g. an *IPlugin* instance
- **period** – notification period

void **notifyPlugins**(uint32_t currentStep)

Notifies plugins that data should be dumped.

Parameters

currentStep – current simulation iteration step

void **checkpointPlugins**(uint32_t currentStep, const std::string checkpointDirectory)

Notifies plugins that a restartable checkpoint should be dumped.

Parameters

- **currentStep** – current simulation iteration step
- **checkpointDirectory** – common directory for checkpoints

void **restartPlugins**(uint32_t restartStep, const std::string restartDirectory)

Notifies plugins that a restart is required.

Parameters

- **restartStep** – simulation iteration to restart from
- **restartDirectory** – common restart directory (contains checkpoints)

template<typename **Plugin**>
inline std::vector<*Plugin**> **getPluginsFromType**()

Get a vector of pointers of all registered plugin instances of a given type.

Template Parameters

Plugin – type of plugin

Returns
vector of plugin pointers

std::list<*IPlugin**> **getAllPlugins**() const

Return a copied list of pointers to all registered plugins.

Friends

friend struct detail::Environment

40.14 SimulationHelper

template<unsigned **DIM**>

class **SimulationHelper** : public *pmacc*::*IPlugin*

Abstract base class for simulations.

Use this helper class to write your own concrete simulations by binding pure virtual methods.

Template Parameters
DIM – base dimension for the simulation (2-3)

Public Types

using **SeqOfTimeSlices** = std::vector<pluginSystem::Slice>

Public Functions

SimulationHelper()

Constructor.

~SimulationHelper() override

virtual void **runOneStep**(uint32_t currentStep) = 0

Must describe one iteration (step).

This function is called automatically.

virtual void **init**() = 0

Initialize simulation.

Does hardware selections/reservations, memory allocations and initializes data structures as empty.

virtual uint32_t **fillSimulation**() = 0

Fills simulation with initial data after *init()*

Returns
returns the first step of the simulation (can be >0 for, e.g., restarts from checkpoints)

virtual void **resetAll**(uint32_t currentStep) = 0

Reset the simulation to a state such as it was after *init()* but for a specific time step.

Can be used to call *fillSimulation()* again.

virtual void **movingWindowCheck**(uint32_t currentStep) = 0

Check if moving window work must do.

If no moving window is needed the implementation of this function can be empty

Parameters
`currentStep` – simulation step

void **notifyPlugins**(uint32_t currentStep)

Call all plugins.

This function is called inside the simulation loop.

Parameters
`currentStep` – simulation step

virtual void **dumpOneStep**(uint32_t currentStep)

Write a checkpoint if needed for the given step.

This function is called inside the simulation loop.

Parameters
`currentStep` – simulation step

inline GridController<*DIM*> &**getGridController**()

void **dumpTimes**(TimeIntervall &tSimCalculation, TimeIntervall&, double &roundAvg, uint32_t currentStep)

virtual void **startSimulation**()

Begin the simulation.

virtual void **pluginRegisterHelp**(po::options_description &desc) override

Register command line parameters for this plugin.

Parameters are parsed and set prior to plugin load.

Parameters
`desc` – boost::program_options description

inline virtual std::string **pluginGetName**() const override

Return the name of this plugin for status messages.

Returns
plugin name

virtual void **pluginLoad**() override

inline virtual void **pluginUnload**() override

inline virtual void **restart**(uint32_t, const std::string) override

Restart notification callback.

Parameters

- `restartStep` – simulation iteration step to restart from
- `restartDirectory` – common restart directory (contains checkpoints)

inline virtual void **checkpoint**(uint32_t, const std::string) override

Notifies plugins that a (restartable) checkpoint should be created for this timestep.

Parameters

- `currentStep` – cuurent simulation iteration step
- `checkpointDirectory` – common directory for checkpoints

Protected Functions

std::vector<uint32_t> **readCheckpointMasterFile**()

Reads the checkpoint master file if any and returns all found checkpoint steps.

Returns

vector of found checkpoints steps in order they appear in the file

Protected Attributes

uint32_t **runSteps** = {0}

uint32_t **softRestarts**

Presentations: loop the whole simulation `softRestarts` times from initial step to runSteps.

std::string **checkpointPeriod**

SeqOfTimeSlices **seqCheckpointPeriod**

std::uint64_t **checkpointPeriodMinutes** = 0u

std::thread **checkpointTimeThread**

std::condition_variable **exitConcurrentThreads**

std::mutex **concurrentThreadMutex**

std::string **checkpointDirectory**

uint32_t **numCheckpoints** = {0}

int32_t **restartStep** = {-1}

std::string **restartDirectory**

bool **restartRequested** = {false}

const std::string **CHECKPOINT_MASTER_FILE**

std::string **author**

bool **useMpiDirect** = {false}

enable MPI gpu direct

bool **tryRestart** = false

40.15 ForEach

template<typename **List**, typename **T_Functor**, typename **T_Accessor** = meta::accessors::Identity<>>

struct **ForEach**

Compile-Time for each for Boost::MPL Type Lists.

Example: List = pmacc::mp_list<int,float> Functor = any unary lambda functor Accessor = lambda operation identity

definition: F(X) means boost::apply<F,X>

call: ForEach<List,Functor,Accessor>()(42); unrolled code: Functor(Accessor(int))(42); Functor(Accessor(float))(42);

Template Parameters

- **List** – An mp_list.
- **T_Functor** – An unary lambda functor with a HDINLINE void operator()(…) method _1 is substituted by Accessor's result using boost::mpl::apply with elements from T_MPLSeq. The maximum number of parameters for the operator() is limited by PMACC_MAX_FUNCTOR_OPERATOR_PARAMS
- **T_Accessor** – An unary lambda operation

Public Types

template<typename **X**>

using **ReplacePlaceholder** = typename boost::mpl::apply1<*T_Functor*, typename boost::mpl::apply1<*T_Accessor*, *X*>::type>::type

using **SolvedFunctors** = mp_transform<*ReplacePlaceholder*, *List*>

Public Functions

template<typename... T_Types> inline HDINLINE void operator() (T_Types &&... ts) const

40.16 Kernel Start

Warning: doxygenstruct: Cannot find class "pmacc::exec::Kernel" in doxygen xml output for project "PIConGPU" from directory: ../xml

PMACC_KERNEL(...)

Create a kernel object out of a functor instance.

This macro add the current filename and line number to the kernel object.

See also:

::pmacc::exec::kernel

Parameters

- **...** – instance of kernel functor

40.17 Identifier

Construct unique types, e.g. to name, access and assign default values to particle species' attributes. See for example PIConGPU's speciesAttributes.param for usage.

`value_identifier`(in_type, name, in_default)

define a unique identifier with name, type and a default value

The created identifier has the following options: getValue() - return the user defined value getName() - return the name of the identifier ::type - get type of the value

e.g. value_identifier(float,length,0.0f) typedef length::type value_type; // is float value_type x = length::getValue(); //set x to 0.f printf("Identifier name: %s",length::getName()); //print Identifier name: length

to create a instance of this value_identifier you can use: `length()` or `length_`

Parameters

- **in_type** – type of the value
- **name** – name of identifier
- **in_value** – user defined value of in_type (can be a constructor of a class)

`alias`(name)

create an alias

an alias is a unspecialized type of an identifier or a value_identifier

example: alias(aliasName); //create type varname

to specialize an alias do: aliasName<valueIdentifierName> to create an instance of this alias you can use: aliasName(); or aliasName_

get type which is represented by the alias typedef typename traits::Resolve<name>::type resolved_type;

Parameters

- **name** – name of alias

CHAPTER

FORTYONE

PYTHON POSTPROCESSING TOOL STRUCTURE

Each plugin should implement at least the following Python classes.

1. A data reader class responsible for loading the data from the simulation directory
2. A visualizer class that outputs a matplotlib plot
3. A jupyter-widget class that exposes the parameters of the matplotlib visualizer to the user via other widgets.

The repository directory for PIConGPU Python modules for plugins is `lib/python/picongpu/extra/plugins/`.

Note: The plugins have been moved to the *picongpu.extra* submodule.

41.1 Data Reader

The data readers should reside in the `lib/python/picongpu/extra/plugins/data` directory. There is a base class in `base_reader.py` defining the interface of a reader. Each reader class should derive from this class and implement the interface functions not implemented in this class.

To shorten the import statements for the readers, please also add an entry in the `__init__.py` file of the `data` directory.

41.2 Matplotlib visualizer

The visualizers should reside in the `lib/python/picongpu/extra/plugins/plot_mpl/` directory. The module names should end on `_visualizer.py` and the class name should only be `Visualizer`.

To shorten the import statements for the visualizers, please also add an entry in the `__init__.py` file of the `plot_mpl` directory with an alias that ends on "MPL".

There is a base class for visualization found in `base_visualizer.py` which already handles the plotting logic. It uses (possibly multiple) instances of the data reader classes for accessing the data. Visualizing data simultaneously for more than one scan is supported by creating as many readers and plot objects as there are simulations for visualization. After getting the data, it ensures that (for performance reasons) a matplotlib artist is created only for the first plot and later only gets updated with fresh data.

All new plugins should derive from this class.

When implementing a new visualizer you have to perform the following steps:

1. Let your visualizer class inherit from the `Visualizer` class in `base visualizer.py` and call the base class constructor with the correct data reader class.

2. Implement the `_create_plt_obj(self, idx)` function. This function needs to access the plotting data from the `self.data` member (this is the data structure as returned by the data readers `.get(...)` function, create some

kind of matplotlib artist by storing it in the `self.plt_obj` member variable at the correct index specified by the idx variable (which corresponds to the data of the simulation at position idx that is passed in construction.

3. Implement the `_update_plt_obj(self, idx)` function. This is called only after a valid `self.plt_obj` was created. It updates the matplotlib artist with new data. Therefore it again needs to access the plotting data from the `self.data` member and call the data update API for the matplotlib artist (normally via `.set_data(...)`.

41.3 Jupyter Widget

The widget is essentially only a wrapper around the matplotlib visualizer that allows dynamical adjustment of the parameters the visualizer accepts for plotting. This allows to adjust e.g. species, filter and other plugin-dependent options without having to write new lines of Python code.

The widgets should reside in the `lib/python/picongpu/extra/plugins/jupyter_widgets/` directory. The module names should end on `_widget.py`.

To shorten the import statements for the widgets, please also add an entry in the `__init__.py` file of the `jupyter_widget` directory.

There is a base class for visualization found in `base_widget.py` which already handles most of the widget logic.

It allows to switch between visualizations for different simulation times (iterations) and different simulations.

When implementing a new widget you have to perform the following steps:

1. Let the widget class inherit from the `BaseWidget` class in `base_widget.py` and call the base class constructor with the correct matplotlib visualizer class.

```
from .base_widget import BaseWidget

class NewPluginWidget(BaseWidget):
```

2. In the constructor, call the base class constructor with the matplotlib visualizer class as `plot_mpl_cls` keyword.

   The base class will then create an instance of the visualizer class and delegate the plotting job to it.

```
# taken from lib/python/picongpu/plugins/jupyter_widgets/energy_histogram_widget.py
from .base_widget import BaseWidget
from picongpu.plugins.plot_mpl import EnergyHistogramMPL

class EnergyHistogramWidget(BaseWidget):
    def __init__(self, run_dir_options, fig=None, **kwargs):

        BaseWidget.__init__(self,
                            EnergyHistogramMPL,
                            run_dir_options,
                            fig,
                            **kwargs)
```

3. implement the `_create_widgets_for_vis_args(self)` function.

   This function has to define jupyter widgets as member variables of the class to allow interactive manipulation of parameters the underlying matplotlib visualizer is capable of handling. It needs to return a dictionary using the parameter names the matplotlib visualizer accepts as keys and the widget members that correspond to these parameters as values.

```
# taken from lib/python/picongpu/plugins/jupyter_widgets/energy_histogram_widget.py
 def _create_widgets_for_vis_args(self):
     # widgets for the input parameters
     self.species = widgets.Dropdown(description="Species",
```

(continues on next page)

(continued from previous page)

```
                                options=["e"],
                                value='e')
    self.species_filter = widgets.Dropdown(description="Species_filter",
                                       options=['all'],
                                       value="all")

    return {'species': self.species,
            'species_filter': self.species_filter}
```

CHAPTER

FORTYTWO

DEBUGGING

Section author: Sergei Bastrakov

When investigating the reason of a crashing simulation, it is very helpful to reduce the setup as much as possible while the crash is still happening. This includes moving to fewer (and ideally to 1) MPI processes, having minimal grid size (ideally 3 supercells), disabling unrelated plugins and simulation stages. Such a reduction should be done incrementally while checking whether the issue is still observed. Knowing when the issue disappears could hint to its source. This process also significantly speeds up and helps to focus a following deeper look.

The following build options can assist the investigation:

- `PIC_VERBOSE=<N>` sets log detail level for PIConGPU, highest level is 127.
- `PMACC_VERBOSE=<N>` sets log detail level for pmacc, highest level is 127.
- `PMACC_BLOCKING_KERNEL=ON` makes each kernel invocation blocking, which helps to narrow a crash down to a particular kernel.
- `PMACC_ASYNC_QUEUES=OFF` disables asynchronous alpaka queues, also helps to narrow a crash down to a particular place.
- `CMAKE_BUILD_TYPE=Debug` compile in debug mode where all assertions are activated. Compiling in debug mode will slowdown your kernels!

These options can be passed when building, or manually modified via cmake. An example build command is

```
pic-build -c "-DPIC_VERBOSE=127 -DPMACC_VERBOSE=127 -DPMACC_BLOCKING_KERNEL=ON -
↪DPMACC_USE_ASYNC_QUEUES=OFF"
```

When reporting a crash, it is helpful if you attached the output `stdout` and `stderr` of such a build.

For further debugging tips and use of tools please refer to our Wiki.

CHAPTER

FORTYTHREE

INDEX OF DOXYGEN DOCUMENTATION

This command is currently taking up to 2 GB of RAM, so we can't run it on read-the-docs:

doxygenindex::

> **project**
> PIConGPU
>
> **path**
> '../xml'
>
> **outline**
> **no-link**

See also:

In order to follow this section, you need to understand the CUDA programming model.

CHAPTER

FORTYFOUR

LOCKSTEP PROGRAMMING MODEL

Section author: René Widera, Axel Huebl

The *lockstep programming model* structures code that is evaluated collectively and independently by workers (physical threads) within a alpaka block. Actual processing is described by one-dimensional index domains which are known compile time and can even be changed within a kernel.

An index domain is **independent** of the data but **can** be mapped to a data domain, e.g. one to one or with more complex mappings. A index domain is processed collectively by all workers.

Code which is implemented by the *lockstep programming model* is free of any dependencies between the number of worker and processed data elements. To simplify the implementation, each index within a domain can be mapped to a single data element (like the common workflow to programming CUDA). But even within this simplified picture one real worker (i.e. physical thread) could still be assigned the workload of any number of domain indices.

Functors passed into lockstep routines can have three different base parameter signatures. Additionally each case can be extended by an arbitrary number parameters to get access to context variables.

- No parameter, if the work is not requiring the linear index within a domain: `[&](){ }`
- An unsigned 32bit integral parameter if the work depends on indices within the domain `range [0,domain size)`: `[&](uint32_t const linearIdx){}`
- `lockstep::Idx` as parameter. lockstep::Idx is holding the linear index within the domain and meta information to access a context variables: `[&](pmacc::mappings::threads::lockstep::Idx const idx){}`

Context variables, over worker distributed arrays, can be passed as additional arguments to the lockstep foreach. The corresponding data for each index element of the domain will be passed as additional argument to the lambda function.

44.1 The naming used for methods or members

- `*DomSize` is the index domain size as scalar value, typically an integral type
- `*DomSizeND` is the N-dimensional index domain size, typically of the type `pmacc::math::Vector<>` or `pmacc::math::CT:Vector`
- `*DomIdx` is the index domain element as scalar value, typically an integral type
- `*DomIdxND` is the N-dimensional index domain element, typically of the type `pmacc::math::Vector<>`
- `*Size` is the size of data as scalar value
- `*SizeND` is the N-dimensional data size, typically of the type `pmacc::math::Vector<>`

44.2 pmacc helpers

template<uint32_t **T_domainSize**, uint32_t **T_numWorkers**, uint32_t **T_simdSize**>

struct **Config**

describe a constant index domain

describe the size of the index domain and the number of workers to operate on a lockstep domain

Template Parameters

- **T_domainSize** – number of indices in the domain
- **T_numWorkers** – number of worker working on T_domainSize
- **T_simdSize** – SIMD width

struct **Idx**

Hold current index within a lockstep domain.

template<typename **T_Acc**, typename **T_BlockCfg**>

class **Worker**

Entity of an worker.

Context object used for lockstep programming. This object is providing access to the alpaka accelerator and indicies used for the lockstep programming model.

Template Parameters

T_numSuggestedWorkers – Suggested number of lockstep workers. Do not assume that the suggested number of workers is used within the kernel. The real used number of worker can be queried with numWorkers() or via the member variable numWorkers.

template<typename **T_Type**, typename **T_Config**>

struct **Variable** : protected *pmacc*::memory::Array<*T_Type*, *T_Config*::maxIndicesPerWorker>, public *T_Config*

Variable used by virtual worker.

This object is designed to hold context variables in lock step programming. A context variable is just a local variable of a virtual worker. Allocating and using a context variable allows to propagate virtual worker states over subsequent lock steps. A context variable for a set of virtual workers is owned by their (physical) worker.

Data stored in a context variable should only be used with a lockstep programming construct e.g. lockstep::ForEach<>

template<typename **T_Worker**, typename **T_Config**>

class **ForEach**

Execute a functor for the given index domain.

Algorithm to execute a subsequent lockstep for each index of the configured domain.

Attention

There is no implicit synchronization between workers before or after the execution of is *ForEach* performed.

Template Parameters

T_Config – Configuration for the domain and execution strategy. T_Config must provide: domainSize, numCollIter, numWorkers, and simdSize at compile time.

44.3 Common Patterns

44.3.1 Create a Context Variable

A context variable is used to transfer information from a subsequent lockstep to another. You can use a context variable `lockstep::Variable`, similar to a temporary local variable in a function. A context variable must be defined outside of `ForEach` and should be accessed within the functor passed to `ForEach` only.

- … and initialize with the index of the domain element

```
// variable 'worker' is provided by pmacc if the kernel launch macro `PMACC_LOCKSTEP_
↪KERNEL()` is used.
constexpr uint32_t frameSize = 256;
auto forEachParticleSlotInFrame = lockstep::makeForEach<frameSize>(worker);
auto elemIdx = forEachParticleSlotInFrame(
    [](lockstep::Idx const idx) -> int32_t
    {
        return idx;
    }
);

// is equal to

// assume one dimensional indexing of threads within a block
constexpr uint32_t frameSize = 256;
auto forEachParticleSlotInFrame = lockstep::makeForEach<frameSize>(worker);
// variable will be uninitialized
auto elemIdx = lockstep::makeVar<int32_t>(forEachParticleSlotInFrame);
forEachParticleSlotInFrame(
    [&](uint32_t const idx, auto& vIndex)
    {
        vIndex = idx;
    },
    elemIdx
);
// is equal to
forEachParticleSlotInFrame(
    [&](lockstep::Idx const idx)
    {
        elemIdx[idx] = idx;
    }
);
```

- To default initialize a context variable you can pass the arguments directly during the creation.

```
// variable 'worker' is provided by pmacc if the kernel launch macro `PMACC_LOCKSTEP_
↪KERNEL()` is used.
constexpr uint32_t frameSize = 256;
auto forEachParticleSlotInFrame = lockstep::makeForEach<frameSize>(worker);
auto var = lockstep::makeVar<int32_t>(forEachParticleSlotInFrame, 23);
```

- Data from a context variable can be accessed within independent lock steps. Only data elements those correspond to the element index of the domain can be accessed.

```
// variable 'worker' is provided by pmacc if the kernel launch macro `PMACC_LOCKSTEP_
↪KERNEL()` is used.
constexpr uint32_t frameSize = 256;
auto forEachParticleSlotInFrame = lockstep::makeForEach<frameSize>(worker);
```

(continues on next page)

(continued from previous page)

```
auto elemIdx = forEachParticleSlotInFrame(
    [](uint32_t const idx) -> int32_t
    {
        return idx;
    }
);

// store old linear index into oldElemIdx
auto oldElemIdx = forEachExample(
    [&](lockstep::Idx const idx) -> int32_t
    {
        int32_t old = elemIdx[idx];
        printf("domain element idx: %u == %u\n", elemIdx[idx], idx);
        elemIdx[idx] += 256;
        return old;
    }
);

// To avoid convusion between read-only and read-write input variables we suggest␣
↪using
// const for read only variables.
forEachExample(
    [&](lockstep::Idx const idx, int32_t const oldIndex, int32_t const vIndex)
    {
        printf("nothing changed: %u == %u - 256 == %u\n", oldIndex, vIndex, idx);
    },
    oldElemIdx,
    elemIdx
);
```

44.3.2 Collective Loop over particles

- each worker needs to pass a loop N times
- in this example, there are more dates than workers that process them

```
// variable 'worker' is provided by pmacc if the kernel launch macro `PMACC_LOCKSTEP_
↪KERNEL()` is used.
// `frame` is a list which must be traversed collectively
while( frame.isValid() )
{
    // assume one dimensional indexing of threads within a block
    constexpr uint32_t frameSize = 256;
    auto forEachParticleSlotInFrame = lockstep::makeForEach<frameSize>(worker);
    forEachParticleSlotInFrame(
       [&](lockstep::Idx const idx)
       {
           // independent work, idx can be used to access a context variable
       }
    forEachParticleSlotInFrame(
       [&](uint32_t const linearIdx)
       {
           // independent work based on the linear index only, e.g. shared memory␣
↪access
       }
```

(continues on next page)

(continued from previous page)

```
    );
}
```

44.3.3 Non-Collective Loop over particles

- each element index of the domain increments a private variable

```
// variable 'worker' is provided by pmacc if the kernel launch macro `PMACC_LOCKSTEP_
↪KERNEL()` is used.
constexpr uint32_t frameSize = 256;
auto forEachParticleSlotInFrame = lockstep::makeForEach<frameSize>(worker);
auto vWorkerIdx = lockstep::makeVar<int32_t>(forEachParticleSlotInFrame, 0);
forEachParticleSlotInFrame(
    [&](auto const idx, int32_t& vWorker)
    {
        // assign the linear element index to context variable
        vWorker = idx;
        for(int i = 0; i < 100; i++)
            vWorker++;
    },
    vWorkerIdx
);
```

44.3.4 Using a Master Worker

- only a single element index of the domain (called *master*) manipulates a shared data structure for all others

```
// example: allocate shared memory (uninitialized)
PMACC_SMEM(
    finished,
    bool
);

// variable 'worker' is provided by pmacc if the kernel launch macro `PMACC_LOCKSTEP_
↪KERNEL()` is used.
auto onlyMaster = lockstep::makeMaster(worker);

// manipulate shared memory
onlyMaster(
    [&]( )
    {
        finished = true;
    }
);

/* important: synchronize now, in case upcoming operations (with
 * other workers) access that manipulated shared memory section
 */
worker.sync();
```

44.4 Practical Examples

If possible kernels should be written without assuming any lockstep domain size and number of alpaka blocks selected at the kernel start. This ensure that the kernel results are always correct even if the user doesn't chose the right parameters for the kernel execution.

```
struct IotaGenericKernel
{
    template<typename T_Worker, typename T_DataBox>
    HDINLINE void operator()(T_Worker const& worker, T_DataBox data, uint32_
↪t size) const
    {
        constexpr uint32_t blockDomSize = T_Worker::blockDomSize();
        auto numDataBlocks = (size + blockDomSize - 1u) / blockDomSize;

        // grid-strided loop over the chunked data
        for(int dataBlock = worker.blockDomIdx(); dataBlock < numDataBlocks;
↪ dataBlock += worker.gridDomSize())
        {
            auto dataBlockOffset = dataBlock * blockDomSize;
            auto forEach = pmacc::lockstep::makeForEach(worker);
            forEach(
                [&](uint32_t const inBlockIdx)
                {
                    auto idx = dataBlockOffset + inBlockIdx;
                    if(idx < size)
                    {
                        // ensure that each block is not overwriting data␣
↪from other blocks
                        PMACC_DEVICE_VERIFY_MSG(data[idx] == 0u, "%s\n",
↪"Result buffer not valid initialized!");
                        data[idx] = idx;
                    }
                });
        }
    }
};

template<uint32_t T_chunkSize, typename T_DeviceBuffer>
inline void iotaGerneric(T_DeviceBuffer& devBuffer)
{
    auto bufferSize = devBuffer.size();
    // use only half of the blocks needed to process the full data
    uint32_t const numBlocks = bufferSize / T_chunkSize / 2u;
    PMACC_LOCKSTEP_KERNEL(IotaGenericKernel{}).config<T_chunkSize>
↪(numBlocks)(devBuffer.getDataBox(), bufferSize);
}
```

The block domain size can also be derived from a instance of any object if the trait `pmacc::lockstep::traits::MakeBlockCfg` is defined.

```
namespace pmacc::lockstep::traits
{
    //! Specialization to create a lockstep block configuration out of a␣
↪device buffer.
    template<>
    struct MakeBlockCfg<pmacc::DeviceBuffer<uint32_t, DIM1>> : std::true_
```

(continues on next page)

(continued from previous page)

```
↪type
    {
        using type = BlockCfg<math::CT::UInt32<53>>;
    };
} // namespace pmacc::lockstep::traits

template<typename T_DeviceBuffer>
inline void iotaGernericBufferDerivedChunksize(T_DeviceBuffer& devBuffer)
{
    auto bufferSize = devBuffer.size();
    constexpr uint32_t numBlocks = 9;
    PMACC_LOCKSTEP_KERNEL(IotaGenericKernel{}).config(numBlocks,␣
↪devBuffer)(devBuffer.getDataBox(), bufferSize);
}
```

Sometimes it is not possible to write a generic kernel and a hard coded block domain size is required to fulfill stencil condition or other requirements. In this case it is possible to use on device `pmacc::lockstep::makeForEach<hardCodedBlockDomSize>(worker)`. The problem is that the user needs to know this hard coded requirement during the kernel call else it could be the kernel is running slow. It is possible that too many worker threads are idling during the execution because the selected block domain during the kernel call is larger than the required block domain within the kernel. By defining the member variable `blockDomSize` and not providing the block domain size during the kernel configuration the kernel will be executed automatically with the block domain size specialized by the kernel. Overwriting the block domain size during the kernel execution is triggering a static assertion during compiling.

```
struct IotaFixedChunkSizeKernel
{
    static constexpr uint32_t blockDomSize = 42;

    template<typename T_Worker, typename T_DataBox>
    HDINLINE void operator()(T_Worker const& worker, T_DataBox data, uint32_
↪t size) const
    {
        static_assert(blockDomSize == T_Worker::blockDomSize());

        auto numDataBlocks = (size + blockDomSize - 1u) / blockDomSize;

        // grid-strided loop over the chunked data
        for(int dataBlock = worker.blockDomIdx(); dataBlock < numDataBlocks;
↪ dataBlock += worker.gridDomSize())
        {
            auto dataBlockOffset = dataBlock * blockDomSize;
            auto forEach = pmacc::lockstep::makeForEach(worker);
            forEach(
                [&](uint32_t const inBlockIdx)
                {
                    auto idx = dataBlockOffset + inBlockIdx;
                    if(idx < size)
                    {
                        // ensure that each block is not overwriting data␣
↪from other blocks
                        PMACC_DEVICE_VERIFY_MSG(data[idx] == 0u, "%s\n",
↪"Result buffer not valid initialized!");
                        data[idx] = idx;
                    }
                });
```

(continues on next page)

(continued from previous page)

```
        }
    }
};

template<typename T_DeviceBuffer>
inline void iotaFixedChunkSize(T_DeviceBuffer& devBuffer)
{
    auto bufferSize = devBuffer.size();
    constexpr uint32_t numBlocks = 10;
    PMACC_LOCKSTEP_KERNEL(IotaFixedChunkSizeKernel{}).
↪config(numBlocks)(devBuffer.getDataBox(), bufferSize);
}
```

Equally to the scalar block domain size `blockDomSize` a member type `BlockDomSizeND` of the pmacc type `pmacc::math::CT::Uint32<>` can be defined to express a N-dimensional block domain. `blockDomSize` and `BlockDomSizeND` are mutual exclusive and can not be defined at the same time for a kernel.

```
struct IotaFixedChunkSizeKernelND
{
    using BlockDomSizeND = pmacc::math::CT::UInt32<42>;

    template<typename T_Worker, typename T_DataBox>
    HDINLINE void operator()(T_Worker const& worker, T_DataBox data, uint32_
↪t size) const
    {
        static constexpr uint32_t blockDomSize = BlockDomSizeND::x::value;

        static_assert(blockDomSize == T_Worker::blockDomSize());

        // grid-strided loop over the chunked data
        auto numDataBlocks = (size + blockDomSize - 1u) / blockDomSize;

        for(int dataBlock = worker.blockDomIdx(); dataBlock < numDataBlocks;
↪ dataBlock += worker.gridDomSize())
        {
            auto dataBlockOffset = dataBlock * blockDomSize;
            auto forEach = pmacc::lockstep::makeForEach(worker);
            forEach(
                [&](uint32_t const inBlockIdx)
                {
                    auto idx = dataBlockOffset + inBlockIdx;
                    if(idx < size)
                    {
                        // ensure that each block is not overwriting data␣
↪from other blocks
                        PMACC_DEVICE_VERIFY_MSG(data[idx] == 0u, "%s\n",
↪"Result buffer not valid initialized!");
                        data[idx] = idx;
                    }
                });
        }
    }
};

template<typename T_DeviceBuffer>
inline void iotaFixedChunkSizeND(T_DeviceBuffer& devBuffer)
```

(continues on next page)

(continued from previous page)

```
{
    auto bufferSize = devBuffer.size();
    constexpr uint32_t numBlocks = 11;
    PMACC_LOCKSTEP_KERNEL(IotaFixedChunkSizeKernelND{}).
↪config(numBlocks)(devBuffer.getDataBox(), bufferSize);
}
```

To use dynamic shared memory within a lockstep kernel the kernel must be configured with `configSMem` instead of *config*

```
struct IotaGenericKernelWithDynSharedMem
{
    template<typename T_Worker, typename T_DataBox>
    HDINLINE void operator()(T_Worker const& worker, T_DataBox data, uint32_
↪t size) const
    {
        constexpr uint32_t blockDomSize = T_Worker::blockDomSize();
        auto numDataBlocks = (size + blockDomSize - 1u) / blockDomSize;

        uint32_t* s_mem = ::alpaka::getDynSharedMem<uint32_t>(worker.
↪getAcc());

        // grid-strided loop over the chunked data
        for(int dataBlock = worker.blockDomIdx(); dataBlock < numDataBlocks;
↪ dataBlock += worker.gridDomSize())
        {
            auto dataBlockOffset = dataBlock * blockDomSize;
            auto forEach = pmacc::lockstep::makeForEach(worker);
            forEach(
                [&](uint32_t const inBlockIdx)
                {
                    auto idx = dataBlockOffset + inBlockIdx;
                    s_mem[inBlockIdx] = idx;
                    if(idx < size)
                    {
                        // ensure that each block is not overwriting data␣
↪from other blocks
                        PMACC_DEVICE_VERIFY_MSG(data[idx] == 0u, "%s\n",
↪"Result buffer not valid initialized!");
                        data[idx] = s_mem[inBlockIdx];
                    }
                });
        }
    }
};

template<uint32_t T_chunkSize, typename T_DeviceBuffer>
inline void iotaGernericWithDynSharedMem(T_DeviceBuffer& devBuffer)
{
    auto bufferSize = devBuffer.size();
    // use only half of the blocks needed to process the full data
    uint32_t const numBlocks = bufferSize / T_chunkSize / 2u;
    constexpr size_t requiredSharedMemBytes = T_chunkSize * sizeof(uint32_
↪t);
    PMACC_LOCKSTEP_KERNEL(IotaGenericKernelWithDynSharedMem{})
        .configSMem<T_chunkSize>(numBlocks,␣
```

(continues on next page)

(continued from previous page)

```
↪requiredSharedMemBytes)(devBuffer.getDataBox(), bufferSize);
}
```

CHAPTER

FORTYFIVE

INTRO

This chapter describes the internals of the PICMI translation (and subsequently the internal representation PyPIConGPU).

Warning: This section is aimed at **developers**. It describes how the *PICMI user interface* is implemented.

For the PICMI in PIConGPU user documentation, see *here*. For the PICMI standard (upstream) definition, see here.

If you read this documentation like a book carry on in the order of the table of contents. However, the recommended way is to also review the implementation side-by-side to this documentation:

To get started familiarize yourself with the *translation process* and skim how the *testing approach* works. Read the *FAQ* for some more fundamental questions. After looking at some examples of implementations yourself, continue with *running*, the *general notes on the implementation*, and read the notes on *how to write schemas*. If you have to work with species also read *their section*. (Note: Species are by far the most complex issue in PyPIConGPU, make sure you understand the fundamentals before working on them.)

CHAPTER

FORTYSIX

TRANSLATION PROCESS

This document explains the semantic and syntactic transformation from PICMI inputs to PIConGPU build files. It also describes when wich type of error is caught or raised.

The data are transformed along the following pipeline:

1. User script
    - is: python code, has `from picongpu import picmi`
    - purpose: input in generic programming language, editable by user
    - defined in: user file (not part of this repo)
2. PICMI objects
    - is: python objects specified by upstream `picmistandard`
        - note that the objects upstream are **inherited** (i.e. **NOT** copied)
    - purpose: user interface
    - located in: `lib/python/pypicongpu/picmi`
3. PyPIConGPU objects (1-to-1 translateable to C++ code)
    - is: python
    - purposes:
        - use PIConGPU representation instead of PICMI (number of cells vs extent in SI units etc.)
        - consruct valid input parameters (where one can also modify data on a semantic level)
        - possibly even modify parameters (e.g. "electron resolution")
    - "intermediate representation"
    - no redundancies
    - aims to be close to PIConGPU native input (in terms of structure)
    - located in: `lib/python/pypicongpu` (everthing that is not PICMI)
4. JSON representation ("rendering context")
    - is: JSON (simple hierarchical data format)
    - purpose: passed to template engine to generate code (also: separate translation logic from code generation)
    - slightly restricted version of json
    - generated according to predefined schemas
    - may contain rendundancies
    - located in:
        - generated in `_get_serialized()` implementations

    - checked against schemas in `share/pypicongpu/schema`
    - stored (for debugging) in generated files as `pypicongpu.json`

5. `.param` and `.cfg` files
   - is: files as defined by PIConPGU
   - purpose: fully compatible to vanilla PIConGPU
   - generated using predefined template
   - created using "templating engine" mustache, terminology:
     - *render*: generate final code by applying a *context* to a template
     - *context*: (JSON-like) data that is combined with a template during rendering
     - a *templating engine* is a more powerful search replace; *rendering* describes transforming a *template* (where things are replaced) with a *context* (what is inserted)
   - located in:
     - generated directory
     - templates (`*.mustache`) in `share/pypicongpu/template`
6. (normal PIConGPU pipeline continues: `pic-build`, `tbg`)
   - `pic-build` compile the code
     - wrapper around cmake
     - (would be run standalone for native PIConGPU)
   - `tbg` runs the code
     - enable usage on different platforms/hpc clustern/submit systems

Please see at the end of this document for an example.

Generally, each object has a method to transform it into the next step: PICMI objects have a method `get_as_pypicongpu()`, pypicongpu object have a method `get_rendering_context()` etc.

The individual steps are fairly dumb (see: KISS), so if you do something simple it should work – and anything complicated won't.

The individual data formats and their transformations are outlined below.

Practically this pipeline is invoked by the runner, see *the corresponding documentation chapter*.

46.1 PICMI

PICMI objects are objects as defined by the picmistandard.

A user script creates PICMI objects, all collected inside the `Simulation` object. Technically these objects are implemented by pypicongpu, though (almost) all functionality is inherited from the implementation inside the picmistandard.

The content of the PICMI objects is assumed to be conforming to standard. PICMI (upstream) does not necessarily enforce this. The user may define invalid objects. Such objects outside the standard produce undefined behavior!

When invoking `get_as_pypicongpu()` on a PICMI object it is translated to the corresponding pypicongpu object. Not all PICMI objects have a direct pypicongpu translation, in which case the object owning the non-translateable object has to manage this translation (e.g. the simulation extracts the PICMI grid translation from the PICMI solver). Before this translations happens, some simple checks are performed, mainly for compatibility. When an incompatible parameter is encountered, one of two things can happen:

1. A utility function is called to notify the user that the parameter is unsupported. This typically applies to parameters that are not supported at all.
2. An assertions is triggered. In this case python will create an `AssertionError`, and an associated message is provided. These errors can't be ignored. This typically applies to parameters for which only some values are supported.

The translation from PICMI to PyPIConGPU does not follow a "great design plan", the approach can be summed up by:

- have unit tests for everything (which are in turn not well organized, but just huge lists of test cases)
- trust PyPIConGPUs self-validation to catch configuration errors
- translate as *local* as possible, i.e. translate as much as possible within the lowest hierachy level, and only rarely leave translation to the upper level (e.g. the all-encompassing simulation)

  The PICMI to PyPIConGPU translation itself is rather messy (and there is no way to entirely avoid that).

46.2 PyPIConGPU

PyPIConGPU objects are objects defined in pypicongpu which resemble the PIConGPU much more closely than plain PICMI. Hence, they (may) hold different data/parameters/formats than the PICMI objects.

As PICMI, all PyPIConGPU objects are organized under the top-level `Simulation` object.

> Be aware that **both** PICMI **and** PyPIConGPU have a class named `Simulation`, but these classes are **different**.

Typically, PyPIConGPU objects do not contain much logic – they are structures to hold data. E.g. the 3D grid is defined as follows (docstrings omitted here):

```python
@typechecked
class Grid3D(RenderedObject):
    cell_size_x_si = util.build_typesafe_property(float)
    cell_size_y_si = util.build_typesafe_property(float)
    cell_size_z_si = util.build_typesafe_property(float)

    cell_cnt_x = util.build_typesafe_property(int)
    cell_cnt_y = util.build_typesafe_property(int)
    cell_cnt_z = util.build_typesafe_property(int)

    boundary_condition_x = util.build_typesafe_property(BoundaryCondition)
    boundary_condition_y = util.build_typesafe_property(BoundaryCondition)
    boundary_condition_z = util.build_typesafe_property(BoundaryCondition)
```

In particular, please note:

- The annotation `@typechecked`: This is a decorator introduced by `typeguard` and it ensures that the type annotations of methods are respected. However, it does not perform typechecking for attributes, which is why the attributes are delegated to:
- `util.build_typesafe_property()` ...is a helper to build a property which automatically checks that only the type specified is used. Additionally, it **does not allow default values**, i.e. a value must be **set explicitly**. If it is read before a write an error is thrown.
- The parent class `RenderedObject`: Inheriting from `RenderedObject` means that this object can be translated to a *context* for the templating egine (see JSON representation). The inheriting Object `Grid3D` must implement a method `_get_serialized(self) -> dict` (not shown here), which returns a dictionary representing the internal state of this object for rendering. It is expected that two object which return the same result for `_get_serialized()` are equal. This method is used by `RenderedObject` to provide

the `get_rendering_context()` which invokes `_get_serialized()` internally and performs additional checks (see next section).

Some objects only exist for processing purposes and do not (exclusively) hold any simulation parameters, e.g. the `Runner` (see *Running*) or the `InitializationManager` (see *Species translation*).

46.3 JSON representation ("rendering context")

A JSON representation is fed into the templating engine to *render* a template.

The rendering engine used here is Mustache, implemented by chevron.

Think of a templating engine as a more powerful search-replace. Please refer to the mustache documentation for further details. (The authoritative spec outlines some additional features.)

We apply the Mustache standard more strictly than necessary, please see below for further details.

Motivation: Why Mustache?

There are plenty of templating engines available. Most of these allow for much more logic inside the template than mustache does, even being turing complete. However, having a lot of logic inside of the template itself should be avoided, as they make the code much less structured and hence more difficult to read, test and maintain. Mustache itself is not very powerful, forcing the programmer to define their logic inside of PyPIConGPU. In other words: The intent is to disincentivize spaghetti template code.

The JSON representation is created inside of `_get_serialized()` implemented by classes inheriting from `RenderedObject`. Before it is passed to the templating engine, it has to go through three additional steps:

1. check for general structure
2. check against schema
3. JSON preprocessor

Notably, the schema check is actually performed *before* the general structure check, but conceptionally the general structure check is more general as the schema check.

46.3.1 Check General Structure

This check is implemented in `Renderer.check_rendering_context()` and ensures that the python dict returned by `_get_serialized()` can be used for Mustache as rendering context, in particular:

- context is `dict`
- all keys are strings
  - keys do **NOT** contain dot `.` (reserved for element-wise access)
  - keys do **NOT** begin with underscore `_` (reserved for preprocessor)
- values are either dict, list or one of:
  - None
  - boolean
  - int or float
  - string
- list items must be dictionaries This is due to the nature of Mustache list processing (loops): The loop header for Mustache `{{#list}}` does enter the context of the list items, e.g. for `[{"num": 1"}, {"num": 2}]` `num` is now defined after the loop header. This *entering the context* is not possible if the item is not a dict, e.g. for `[1, 2]` it is not clear to which variable name the value is bound after the loop header. Such simpe lists **can't be handled by Mustache** and hence are caught during this check.

Simply put, this check ensures that the given dict can be represented as JSON and can be processed by mustache. It is **independent** from the origin of the dict.

> Notably native mustache actually *can*, in fact, handle plain lists. The syntax is not straight-forward though, hence we forbid it here. (For details see mustache spec, "Implicit Iterators")

46.3.2 Schema Check

This check is implemented in `RenderedObject.get_rendering_context()` as is performed on the dict returned by `_get_serialized()`.

It ensures that the structure of this object conforms to a predefined schema **associated with the generating class**.

> A *schema* in general defines a structure which some data follows. E.g. OpenPMD can be seen as a schema. Some database management systems call their tables *schemas*, XML has *schemas* etc. For the JSON data here JSON schema is used. At the time of writing, JSON schema has not been standardized into an RFC, and the version **Draft 2020-12** is used throughout. To check the schemas, the library jsonschema is employed.
>
> live online JSON schema validator | comprehensive guid

The schemas that are checked against are located at `share/pypicongpu/schema/`. All files in this directory are crawled and added into the schema database. **One file may only define one schema.**

To associate the classes to their schemas, their **Fully Qualified Name** (FQN) is used. It is constructed from the modulename and the class name, i.e. the PyPIConGPU simulation object has the FQN `pypicongpu.simulation.Simulation`. The FQN is appended to the URL `https://registry.hzdr.de/crp/picongpu/schema/` which is used as identifier (`$id`) of a schema.

E.g. the Yee Solver class' schema is defined as:

```
{
    "$id": "https://registry.hzdr.de/crp/picongpu/schema/pypicongpu.solver.YeeSolver",
    "type": "object",
    "properties": {
        "name": {
            "type": "string"
        }
    },
    "required": ["name"],
    "unevaluatedProperties": false
}
```

which is fullfilled by it serialization:

```
{"name": "Yee"}
```

The URL (`https://registry.hzdr.de/crp/picongpu/schema/pypicongpu.solver.YeeSolver`) can be used to refer to a serialized YeeSolver, e.g. by the PyPIConGPU Simulation schema.

For all schema files, the following is checked:

- Have an `$id` (URL) set (if not log error and skip file)
- Have `unevaluatedProperties` set to `false`, i.e. do not allow additional properties (if not log warning and continue)

If no schema can be found when translating an object to JSON operation is aborted.

46.3.3 JSON preprocessor

If the created context object (JSON) passes all checks (structure + schema) it is passed to the preprocessor.

> Before any preprocessing is applied (but after all checks have passed) the runner dumps the used context object into `pypicongpu.json` inside the setup directory.

The preprocessor performs the following tasks:

- Translate all numbers to C++-compatible literals (stored as strings, using sympy)
- Add the properties `_first` and `_last` to all list items, set to `true` or `false` respectively (to ease generation of list separators etc.)
- Add top-level attributes, e.g. the current date as `_date`.

46.3.4 Rendering Process

The rendering process itself is launched inside of Runner in `Runner.generate()`. This creates the "setup directory" by copying it from the template, which contains many `NAME.mustache` files. These are the actual string templates, which will be rendered by the templating engine.

After the setup directory copy the following steps are performed (inside of `Runner.__render_templates()`):

1. Retrieve rendering context of the all-encompassing PyPIConGPU simulation object
   - The PyPIConGPU simulation object is responsible for calling translate-to-rendering-context methods of the other objects.
   - This automatically (implicitly) checks against the JSON schema
2. The rendering context general structure is checked (see above)
3. Dump the fully checked rendering context into `pypicongpu.json` in the setup dir
4. preprocess the context (see above)
5. Render all files `NAME.mustache` to `NAME` using the context (including all child dirs)
   - check that file `NAME` mustache does not exist, if it does abort
   - check syntax according to rules outlined below, if violated warn and continue
   - print warning on undefined variables and continue
6. rename the fully rendered template `NAME.mustache` to `.NAME.mustache`; rationale:
   - keep it around for debugging and investigation
   - these files are fairly small, they don't hurt (too bad)
   - hide it from users, so they don't confuse the generated file (which is actually used by PIConGPU) with the template `NAME.mustache` (which is entirely ignored by PIConGPU)

Due to the warnings on undefined variables optional parameters are should **never** be omitted, but explicitly set to null if unused. E.g. the laser in the PyPIConGPU simulation is expected by this (sub-) schema:

```
"laser": {
    "anyOf": [
        {
            "type": "null"
        },
        {
            "$ref": "https://registry.hzdr.de/crp/picongpu/schema/pypicongpu.laser.
↪GaussianLaser"
        }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

which makes both `{..., "laser": null, ...}` and `{..., "laser": {...}, ...}` valid – but in both cases the variable `laser` is defined.

Notably, from this process' perspective "the rendering context" is the rendering context of the PyPIConGPU simulation object.

46.3.5 Mustache Syntax

Mustache syntax is used as defined by the Mustache Spec, whith the following exceptions:

The Mustache Language Documentation is the human-readable explanation, though it omits some details.

- Variables are always inserted using **3** braces: `{{{value}}}`. Using only two braces indicates that the value should be HTML-escaped, which is not applicable to this code generation. Before rendering, all code is checked by a (not-fully correct) regex, and if only two braces are found a warning is issued.
- Subcomponents of objects can be accessed using the dot `.`, e.g. `nested.object.value` returns `4` for `{"nested": {"object": {"value": "4"}}}`. This is a mustache standard feature and therefore supported by the used library chevron, though it is not mentioned in the documentation linked above.
- Unkown variables are explicitly warned about. Standard behavior would be to pass silently, treating them as empty string. Notably this also applies to variables used in conditions, e.g. `{{^laser}}` would issue a warning if laser is not set. Due to that **all used variables** should **always** be defined, if necessary set to null (`None` in Python).
- Partials are not available
- Lambdas are not available

46.4 Example Sequence

These examples should demonstrate how the translation process works.

46.4.1 Bounding Box

The Bounding Box is defined as a grid object. PICMI does not use a global grid object, but PIConGPU does.

So when invoking `picmi.Simulation.get_as_pypicongpu()` it uses the grid from the picmi solver:

```python
# pypicongpu simulation
s = simulation.Simulation()
s.grid = self.solver.grid.get_as_pypicongpu()
```

`picmi.grid.get_as_pypicongpu()` (here `Cartesian3DGrid`) checks some compatibility stuff, e.g. if the lower bound is correctly set to 0,0,0 (only values supported) and if the upper and lower boundary conditions are the same for each axis. For unsupported features a util method is called:

```python
assert [0, 0, 0] == self.lower_bound, "lower bounds must be 0, 0, 0"
assert self.lower_boundary_conditions == self.upper_boundary_conditions, "upper and␣
↪lower boundary conditions must be equal (can only be chosen by axis, not by␣
↪direction)"

# only prints a message if self.refined_regions is not None
util.unsupported("refined regions", self.refined_regions)
```

PIConGPU does not use bounding box + cell count but cell count + cell size, so this is translated before returning a pypicongpu grid:

```
# pypicongpu grid
g = grid.Grid3D()
g.cell_size_x_si = (self.xmax - self.xmin) / self.nx
g.cell_size_y_si = (self.ymax - self.ymin) / self.ny
g.cell_size_z_si = (self.zmax - self.zmin) / self.nz
g.cell_cnt_x = self.nx
g.cell_cnt_y = self.ny
g.cell_cnt_z = self.nz
# ...
return g
```

The pypicongpu `Grid3D._get_serialized()` now translates these parameters to JSON (a python dict):

```
def _get_serialized(self) -> dict:
    return {
        "cell_size": {
            "x": self.cell_size_x_si,
            "y": self.cell_size_y_si,
            "z": self.cell_size_z_si,
        },
        "cell_cnt": {
            "x": self.cell_cnt_x,
            "y": self.cell_cnt_y,
            "z": self.cell_cnt_z,
        },
        "boundary_condition": {
            "x": self.boundary_condition_x.get_cfg_str(),
            "y": self.boundary_condition_y.get_cfg_str(),
            "z": self.boundary_condition_z.get_cfg_str(),
        }
    }
```

By invoking `grid.get_rendering_context()` in the owning `Simulation` object this is checked against the schema located in `share/pypicongpu/schema/pypicongpu.grid.Grid3D.json`

```
{
    "$id": "https://registry.hzdr.de/crp/picongpu/schema/pypicongpu.grid.Grid3D",
    "description": "Specification of a (cartesian) grid of cells with 3 spacial␣
↪dimensions.",
    "type": "object",
    "properties": {
        "cell_size": {
            "description": "width of a single cell in m",
            "type": "object",
            "unevaluatedProperties": false,
            "required": ["x", "y", "z"],
            "properties": {
                "x": {
                    "$anchor": "cell_size_component",
                    "type": "number",
                    "exclusiveMinimum": 0
                },
                "y": {
                    "$ref": "#cell_size_component"
                },
```

(continues on next page)

(continued from previous page)

```
                "z": {
                    "$ref": "#cell_size_component"
                }
            }
        },
        "cell_cnt": {},
        "boundary_condition": {
            "description": "boundary condition to be passed to --periodic (encoded as␣
↪number)",
            "type": "object",
            "unevaluatedProperties": false,
            "required": ["x", "y", "z"],
            "properties": {
                "x": {
                    "$anchor": "boundary_condition_component",
                    "type": "string",
                    "pattern": "^(0|1)$"
                },
                "y": {
                    "$ref": "#boundary_condition_component"
                },
                "z": {
                    "$ref": "#boundary_condition_component"
                }
            }
        }
    },
    "required": [
        "cell_size",
        "cell_cnt",
        "boundary_condition"
    ],
    "unevaluatedProperties": false
}
```

This entire process has been launched by the `Runner`, which now dumps these parameters to the `pypicongpu.json` before continuing rendering:

```
{
    "grid": {
        "cell_size": {
            "x": 1.776e-07,
            "y": 4.43e-08,
            "z": 1.776e-07
        },
        "cell_cnt": {
            "x": 192,
            "y": 2048,
            "z": 12
        },
        "boundary_condition": {
            "x": "0",
            "y": "0",
            "z": "1"
        }
    },
}
```

These are now used by the template from `share/pypicongpu/template`, e.g. in the `N.cfg.mustache`:

```
{{#grid.cell_cnt}}
TBG_gridSize="{{{x}}} {{{y}}} {{{z}}}"
{{/grid.cell_cnt}}

TBG_steps="{{{time_steps}}}"

{{#grid.boundary_condition}}
TBG_periodic="--periodic {{{x}}} {{{y}}} {{{z}}}"
{{/grid.boundary_condition}}
```

and in `grid.param.mustache`:

```
constexpr float_64 DELTA_T_SI = {{{delta_t_si}}};

{{#grid.cell_size}}
constexpr float_64 CELL_WIDTH_SI = {{{x}}};
constexpr float_64 CELL_HEIGHT_SI = {{{y}}};
constexpr float_64 CELL_DEPTH_SI = {{{z}}};
{{/grid.cell_size}}
```

CHAPTER

FORTYSEVEN

TESTING STRATEGY

This document aims to explain which components are tested to which degree, and more generally how pypicongpu tests should be treated.

The extent of the tests tries to balance multiple factors:

- more tests means more work (and more maintenance down the road)
- fewer test mean more errors go unnoticed
- more important/fragile components should be tested more rigorously ("important" here means "could create a lot trouble down the road if it does not work as specified")
- some type of checks require a lot of code (which needs to be maintained), others do not

How these factors are traded against each other is layed out for each module below.

All of the tests are built to get broken – run them frequently and expand them if you add functionality! Some of the tests are rather rigid, so they might fail even if they are not supposed to. When encountering a failing test act with care and adjust the test case if that is plausible.

Note: For an explanation on the python `unittests` module please refer to the python manual.

For more notes on tools support (including test coverage) see *Tool Support*.

47.1 Structure

All tests are stored in the directory `test/python/picongpu/`. For convenience the tests are separated into several categories (stored in the respective directories):

- `quick`: tests which run in a short amount of time (a couple of seconds, typically unit tests)
- `compiling`: tests which run a longer amount of time (mainly the runner)
- `e2e`: tests which tests picongpu end-to-end

Inside of each of these directories the structure of the python source (`lib/python/picongpu/`) is replicated.

To execute the tests run their `__main__.py` by invoking from `test/python/picongpu/` (they don't work from anywhere else!): repository root:

```
python -m quick
python -m compiling
python -m e2e
```

Append `--help` for options. E.g. for quick debugging use `python -m quick -f --locals`. Individual test cases can be accessed with their class name, and optional method name:

```
python -m quick TestElement
python -m quick TestElement.test_periodic_table_names
```

Note: The tests are loaded by using `from SOMEWHERE import *` – which is bad style. For that reason all `__init__.py` files in the tests have the style checks disabled with `# flake8: noqa`. When changing the test runner (maybe in the future) these skipped checks should be abandoned.

47.2 PICMI

Any passed PICMI object is assumed to be correct. PICMI (upstream) does not necessarily enforce this – if that should be enforced please add checks upstream with the picmistandard instead of adding them in pypicongpu.

The method `get_as_pypicongpu()` is only tested to a shallow level: Most test cases are (significantly) longer than the assertions/checks, yet would usually only check if the values are copied correctly.

Such tests would achieve only a small benefit yet be fairly long and are hence kept to a minimum.

47.3 PyPIConGPU objects

The most important functionality of pypicongpu objects is storing data in a defined type and ensuring that mandatory parameters are provided. Both are ensured by using a utility function to generate getters/setters, and hence there is a pretty low chance to make a mistake there.

Yet there are some sanity checks performed, including the translation to CPP objects. As these translations are tightly coupled to the used `.param` file (templates), they could change fairly frequently and are as such not subject to rigerous testing.

47.4 PIConGPU project template

The `.param` and `.cfg` files where the params defined by pypicongpu are placed inside. These are not tested separately, as such tests would have to be pretty much equivalent to the end-to-end tests.

47.5 PIConGPU (and its build process)

PIConGPU and its build process (using `pic-build`) are assumed to be correct when used as documented.

47.6 End-To-End

> TODO: This functionality is not yet implemented

To verify the PICMI input standard there are a few examples provided for which both a PICMI input and an equivalent set of `.param` files exists.

Their respective outputs are compared programmatically, tiny differences are ignored, anything else is considered a failure.

As maintaining both `.param` and PICMI input files takes a lot of time and effort, there only exist a couple of such examples. To compensate for their low number, these examples themselves are fairly complicated.

CHAPTER

FORTYEIGHT

RUNNING

This document describes the process of executing a simulation after all parameters have been configured.

> This doument describes the current state of the implementation, which does not implement all necessary features for productive HPC use. For HPC it is best to just generate input files and ignore running a simulation entirely.

The starting point is a picmi simulation object, e.g. initialized like that:

```
from picongpu import picmi
grid = picmi.Cartesian3DGrid(number_of_cells=[192, 2048, 12],
                             lower_bound=[0, 0, 0],
                             upper_bound=[3.40992e-5, 9.07264e-5, 2.1312e-6],
                             lower_boundary_conditions=["open", "open", "periodic"],
                             upper_boundary_conditions=["open", "open", "periodic"])
solver = picmi.ElectromagneticSolver(method="Yee", grid=grid)
laser = picmi.GaussianLaser(0.8e-6, 5.0e-6 / 1.17741, 5.0e-15,
                            a0=8,
                            propagation_direction=[0, 1, 0],
                            focal_position=[0, 4.62e-5, 0]
                            )
sim = picmi.Simulation(time_step_size=1.39e-16, max_steps=int(2048), solver=solver)
sim.add_laser(laser, None)
```

The object `sim` now **bundles all information** and is consequently the only way we will refer to the parameters.

"Running" a simulation means the execution of the following steps:

1. *Generating* the code (`.param`, `.cfg`) from the given parameters and joining those files with the predifined template (`pic-create`)
2. *Building* (compiling) the PIConGPU simulation (`pic-build`)
3. *Running* the simulation (`tbg`) itself

These steps can be invoked using two main methods:

- the **PICMI native interface** (methods specified by picmi)
- the **PyPIConGPU Runner**

For basic operation the PICMI native interface is sufficient, but for a tight control of the individual steps please use the PyPIConGPU runner.

48.1 PICMI Native Interface

This describes the implementation of the methods to run the simulation as described by PICMI.

It is mainly supported to conform to PICMI, but PIConGPU is much more complicated than this interface.

Unless testing a small scenario, please use the Runner as specified below.

Internally, this creates a runner and calls the necessary steps. To retrieve this runner, call `sim.picongpu_get_runner()`. (See below for usage.) **This is not recommended to control the simulation run, because you will skip all sanity checks.** Instead, create the runner yourself using the methods outline below.

48.1.1 Only Create Input Data

```
# sim init...
sim.write_input_file("/scratch/mydepartment/myname/mydata/myproject/scenario05")
```

This creates an input data set to be consumed by PIConGPU – after this call, PICMI is no longer required for the operation at all. This, in turn, means that you now have to call the other tools (`pic-build`, `tbg`) on your own.

The passed location must be a not-yet-existing directory. After this call this directory will be populated and you can run `pic-build` there.

This is intended for environments where you can't use the Runner as described below.

48.1.2 Run the Simulation Directly

```
# sim init...
sim.step(sim.max_steps)
```

This runs the entire build pipeline.

PICMI specifies that you can run an arbitrary number of steps. **This is not the case for PyPIConGPU.** You can only run **all steps at once**. (Any other number of steps will raise an error.)

> Note: While it would theoretically be possible to run a PIConGPU simulation one step at a time, that would be ludicrously expensive in terms of execution overhead an memory throughput. If you want to do that, please use PIConGPU without PyPIConGPU.

This also **has no multi-device support**: Without additional configuration it is impossible to ensure that a simulation is correctly distributed across multiple nodes.

Unless a scratch directory is configured via the environment variable `$SCRATCH`, **the results will be saved to the temporary directory** (`/tmp`). In this case a fat warning will be printed.

To retrieve the result location, use:

```
# sim init...
sim.step(sim.max_steps)
runner = sim.picongpu_get_runner()

import os
print("results are located at: {}", os.path.join(runner.run_dir, "simOutput"))
```

Did I emphasize enough that you should rather use the runner?

48.2 PyPIConGPU Runner

The PyPIConGPU Runner object is the intended way for simulations to be created and executed.

It is initialized using either **a PICMI simulation object** or alternatively **a PyPIConGPU simulation object**. (Note: If PICMI is enough for you, you will probably never see a pypicongpu simulation object. In this case just forget the last paragraph, just use **your** simulation object.)

Additionally you can supply up to four locations (see next section), but all of them are optional. Generally speaking, just setting the environment variable `$SCRATCH` is enough for an okay-ish operation. (To help your future self please organize yourself a little bit better.)

The Runner object tries to help you by running plenty of checks on your configuration. If you did an obvious mistake it will instantly fail, non-obvious mistakes can still cause problems down the road – it is not uncommon for fails to occur after 2 minutes of building. (So don't run away 10 seconds after launch.)

Please **use the constructor for the settings**, as the majority of checks will occur there. You can access the settings directly (see: *Dirty Tricks*), but this will bypass many of the checks and should normally **only be done for reading** the settings.

48.2.1 Used Locations

- scratch directory (`scratch_dir`): Directory, where many results can be stored. Must be **accessable from all involved machines**. (PyPIConGPU does not check that!) If not specified, will be loaded from the environment variable `$SCRATCH`. Possibly just specify your scratch location in your environment where you also load PIConGPU. Note: Do not use your home as scratch on HPC. Will be left empty if not specified. Must already exist if specified.
- setup directory (`setup_dir`): Holds the parameter set and `pic-build` is called from here. Will be generated in temporary directory (`/tmp`) if not given. Must not yet exist when invoking the runner. If you just want to run a simulation, the setup dir is not important to you. Forget this paragraph.
- run directory (`run_dir`): Holds all data associated with a single PIConGPU run. This includes the input data (copy of setup dir) and **all results**, typically **inside the subdirectory ``simOutput``**. Must be **accessable from all involved machines**. (This is not checked automatically!) If not given, a directory inside the scratch dir will be generated. Must not yet exist when invoking the runner.
- template directory (`pypicongpu_template_dir`): **Only needs to be specified if things fail.** (Specifically, *generation* fails at `pic-create`). Holds the template inside which the generated code is placed.

In summary:

directory	must exist	must not exist	created by	used by	description
scratch	yes	no	user	PyPIConGPU	holds many run dirs
setup	no	yes	generation (*pic-create*)	build (*pic-build*)	holds one simulation setup ("scenario")
run	no	yes	execution (*tbg*)	user/analysis scripts	holds results of a single simulation run
template	yes	no	PyPIConGPU source code	generation (*pic-create*)	holds predefined project template – **can usually be ignored**

48.2.2 Normal Operation

```
# sim initialized ...
from picongpu.pypicongpu.runner import Runner
r = Runner(sim, run_dir="/scratch/mydata/new_run")
r.generate()
r.build()
r.run()

import os
results_dir = os.path.join(r.run_dir, "simOutput")
analyze(results_dir)
```

Set the parameters (`setup_dir`, `scratch_dir`, `run_dir`, `pypicongpu_template_dir`) in the constructor of the Runner. All parameters are optional, pypicongpu will try to guess them for you.

Some sanity checks are performed on all given parameters, and all paths will be translated to absolute paths. For all paths only a small character set is allowed: Alphanum, `-_.` (and `/`): This is to ensure compatibility to the used tools, as they frequently have hiccups on "special" characters. The used dirs will be logged with the level info.

After the Runner has been constructed the used directories can be accessed directly:

```
r = Runner(sim, run_dir="/scratch/mydata/new_run")

# re-do some checks already performed inside the Runner
import os
assert not os.path.exists(r.run_dir)
assert not os.path.exists(r.setup_dir)

r.generate()
assert os.path.isdir(r.setup_dir)

import tempfile
if r.setup_dir.startswith(tempfile.gettempdir()):
    print("will build inside tempdir")
```

In addition to the checks in the constructor, some sanity checks are performed when invoking the individual steps. These steps are (as outlined above):

- `generate()`: translate the PICMI simulation to PIConGPU-understandable files
- `build()`: prepare the simulation for execution by calling `pic-build`
- `run()`: execute the actual simulation

Every step can only be **called once**, and exactly **in that order**.

If a step itself fails (i.e. the invoked programs return something else than 0), execution is aborted and the program output (stdout & stderr) printed.

It is completely valid (and intended design goal) that you can only run some of the steps: When `run()` (or more precisely `tbg`) can't handle your local job submission system, you might only build:

```
r = Runner(sim, setup_dir="/scratch/mysetups/setup01")
r.generate()
r.build()
# ... now manually submit the job from /scratch/mysetups/setup01
```

48.2.3 Dirty Tricks

While the Runner is designed to control **the entire lifecycle** of a simulation, you can try to feed it a simulation/setup where other steps have been performed externally.

For that **skip the checks in the constructor**: Construct the object using an empty simulation and then overwrite the generated paths:

```
from picongpu.pypicongpu.runner import Runner
from picongpu.pypicongpu.simulation import Simulation

empty_sim = Simulation()
# leave all dirs empty, this way no checks will trigger
r = Runner(empty_sim)
# now overwrite setup dir
r.setup_dir = "/other/prepared/setup"

# skip r.generate() (which would fail), directly build
r.build()
r.run()
```

While you can try that for you local setup, **it is not guaranteed to work**.

CHAPTER

FORTYNINE

SPECIES TRANSLATION

In PIConGPU (and PIC-codes in general) “Species” is a catch-all term for all things particles; it is defined rather losely and used differently in different contexts.

Generally, a “species” is a **set of macroparticles** in a simulation which are initialized from a **common parameter set**.

> A macroparticle is a representation of a varying number of simulated (“real”) particles.

For PyPIConGPU a species consists of a set **constant properties** (same for each macroparticle, “flags”) and a set of **variable properties** (attributes, exists for every macroparticle, but may have a different value). Each *attribute* is initialized by **exactly one operation**.

This results in this UML class diagram.

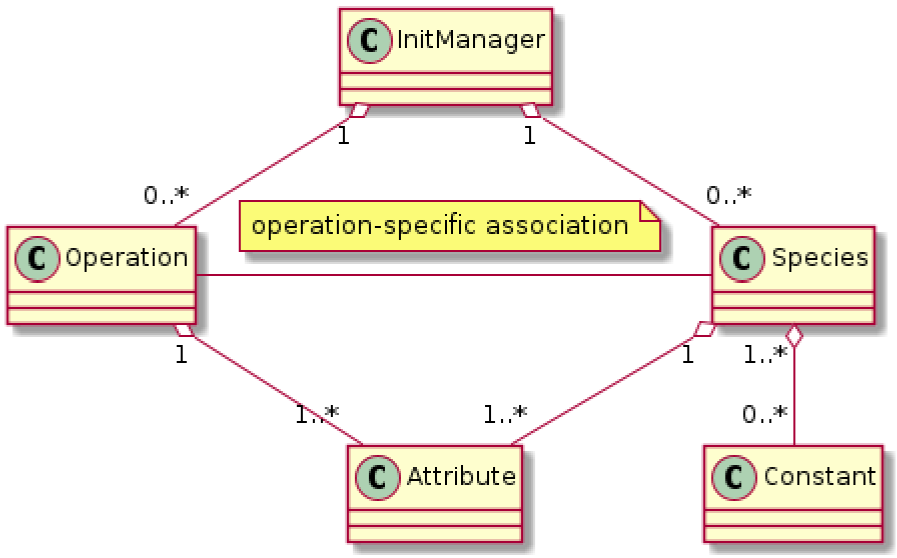


Fig. 49.1: Final relationships of species-related classes

> In particular, the name is part of this *common parameter set*, as well as the *profile* where these particles are placed. If you want to initialize the same type of particles (e.g. the same substance) with a different *profile*, a second species has to be created.

This document describes how species are translated from PICMI, via an intermediate PyPIConGPU-only representation, to executable PIConGPU `.param` files. For the general process of translation please refer to the respective documentation section. The process is outlined back-to-front here, i.e. starting with the desired PIConGPU result, deriving the PyPIConGPU representation from that and explaining some PICMI details in the end.

> Neither PICMI nor PIConGPU species declaration are introduced in detail, please familiarize yourself with both.

49.1 Code Generation

A species is directly associated to two code sections:

1. A species definition, declaring a C++ species type, placed inside `speciesDefinition.param`.
2. A species initialization, creating and initializing the attributes of the species' macroparticles, placed inside the init pipeline of `speciesInitialization.param`.

Particles of a species have different properties, which fall into one of two categories:

- **Constants** (*Flags* in PIConGPU lingo): Properties that apply to **all (macro-) particles** and can't be changed for each macroparticle
  - typically matter constants, e.g. number of protons, mass, charge
  - but also: used pusher, particle shape, etc.
  - initialized in `speciesDefinition.param`
- **Attributes**: Properties that can be different **from macroparticle to macroparticle**
  - most importantly the **position**
  - other changing/changeable properties, e.g. momentum, number of bound electrons
  - **existence** declared in `speciesDefinition.param`
  - **initialized** in `speciesInitialization.param` (actual initialization typically loaded from other files, e.g. `density.param`)

The **constants** are stored purely in the **type definition**. **Attributes** must be enabled for their respective species in their type definition, but they are set in the **init pipeline**.

A list of flags and a list of attributes is passed directly from PyPIConGPU, which makes code generation rather straightforward.

The initpipeline is expressed by a set of operations, which by definition have no interdependencies. For more details on operations please refer to the next section.

Note that while the init operations have no inherent order, they are generated in such a way that macroparticles are placed before any other attributes are set.

When reading the generated code note that while all operations are **referred to** in the initpipeline, they are typically defined in a separate file. (This is more in line with the native PIConGPU configuration.)

49.2 Architecture of PyPIConGPU Representation

The PyPIConGPU representation of species initialization uses the concepts **species**, **constant**, **attribute**, **operation** and **init manager** and defines their relationships.

- The **init manager** is the global context managing the entire lifecycle.
  - …performs all checks between objects that are not directly associated to one another
- A **species** is a type of macroparticles sharing the same properties.
  - …has a (mandatory) name
  - …has a list of *constants* and a list of *attributes*
  - …will only be considered "valid" if position and momentum *attributes* are set
  - …has at most one of each *constant* (type)
  - …has at most one of each *attribute* (type)
- A **constant** is a unmodifiable property shared among all macroparticles of a species (*flag* in PIConGPU lingo)

  - …are for example charge and mass.
  - …carries its value inside its object representation
  - …is owned by a species
  - …may depend on other species being present (during code generation)
- An **attribute** is a property which every macroparticle has, but each may have a different value
  - …are for example position and weighting
  - …object only contains a name
  - …is exclusive to a species
  - …is provided (generated) by an operation, which also sets it (inside of PIConGPU); in other words:
  - …is "in exclusive custody" of exactly one operation.
- An **operation** initializes a set of *attributes* for a set of *species*
  - …is independent from all other operations
  - …gets passed its parameters and affected species in a custom way (i.e. there is no globally unified way to pass parameters to operations)
  - …provides and sets a attributes for its species
  - …may depend on *constants*.
  - …must **NOT** depend on other attributes/operations.

This leads to some notable conclusions:

An attribute is always managed exclusively by **exactly one** operation. The purpose of this is it to make it easy to detect conflicts between operations, and at the same time making it very clear what caused the value of an attribute.

Therefore, if two attributes depend on each other, they must be set **in the same operation**. E.g., when placing ions and their electrons, both are created from **the same** `SimpleDensity` operation.

Also, if multiple factors determine an attribute, they must be specified inside of the same operation: E.g. the momentum is influenced by both temperature and drift, hence they are properties of the `SimpleMomentum` operation.

> We somewhat jokingly refer to this setup as the "grand unified theory of species".

49.3 Lifecycle

The core idea behind this architecture is to associate every **attribute** (of one species) with **exactly one operation**.

> This has two main advantages:
> (1) operations are independent of one another, and
> (2) the initialization can be traced back to exactly one operation.

To ensure this, the species initialization follows this lifecycle:

1. initialize species, constants, and operations
2. check if this setup is valid
3. operations create attributes and *prebook* them for some species
4. these *prebooked* attributes are checked, if no conflicts are found they are added to their species

For the implementation of this process, please see *`picongpu.pypicongpu.species.InitManager.bake()`*.

The generating script creates this setup:

After all checks have passed, the operations may *pre-book* the attributes they want to create. *Pre-Booking* is performed in *`picongpu.pypicongpu.species.operation.Operation.prebook_species_attributes()`* by

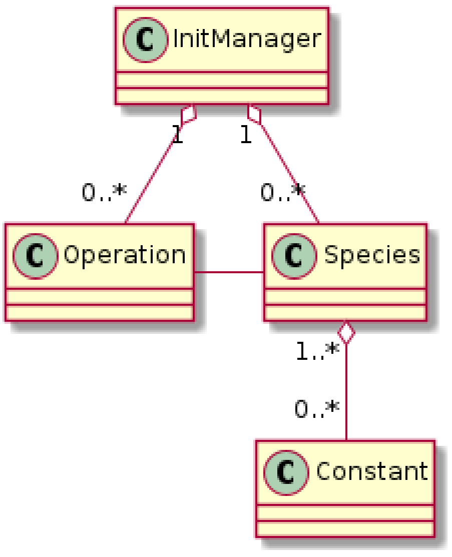


Fig. 49.2: Setup as created by PyPIConGPU (before Operations have been applied)

creating the intended Attribute objects (one per species) and storing the intended association into the `Operation.attributes_by_species`.

Pre-Booking results in this intermediate connection:

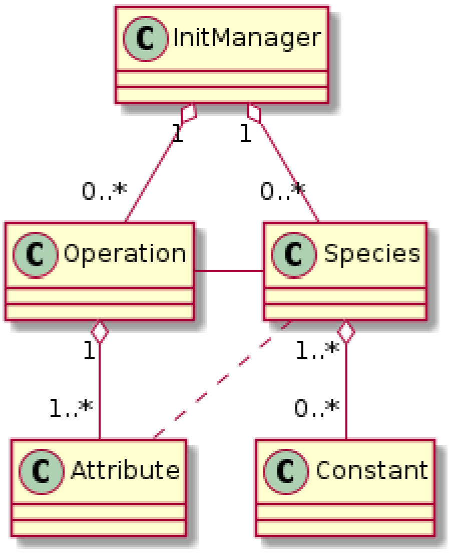


Fig. 49.3: classes with prebooked association

After these pre-booked species-attribute connections have been checked, they are *baked*, i.e. applied into the final model:

From these classes the code generation can simply derive:

- a list of species, and for each species
  - its constants
  - required attributes
- a list of operations

  Note that the some associations from the model above are lost when translating to JSON for code generation, e.g. which operation initializes which attribute.

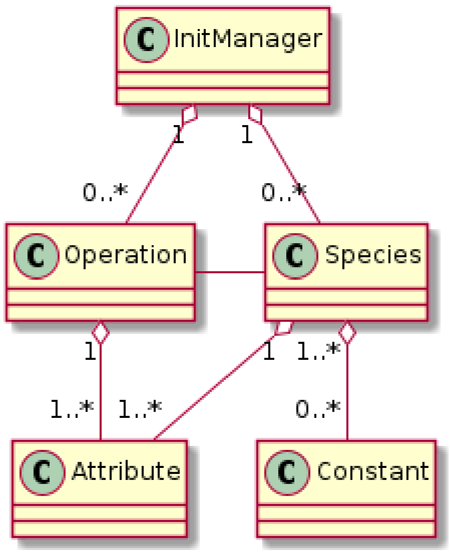


Fig. 49.4: final (*baked*) species-related classes

49.4 Checks

Constants, Attributes, and Operations may have various dependencies. **All** possible dependencies are listed here:

- Species may depend on:
  - Nothing: Species are passive, and are manipulated by **multiple** Operations.
  - As a sanity check species require *Position* and *Momentum* attributes (as hard-coded checks). This is not because the model requires it, but because species without these make no sense in PIC implementations.
- Attributes may depend on:
  - Nothing: Attributes are passive, and are manipulated by **exactly one** Operation.
- Constants may depend on:
  - (not-finalized) **Species** objects: E.g. ions may depend on their electrons. These dependencies modify the order in which the species are rendered to code.
  - (existence of) **Attribute** types: E.g. ionizers depend on the attribute `BoundElectrons`. The Operation which sets the Attribute is not visible to the check.
  - (existence of) **Constant** types: E.g. ionizers depend on the constant `ElementProperties` (which contains the ionization levels). The value of these constants is not visible to the check.
- Operations may depend on:
  - their **Species**: Note that at this point the Species do not have any Attributes added. (Accessing the Attributes will result in an error.)
  - (through their Species) these Species' **Constants**: These are the Attribute objects, i.e. their value can be read (and checked against).

Other dependencies are forbidden by design. Notably, this includes Operation-Operation and Operation-Attribute dependencies, because Operations are independent from one another.

Note: Errorneous (circular) dependencies are checked by the **init manager**.

All checks are invoked when applying the Operations in the init manager's `bake()` method (see also: *Lifecycle*). (For their implementation details please refer to the respective base classes.)

49.5 Constants

PyPIConGPU `Constant` objects follow the concept of a *constant property*, instead of mirroring the PIConGPU *Species Flags* 1-to-1.

E.g., the Constant `ElementProperties` is translated to atomic numbers and ionization energies.

(Note: Mass and Charge are treated as separate constants, even if they are element properties too.)

Constants may depend on other species, which is required due to the C++ syntax: Forward declarations (referring to to-be-defined *things*, in this case species) are not possible for this case (template programming), so the order of the code generation matters. This order is governed by the Constant-Species dependencies and handled by the init manager.

49.6 Attributes

Attributes are enabled by adding them to the type definition in PIConGPU. Their values are set in the init pipeline.

The PyPIConGPU species model (see: *Architecture of PyPIConGPU Representation*) ties *Attributes* strongly to *Operations*: An Attribute is always initialized (handled, created) by **exactly one Operation**. Hence, Attributes are **never created directly**, but only ever through Operations (see: *Lifecycle*).

49.7 Operations

Operations create and initialize *Attributes* **exclusively**, i.e. one Attribute of a Species can only ever be initialized by **exactly one Operation**.

There is no unified way in which Operations must accept their arguments, i.e. an Operation must define its own interface how Species and arguments must be passed.

Operations are the intended way how dependencies between Attribute values should be expressed: **If two Attributes are related, they must be initialized by the same Operation.** For that, the Operation may depend on Constants of these Species (in `check_preconditions()`), but it **must NOT modify** these Constants.

During the code generation these Operations may result in any number of "Functors" in the init pipeline.

The init manager translates the Operations to JSON, creating a dictionary

```
{
    "op_type1": [ "OP1", "OP2" ],
    "op_type2": [ "OP3" ],
    "op_type3": []
}
```

For this every type of Operation must be registered with a name in `_get_serialized()` in the init manager (and the respective schema).

Conceptually Operations are **independent from other Operations**. However, for code generation an inherent order is required, which implicitly introduces an order between these Operations. Yet, this order is only required for placing macroparticles (particles must be created before they can be modified) – for all other cases it is ignored.

The concept of Operations aims to move the species initialization closer to a *declarative* description ("momentum **is** x"), instead of PIConGPU's *procedural* description ("**set** momentum to x").

Note: The purpose of this is to enable traceability and thereby error handling. A *procedural* description is very powerful, but quickly runs into the Halting Problem, rendering any programmatic handling (besides execution) **impossible**. (This includes any manipulation, checking or predicting results.)

In many cases PIConGPU's *functors* from the init pipeline can be translated to a single operation, however this is not the case for all examples:

- setting the number of bound electrons for ions
  - Attributes: `boundElectrons`
  - PIConGPU: set bound electrons functor (implementation of `Free` functor)
  - PyPIConGPU: `BoundElectrons` Operation
- not placing a species itself (e.g. electron species)
  - Attributes: `position<position_pic>`, `weighting`
  - PIConGPU: leave init pipeline empty, but add `position<position_pic>` Attribute
  - PyPIConGPU: `NotPlaced` Operation
- setting the momentum, composed of *drift* and *temperature*
  - Attributes: `momentum`
  - PIConGPU: **set** *drift*, then **add** *temperature* in second operation
  - PyPIConGPU: `SimpleMomentum` Operation (which accepts **both** drift **and** temperature)
- placing multiple species together (see also *next section*)
  - Attributes: `position<position_pic>`, `weighting`
  - PIConGPU: `CreateDensity` for first species, then `Derive` all other species
  - PyPIConGPU: `SimpleDensity` Operation, which accepts a list of Species

49.8 Placement

The *Placement* describes the **creation** of macroparticles. (Which differs from all other Operations, because they only modify existing macroparticles.)

Ultimately, the particle placement expresses where macroparticles with which weighting (number of represented real particles) should be placed.

For this, the following components may be specified:
PICMI term (PIConGPU term)

- particle distribution (density profile): specifies density by location
  - function w.r.t. x, y, z of global coordinate system
  - typically expressed relative to base density
- density scale (density ratio): modifies density
  - scalar factor, multiplied with particle distribution
- particle layout (particle position): specifies position and number of macro particles inside the cell
  - typically random or fixed to grid positions

Note that there are two steps in deriving the placement of particles from the density. The density, i.e. number of (real) particles per volume is given by a particle distribution (density profile). This profile is multiplied with the density scale (density ratio) and then

1. discretized, i.e. for every cell the average density/the number of particles is computed
2. Based on the particle layout (particle position) a certain number of macroparticles are placed at a certain position *inside every cell*.
3. The weighting of these macroparticles is adjusted to match the computed density. If the weighting is below a configurable threshold, particles will be deleted.

This document uses the following terms:

used here	PIConGPU	PICMI	description
profile	density profile	particle distribution	description of number of particles per unit of volume
ratio	density ratio	density scale	scalar factor applied to profile (modifies the number of particles represented by one macroparticle)
layout	particle position	particle layout	positioning of macroparticles within a cell

The term "placement" refers to the entire process of placing macroparticles from parameters to final positioning.

49.8.1 Initializing Multiple Species

Some species are to be created together, e.g. ions and their respective electrons. Though, in general, species are treated separately during initialization.

> The main purpose of this grouping is to join particles in such a way, that in total their charge is (locally) neutral. See: *Quasi-Neutral Initialization*

For the purposes of this document, species might be related as follows:

- If two (or more) species are related in such a way that their purpose is to bring the total (local) charge to a neutral level, they are to be *derived* from each other.
- If the species are not related in such a way, they are to be *treated separately*.

PICMI initializes all species equally, i.e. simply stores a list of all species to be initialized: The distinction above is not made explictly: For PICMI, species that are to be *derived* from each other simply end up at the same place (and can thus neutralize each other) after applying all initialization of values & position (profile, layout, ratio).

PIConGPU on the other hand uses a different approach. There, particles that are to be *derived* from each other are initialized in order: Only the first species (of the group to be *derived*) is placed by a placing algorithm, all other *derived* species have their position simply copy-pasted from this first. (If necessary the scalar *density scale* is adjusted.)

> PIConGPU here is more efficient than (naive) PICMI, because instead of placing all particles s.t. the positions are equal for two (or more) species (in which case they are *derived*), only the initial positions are calculated – and all *derived* species' positions are **made** to be equal.

This schema of separate initialization, yet same final positioning **can not be recreated in PIConGPU**. The problem lies in the random layout: When calling the **same random layout** in PIConGPU twice, **the results do not match**. Therefore, particles that are to be grouped together **are derived from each other in PIConGPU-native setups**.

> There are very good reasons of the implementation to do that: (1) It is more efficient to copy positions instead of generating them twice. (2) Due to the parallel nature of PIConGPU, a random placement is **not reproducable**. (Not reasonably in practical scenarios at least: In particular, it is very difficult to keep the RNG state in sync==reproducable across many processes, across many different algorithms, and across all supported architectures.)

The implicit grouping in PICMI has to be made explicit before translating the particle initialization to PIConGPU. For that we consider the following cases to decide if two species are to be *derived* or *treated separately*, depending if their initialization parameters (profile, layout, ratio) are different or equal.

profile	layout	ratio	behavior
different	different	different	treat separately
different	different	equal	treat separately
different	equal	different	treat separately
different	equal	equal	treat separately
equal	different	different	treat separately
equal	different	equal	treat separately
equal	equal	different	derive
equal	equal	equal	derive

This grouping is performed **during the translation from PICMI to PyPIConGPU**. Profiles/Layouts are considered *equal* if they are == equal. For PICMI, == equality is the same to `is` (object identity), i.e. two species have the same profile (particle distribution) if they use **the same** particle distribution **object**. (PICMI may redefine == in the future.)

In PyPIConGPU Species that are to be derived are grouped into the same `SimpleDensity` Operation.

Ratios are stored in Constants (PIConGPU species flags), and the species with the **lowest ratio** is placed first to make sure the minimum weighting is always respected. Note that the first species is not required to use a ratio of one, which is handled properly by PIConGPU (both during placement and copying).

49.9 Notes

49.9.1 Seeds

Seeds are not supported and entirely ignored by PyPIConGPU.

49.9.2 Use Ratios over Profiles!

The density is defined by a profile multiplied by a ratio. Hence, in theory the following two are equal:

- Ratio 1, Profile `f(x,y,z) = 6e24 + x * 2e23`
- Ratio 2, Profile `f(x,y,z) = 3e24 + x * 1e23`

However, this is to be discouraged:
Using a new profile (1) has to be computed for every cell and (2) disables the option to detect species placed together.
Reusing an existing profile (1) is more efficient (reuses previous results) and (2) allows to detect species placed together.

49.9.3 Ratio Handling

Ratios are scalar factors applied to profiles.

Profiles are expressed that they are valid with a ratio of one. By specifying a different PIConGPU density ratio (in the species flags) the PIConGPU `CreateDensity` functor in the init pipeline will adjust the weighting accordingly.

The subsequent `Derive` functors to copy the species location respect the ratios of the respective species (specifically: the ratio of their ratios), so no additional measures are required.

49.9.4 Electron Resolution

PIConGPU requires an explicit species to be used for electrons, whereas this is not possible to specify with PICMI.

The user can either explicitly set the electron species by providing `picongpu_ionization_electrons` to the PICMI species, or the PICMI translation will try to guess the used electron species. If there is exactly one electron species it will be used, if there is no electron species one will be added, and if there are two or more electron species an error will be raised.

49.10 Example

This example shows the initialization of a constant density of Helium with one bound electron. For the sake of the example the ratio is 0.5. The PICMI initialization is:

49.10.1 PICMI

```
uniform_dist = picmi.UniformDistribution(density=8e24,
                                         rms_velocity=[1e5, 1e5, 1e5])
species_helium = picmi.Species(name="helium",
                               particle_type="He",
                               charge_state=1,
                               density_scale=0.4,
                               initial_distribution=uniform_dist)

# init of sim omitted

random_layout = picmi.PseudoRandomLayout(n_macroparticles_per_cell=2)
sim.add_species(species_helium, random_layout)
```

For which an additional electron species is added, equivalent to:

```
electrons = picmi.Species(name="e",
                          particle_type="electron")
sim.add_species(electrons, None)
```

49.10.2 PyPIConGPU

This translated to these PyPIConGPU objects (simplified, some helper objects are omitted):

Warning: This is a UML object diagram, not a (generally more common) class diagram as above. (Boxes here are objects instead of classes.)

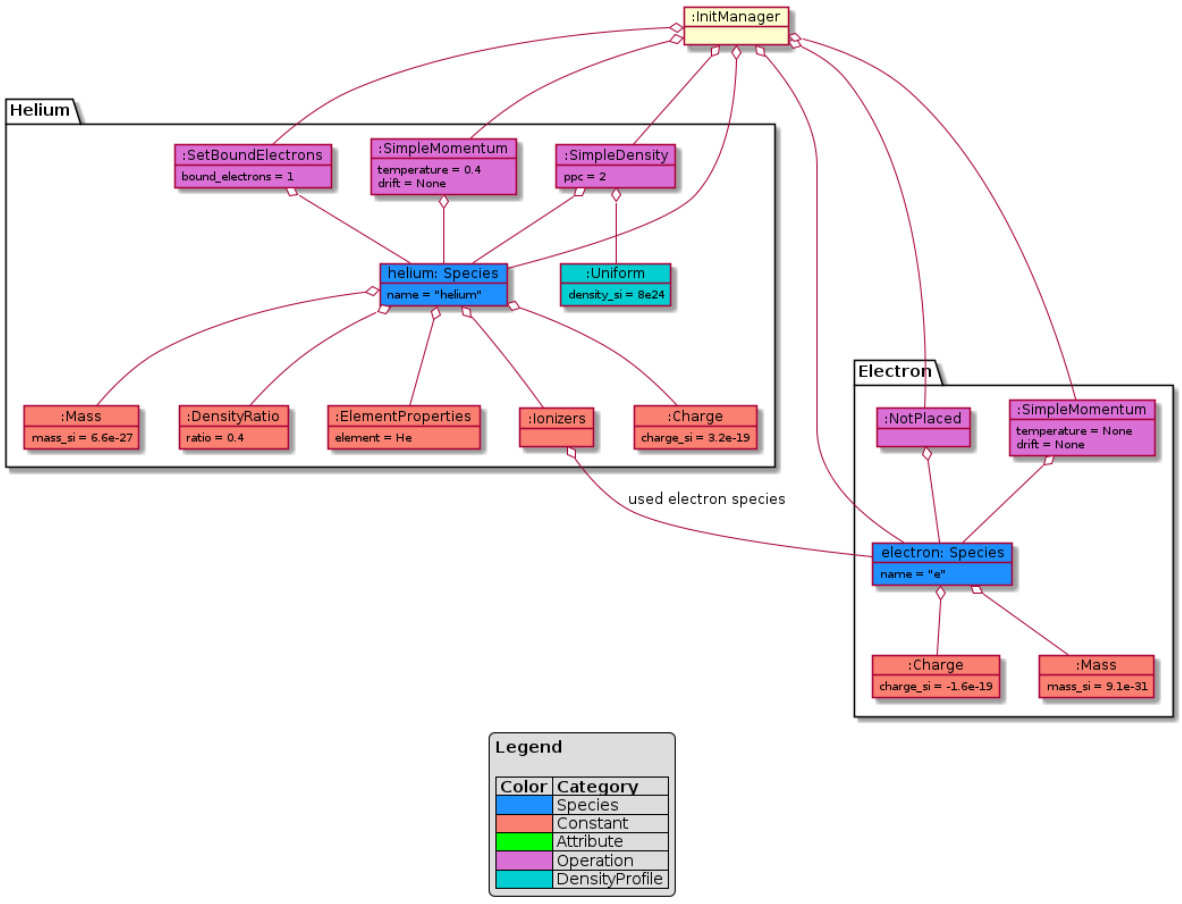


Fig. 49.5: example setup as translated by PICMI before Baking

After attribute creation, prebooking, checking and finally baking, this results in these objects:

49.10.3 Rendering JSON Representation

From this, the following (abbreviated) JSON representation is derived:

> ALL CAPS strings are placeholders for full (complex) objects

```
{
    "species": [
        {
            "name": "e",
            "typename": "species_e",
            "attributes": [
                {"picongpu_name": "position<position_pic>"},
                {"picongpu_name": "weighting"},
                {"picongpu_name": "momentum"}
```

(continues on next page)

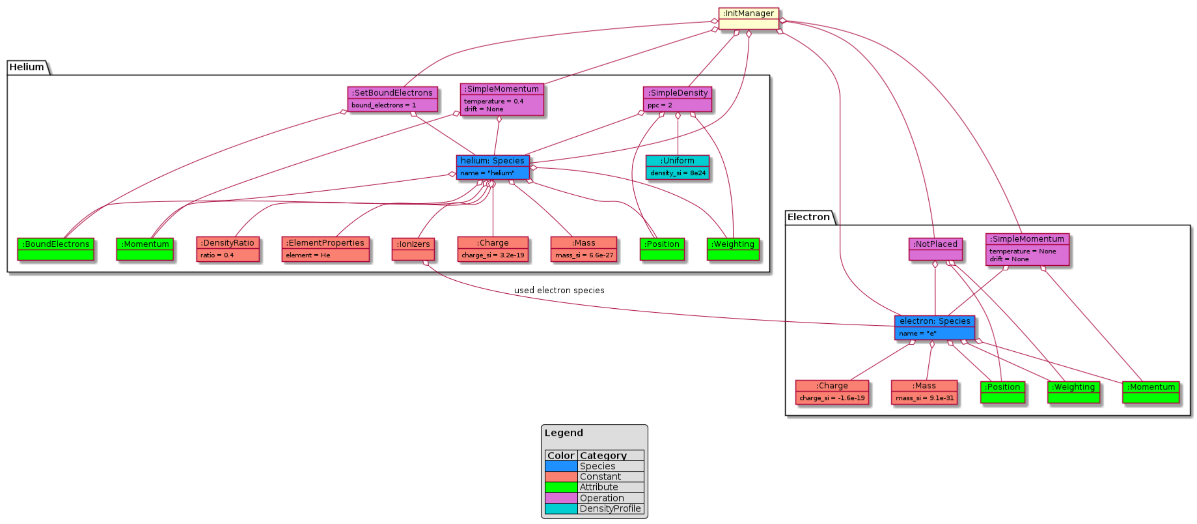


Fig. 49.6: example setup after baking attributes

(continued from previous page)

```
            ],
            "constants": {
                "mass": {"mass_si": 9.1093837015e-31},
                "charge": {"charge_si": -1.602176634e-19},
                "density_ratio": null,
                "ionizers": null,
                "element_properties": null
            }
        },
        {
            "name": "helium",
            "typename": "species_helium",
            "attributes": [
                {"picongpu_name": "position<position_pic>"},
                {"picongpu_name": "weighting"},
                {"picongpu_name": "momentum"},
                {"picongpu_name": "boundElectrons"}
            ],
            "constants": {
                "mass": {"mass_si": 6.64647366797316e-27},
                "charge": {"charge_si": 3.204353268e-19},
                "density_ratio": {"ratio": 0.4},
                "ionizers": {"electron_species": "ELECTRON SPECIES OBJECT"},
                "element_properties": {
                    "element": {
                        "symbol": "He",
                        "picongpu_name": "Helium"
                    }
                }
            }
        }
    ],
    "operations": {
        "simple_density": ["SIMPLE DENSITY OBJECT"],
        "simple_momentum": [
            {
```

(continues on next page)

(continued from previous page)

```
                "species": "HELIUM SPECIES OBJECT",
                "temperature": {"temperature_kev": 0.41484025711818995},
                "drift": null
            },
            {
                "species": "ELECTRON SPECIES OBJECT",
                "temperature": null,
                "drift": null
            }
        ],
        "set_bound_electrons": [
            {
                "species": "HELIUM SPECIES OBJECT",
                "bound_electrons": 1
            }
        ]
    }
}
```

49.10.4 Generated Code

Ultimately rendering this code (reformatted for reading):

speciesDefinition.param:

```
value_identifier(float_X, MassRatio_species_e,
                9.1093837015000008e-31 / SI::BASE_MASS_SI);
value_identifier(float_X, ChargeRatio_species_e,
                -1.6021766339999999e-19 / SI::BASE_CHARGE_SI);

using ParticleFlags_species_e = MakeSeq_t<
    massRatio<MassRatio_species_e>, chargeRatio<ChargeRatio_species_e>,

    particlePusher<UsedParticlePusher>, shape<UsedParticleShape>,
    interpolation<UsedField2Particle>, current<UsedParticleCurrentSolver>>;

using ParticleAttributes_species_e =
    MakeSeq_t<position<position_pic>, weighting, momentum>;

using species_e = Particles<PMACC_CSTRING("e"), ParticleFlags_species_e,
                            ParticleAttributes_species_e>;


value_identifier(float_X, MassRatio_species_helium,
                6.6464736679731602e-27 / SI::BASE_MASS_SI);
value_identifier(float_X, ChargeRatio_species_helium,
                3.2043532679999998e-19 / SI::BASE_CHARGE_SI);
value_identifier(float_X, DensityRatio_species_helium, 0.40000000000000002);

using ParticleFlags_species_helium = MakeSeq_t<
    massRatio<MassRatio_species_helium>,
    chargeRatio<ChargeRatio_species_helium>,
    densityRatio<DensityRatio_species_helium>,

    ionizers<MakeSeq_t<particles::ionization::BSI<
```

(continues on next page)

(continued from previous page)

```
                        species_e, particles::ionization::current::None>,
                    particles::ionization::ADKCircPol<
                        species_e, particles::ionization::current::None>,
                    particles::ionization::ThomasFermi<species_e>>>,

    atomicNumbers<ionization::atomicNumbers::Helium_t>,
    ionizationEnergies<ionization::energies::AU::Helium_t>,

    particlePusher<UsedParticlePusher>, shape<UsedParticleShape>,
    interpolation<UsedField2Particle>, current<UsedParticleCurrentSolver>>;

using ParticleAttributes_species_helium =
    MakeSeq_t<position<position_pic>, weighting, momentum, boundElectrons>;

using species_helium =
    Particles<PMACC_CSTRING("helium"), ParticleFlags_species_helium,
            ParticleAttributes_species_helium>;

using VectorAllSpecies = MakeSeq_t<species_e, species_helium>;
```

The operations result in this `speciesInitialization.param`:

```
using InitPipeline = boost::mp11::mp_list<
    /*************************************/
    /* phase 1: create macroparticles */
    /*************************************/

    CreateDensity<densityProfiles::pypicongpu::init_species_helium,
                  startPosition::pypicongpu::init_species_helium,
                  species_helium>,

    /************************************************/
    /* phase 2: adjust other attributes (if any) */
    /************************************************/

    // *adds* temperature, does *NOT* overwrite
    Manipulate<manipulators::pypicongpu::AddTemperature_species_helium,
               species_helium>,

    Manipulate<manipulators::pypicongpu::PreIonize_species_helium,
               species_helium>,

    // does nothing -- exists to catch trailing comma left by code generation
    pypicongpu::nop>;
```

Utilizing this density definition from `density.param`:

```
/**
* generate the initial macroparticle position for species "helium"
* (species_helium)
*/
struct init_species_helium_functor {
    HDINLINE float_X operator()(const floatD_64 &position_SI,
                                const float3_64 &cellSize_SI) {
        return 7.9999999999999999e+24 / SI::BASE_DENSITY_SI;
    }
};
```

(continues on next page)

(continued from previous page)

```

using init_species_helium =
    FreeFormulaImpl<init_species_helium_functor>;
```

CHAPTER

FIFTY

MISC

This is a catch-all section for general notes on PyPIConGPU.

50.1 Python Concepts

Some common concepts recurr throughout the code and are not explained in detail when used. This aims to give a non-exhaustive list of such patterns.

They are as close to "standard python" as possible, so using a search engine/asking a python expert should also help you understand them.

50.1.1 Type Checking with Typeguard

Use as defined in PEP 484 "Type Hints". Briefly:

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

Note that these type annotations are **not checked by python** on their own.

Typechecks are enabled through typeguard, mostly using the annotation `@typechecked` for classes.

This does not check attribute type annotations, see next section.

50.1.2 Typesafe Class Attributes

Class attributes can use type annotations according to PEP 484, but this is not enforced by typeguard.

The solution is to use properties.

In the code you will see:

```
class SomeObject:
    attr = util.build_typesafe_property(int)
```

Now the following hold:

1. Accessing `attr` before it has been set will **raise an exception**
2. Assigning `attr` anything other than an `int` will raise a `TypeError`

   Note: instead of types you can use specifications provided by `import typing`, e.g. `typing.Optional[int]`. Optional vars still have **no default**, so you have to set them to `None` explicitly if you don't want to provide them.

Internally the declaration above is expanded to the equivalent of:

```
@typechecked
def getter(self) -> int:
    if not hasattr(self, "actual_name_for_var__attr"):
        raise AttributeError("variable is not initialized")
    return getattr(self, "actual_name_for_var__attr")

@typechecked
def setter(self, value: int) -> None:
    setattr(self, "actual_name_for_var__attr", value)

class SomeObject:
    attr = property(getter, setter)
```

50.1.3 Map-Filter-Reduce

`map()`, `filter()`, and `reduce()` are "higher-order functions" and a basis of the functional programming paradigm (as opposed to loops for iterative processing). You can find an an introduction here (but any other intro is probably fine too).

Boils down to:

```
[2, 4, 6] == list(map(lambda x: 2*x,
                      [1, 2, 3]))
```

`map()` does not return a list directly and must be cast again. (Python, why?)

`map()` does not work on dictionaries, to for this the dictionary is cast to a set of key-value tuples:

```
{"a": 3, "b": 0} == dict(map(
    lambda kv_pair: (kv_pair[0], len(kv_pair[1])),
    {"a": [0, 0, 0], "b": []}.items()))
```

Note: Do not feel obliged to follow this pattern. It is commonly used, because it allows a concise notation, yet it is very much **not** mandatory.

50.2 Tool Support

This is a short list of tools to aid you with development.

50.2.1 Formatting

PEP 8 (formatting guidelines) compliance:

run from repo root `flake8 lib/python/ test/python/`

Note: Please obey the line length from PEP 8 even if that is annoying at times, this makes viewing files side-by-side much simpler.

50.2.2 Run Tests

go into `test/python/picongpu`, execute `python -m quick` (for tests with compilation `python -m compiling`)

50.2.3 Test Coverage

0. install coverage.py: `pip install coverage`
1. go to tests: `cd test/python/picongpu`
2. execute tests, record coverage: `coverage run --branch -m quick`
3. view reports
   - for PyPIConGPU: `find ../../../lib/python/picongpu/pypicongpu/ -name '*.py' | xargs coverage report -m`
   - for PICMI: `find ../../../lib/python/picongpu/picmi/ -name '*.py' | xargs coverage report -m`
   - Goal is 100% coverage (missing sections are printed)

For further options see the coverage.py doc.

50.3 API Documentation

Document the API using docstrings, which will be rendered by Sphinx AutoAPI automatically. The result is available from the TOC, although the name is generated to be the python module name: *picongpu.pypicongpu*.

The configuration is placed in Sphinx's `conf.py`, all classes are traversed by using `__all__` defined in **every** `__init__.py`.

Additional checks (completeness etc.) are **NOT** employed (and not even possible). So pay attention that you document everything you need.

As everything is passed to Sphinx, **docstring should be valid** reStructuredText. (reStructuredText quick guide from Sphinx)

You may use Sphinx-exclusive directives too, e.g. ``:ref:`MARK` ``.

To document parameters, return values etc. make them compatible with Sphinx's autodoc. For details please refer to the respective documentation section.

Note: If there is an reStructuredText syntax error during generation (printed as warning after sphinx invocation `make html`) to debug:

1. visit the respective page
2. click *View page source* at the top of the page

   (This might be only availble **locally**, the online version displays *Edit on GitHub* instead)

An example can be found in the official documentation. The types are implictly given by the type hints, so these can be omitted:

```python
from typeguard import typechecked
import typing


@typechecked
```

(continues on next page)

(continued from previous page)

```python
def my_func(a_char: str, num: int) -> typing.List[str]:
    """
    build list with "triangle" of chars

    Accepts a single char and builds a list where the n-th element contains a
    string consisting of n times the given char.
    The list has the given total length.

    This is an example for a section of the PyPIConGPU doc,
    see: :ref:`pypicongpu-misc-apidoc`.
    to show that all Sphinx directives are valid.
    Because this is reStructuredText, (nested) lists require separate
    paragraphs:

    - an item
    - another item

        - nested 1
        - nested 2

    - continuation of top level

    :param a_char: character to use, e.g. "."
    :param num: max, e.g. "3"
    :raises AssertionError: on a_char being not a single character
    :raises AssertionError: if num < 0
    :return: [".", "..", "..."]
    """
    assert 1 == len(a_char)
    assert 0 <= num

    if 0 == num:
        return []

    return my_func(a_char, num - 1) + [num * a_char]
```

Note that `:raises TypeError:` is omitted too, because all functions are checked for types anyways.

This produces the following code section:

`doc_example.`**`my_func`**`(`*a_char: str*, *num: int*`)` → List[str]

build list with "triangle" of chars

Accepts a single char and builds a list where the n-th element contains a string consisting of n times the given char. The list has the given total length.

This is an example for a section of the PyPIConGPU doc, see: *API Documentation*. to show that all Sphinx directives are valid. Because this is reStructuredText, (nested) lists require separate paragraphs:

- an item
- another item

  - nested 1

  - nested 2

- continuation of top level

Parameters

- **a_char** – character to use, e.g. "."
- **num** – max, e.g. "3"

Raises

- `AssertionError` – on a_char being not a single character
- `AssertionError` – if num < 0

Returns

[“.”, “..”, “. . . ”]

CHAPTER

FIFTYONE

FAQ

This file contains frequently asked questions.

> This is more practical compared to *the misc* documentation section (which discussed fundamental and/or interesting concepts). There are no rules for this document, please add much stuff.

To learn more about the context of this FAQ please read the *translation* documentation section.

51.1 What are PICMI and PyPIConGPU?

PICMI is defined upstream and is a common interface for multiple PIC implementations.

PyPIConGPU is a Python module which generates `.param` and `.cfg` files for PIConGPU.

PICMI is the user interface, and its data structures (objects) are translated to PyPIConGPU data structures (objects), from which ultimately `.param` and `.cfg` files are generated.

As a typical user you will only ever interact with PICMI, PyPIConGPU is **purely internal**.

PICMI is very lax in its checks, whereas PyPIConGPU is much stricter and more "trigger-happy" with errors and exceptions.

51.2 What does "Testing" mean?

> Please also see *the respective documentation section*.

To check if a program works by trying some examples on it and comparing the actual to expected results. Note however, that this does not *verfiy* the correctness in a formal way.

We try to follow "Test Driven Development", i.e. writing the test before the implementation. This has a couple of advantages:

- coverage is kept high
- "regressions" (a feature stops working) can be caught quickly
- the programmer is forced to use their code
- the programmer must think about their problem **both** in terms of concrete examples **and** an abstract algorithm
- the tests serve as an example of how the implementation can be used

51.3 What is tested?

The pipeline from PICMI (user input) to the generated JSON representation.

If the JSON representation is used correctly is not tested.

> As of the time of writing, this is b/c there is no nice and simple framework to check simulation results (in a concise way). If there is one when you read this, please add tests checking the compilation too.

51.4 What are the different types of tests?

We differentiate by **quick** and **compiling** tests.

All **quick** tests run in a matter seconds, because they do not compile anything. The **compiling** tests actually invoke `pic-build` and hence take quite a while to all run.

Unit and *Integration* tests are not separated on a structural basis, both are in the **quick** tests.

51.5 How to execute tests?

```
cd $PICSRC/test/python/picongpu
python -m quick
python -m compiling
```

51.6 Errors: What types of errors exist and when are they used?

Please refer to the Python documentation for available exceptions.

PICMI typically uses `AssertionError`, in PyPIConGPU `TypeError` and `ValueError` are the most common.

51.7 In which order should one write tests?

Technically speaking this does not matter, however the recommended order is:

1. PyPIConGPU
2. PyPIConGPU translation to JSON (by calling `get_rendering_contex()`, which automatically includes schema check)
3. PICMI to PyPIConGPU translation
4. Compilation (in `compiling` tests)

51.8 What does it mean to test/use `get_rendering_context()`?

The method `get_rendering_context()` is used to retrieve a JSON representation.

It is generated inside of `_get_serialized()`, and then the schema is checked. (See also: *What is a JSON schema?*)

I.e.: `get_rendering_context()` = `_get_serialized()` + **schema check**

When you encounter it in tests it is typically used to ensure that the translation to JSON works.

51.9 What does "Variable not initialized" mean, and when does it occur?

When accessing a property that has been created by `util.build_typesafe_property()`, an error is thrown if this property is not set yet.

Note that even something as innocent as this can trigger the message:

```
if 0 == len(self.my_list):
    pass
```

So pay attention to the stack trace and the line numbers.

51.10 How to install requirements?

From the repository root execute:

```
pip install -r requirements.txt
```

51.11 When should a `picongpu_` prefix be used for variable names?

Inside of PICMI prefix everything PIConGPU-specific with `picongpu_`. In PyPIConGPU should **not** be used.

51.12 What is a JSON schema?

A JSON schema describes how a JSON document may look.

It is used to ensure that PyPIConGPU output is stable: The templates used for code generation rely on that schema being held.

See the full spec for an in-depth explanation.

CHAPTER

FIFTYTWO

PYPICONGPU HOWTOS

For particular tasks HowTos are collected here. Please feel free to contribute!

52.1 How to Add a Species Constant ("Flag")

Species constants are constants associated to all particles of a species. Hence they are included in the PIConGPU species type definition, located in the "Species Flags".

Replace `CONST` with your new constant.

1. add constant in `lib/python/picongpu/pypicongpu/species/constant/CONST.py`
   - inherit from `Constant`
2. add test, checking **at least** for:
   - rendering from `get_rendering_context()`
   - check (even if check is empty)
   - typesafety
   - (possibly empty) dependencies on species, attributes, other constants
3. (implement tests, add schema)
4. add constant passthrough test in species test
   - add constant in map `expected_const_by_name` in test `test_rendering_constants()`
   - background: constants are not automatically exposed by the PyPIConGPU species class
5. (implement test by adding passthrough from species.py)
6. write PICMI test
   - new constant is created from PICMI
7. (implement test)
8. adjust code generation
   - keep in mind that your new constant is optional, i.e. use

     ```
     {{#constants.CONST}}
        my-code-to-generate-CONST = {{{value}}};
     {{/constants.CONST}}
     ```
9. create & compile an example

Note that constants are not stored in a list, but every constant type has a key in a dictionary associated to it. This means **when adding a new constant** the **species class** has to be adjusted to pass the new constant into its rendering context.

52.2 How to Write a Schema

52.2.1 General Process

1. Write test for `get_rendering_context()`
    - checks passthru of values
    - (implictly checks existance and against schema)
2. Run test
    - must report NameError
3. Make checked object **inherit** from `RenderedObject`
    - located at: `pypicongpu.rendering.RenderedObject`
    - (use relative imports!)
4. Implement `_get_serialized(self) -> dict`
    - if using TDD: `return {}`
5. Run tests
    - must report `RuntimeError` with message `schema not found`
6. Add schema (see below)
7. Run tests, must report validation error
8. Implement content of `_get_serialized()`

52.2.2 Schema Writing Checklist

All schemas must:

- top-level object contains the following keys (properties):
    - `$id`: URI beginning with `https://registry.hzdr.de/crp/picongpu/schema/` + *fully qualified name* (FQN, see below)
    - `description`: Human-Readable description (avoid mentioning PyPIConGPU internals)
    - `type`: `object`
    - **also** has keys of next point (b/c is object)
- every `type:  object` has
    - `unevaluatedProperties`: `false`, forbids additional properties
    - `required`: list of all properties defined, makes all properties mandatory
- all properties are mandatory (in `required`)
    - exception: some other property indicates if said property is present or missing (see below)
- every field has a `description` (unless pure reference `$ref`)
- **only one schema per file**, i.e. only one URI
    - required for schema database loading
- filename ends in `.json`
    - located in `share/pypicongpu/schema`
    - (no structure is enforce, typically `FILE.CLASS.json` in respective subdirectory)
- references use absolute URIs (start with `https://`)

- When dealing with optional properties use this pattern:

```
{
  "anyOf": [
    { "type": "null" },
    { "$ref": "URI" }
  ]
}
```

  Rationale: As of the time of writing there is a bug in the `jsonschema` python lib when using `oneOf`, this is a workaround.

  These are not enforced. Some of the points raise warnings if not fullfilled, but not all of them.

52.2.3 Schema Writing Background

- Used version: Draft 2020-12
- JSON schema main page
  - **recommended** (includes many practical examples): Full Reference | print version
  - Learn JSON Schema | tutorial | examples
- Online live validator
- URIs are used purely for **identification**, i.e. they don't have to resolve to anything meaningful
  - Avoid collisions at all costs
  - URI vs URL: uniform resource *identifier* vs. *locator* – here we identify, hence URI
  - PyPIConGPU URIs: Base URI + classname suffix
  - base uri `https://registry.hzdr.de/crp/picongpu/schema/`
  - suffix: *FQN* (fully qualified name) – full module+class path, let python generate it
- refer to schemas using `$ref` key
  - recommended section from manual
- only use absoulte URIs
  - (technically speaking relative references would work, but they are prone to introduce headaches)

52.2.4 Polymorphy and Schemas

- Polymorphy: Concept of Object Oriented Programming, where *the same name* can refer to *different implementations*.
  - E.g.: all operations inherit the same `check_preconditions()` method, but each implementation is differnt
  - E.g.: All density profiles inherit from `DensityProfile`, but have differnt implementations
  - (not polymorphy: multiple species objects – not polymorphy b/c they use the same implementation)
- Translating every object to a JSON version results in different possible versions
  - e.g.: valid profile could be `{density_si: 17}` as well as `{expression: "x+y"}`
  - solution: explicitly state type
  - problem: for templating engine, type must be encoded in **key** (Mustache does **NOT** access values)
    * workaround: one key per type

  - problem: templating engine complains on undefined keys (i.e. checks like "if the key exists do this" produce warnings and **must be avoided**)
    * workaround: always define all keys which are checked against (these checks indicate if other, optional, keys are available)
    * (note: fewer optionals make schema definition easier)

- possible solutions:
  1. externally specify types (e.g. as for density profiles)

```
{
  "type": {
    "uniform": false,
    "gaussian": true,
    "analytic": false
  },
  "data": { }
}
```

  2. one key per type (e.g. as for operations)

```
{
  "simple_density": [ {"name": "OP1"}, {"name": "OP2"} ],
  "simple_momentum": [ {"name": "OP3"} ]
}
```

- problem: need to provide wrapper which generates these structures
  - do **NOT** change the rendering interface, i.e. `_get_serialized()` should **ONLY** provide the data (not the type)
  - DO: add extra wrapper method (e.g. `get_generic_profile_rendering_context()` for profiles, calls original rendering interface internally)
  - DO: add type data from rendering object (e.g. as in init manager)

CHAPTER

FIFTYTHREE

PICONGPU.PYPICONGPU

internal representation of params to generate PIConGPU input files

53.1 Subpackages

53.1.1 picongpu.pypicongpu.output

Submodules

picongpu.pypicongpu.output.auto

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre, Richard Pausch License: GPLv3+

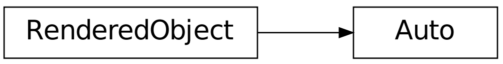
Module Contents

Classes

Auto	Class to provide output **without further configuration**.

class picongpu.pypicongpu.output.auto.**Auto**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Class to provide output **without further configuration**.

This class requires a period (in time steps) and will enable as many output plugins as feasable for all species. Note: The list of species from the initmanager is used during rendering.

No further configuration is possible! If you want to provide additional configuration for plugins, create a separate class.

period
    period to print data at

check() → None
    validate attributes

    if ok pass silently, otherwise raises error

    Raises

    - **ValueError** – period is non-negative integer
    - **ValueError** – species_names contains empty string
    - **ValueError** – species_names contains non-unique name

get_rendering_context() → dict
    get rendering context representation of this object

    delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None
    check if the given context is valid for the given type

    Raises on error, passes silently if okay.

    Raises

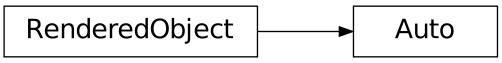
    - **ValidationError** – on schema violiation
    - **RuntimeError** – on schema not found

Package Contents

Classes

Auto	Class to provide output **without further configuration**.

class picongpu.pypicongpu.output.**Auto**
    Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Class to provide output **without further configuration**.

This class requires a period (in time steps) and will enable as many output plugins as feasable for all species. Note: The list of species from the initmanager is used during rendering.

No further configuration is possible! If you want to provide additional configuration for plugins, create a separate class.

`period`

period to print data at

`check()` → None

validate attributes

if ok pass silently, otherwise raises error

Raises

- `ValueError` – period is non-negative integer
- `ValueError` – species_names contains empty string
- `ValueError` – species_names contains non-unique name

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

53.1.2 `picongpu.pypicongpu.rendering`

Submodules

`picongpu.pypicongpu.rendering.renderedobject`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

Module Contents

Classes

RenderedObject	Class to be inherited from for rendering context generation

`class picongpu.pypicongpu.rendering.renderedobject.RenderedObject`

Class to be inherited from for rendering context generation

Every object that intends to generate a rendering context (typically as a dict) must inherit from this class and implement _get_serialized().

It is expected that _get_serialized() fully encodes the internal state, i.e. two equal (==) objects also return an equal result for _get_serialized().

For external usage, the method get_rendering_context() is provided. It passes _get_serialized() through, checking its result against a predefined json schema. If this schema is not fullfilled/not available, an error is raised.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.rendering.renderer

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

Module Contents

Classes

Renderer	helper class to render Mustache templates

class picongpu.pypicongpu.rendering.renderer.**Renderer**

helper class to render Mustache templates

Collection of (static) functions to render Mustache templates.

Also contains checks for structure of passed context (JSON) objects and the JSON schema checks and related functions (loading schema database, checking schemas, looking up schemas by type etc.)

static **check_rendering_context**(*context: Any*) → None

check if a context object may be renderd

If checks succeed passes silently, else raises.

Must be used *before* the preprocessor (as it checks that keys reserved for the preprocessor are not yet used).

Performs if the given object is acceptable as rendering context: - is dict - leafs are string, boolean, None, int, or float (or empty list) - child nodes are leaf, set or list - list items must be dict - keys are strings - keys do *not* contain dot (.) - keys do *not* begin with underscore (_) -> reserved for preprocessor

Parameters
context – object to be checked

Raises
Exception – on check failure

static get_context_preprocessed(*context: dict*) → dict

preprocess a context object

Applies the following preproccessing to an object: - list items have property "_first" and "_last" (boolean) added - numbers are translated to C++-compatible strings

rejects unchecked rendering contexts

Parameters
`context` – context to be preprocessed

Returns
preprocessed copy of context dict

static get_rendered_template(*context: dict*, *template: str*) → str

get rendered version of the provided string

Renders a given string using the given context object.

Parameters

- `context` – (checked and preprocessed) rendering context
- `template` – string containing template to be rendered

Returns
rendered template

static render_directory(*context: dict*, *path: str*) → None

Render all templates inside a given directory and remove the templates

Recursively find all files inside given path and render the files ending in ".mustache" to the same name without the ending. The original ".mustache" files are renamed with a dot "." prefix.

Parameters

- `context` – (checked and preprocessed) rendering context
- `path` – directory containing ".mustache" files

Package Contents

Classes

Renderer	helper class to render Mustache templates
RenderedObject	Class to be inherited from for rendering context generation

class picongpu.pypicongpu.rendering.Renderer

helper class to render Mustache templates

Collection of (static) functions to render Mustache templates.

Also contains checks for structure of passed context (JSON) objects and the JSON schema checks and related functions (loading schema database, checking schemas, looking up schemas by type etc.)

static check_rendering_context(*context: Any*) → None

check if a context object may be renderd

If checks succeed passes silently, else raises.

Must be used *before* the preprocessor (as it checks that keys reserved for the preprocessor are not yet used).

Performs if the given object is acceptable as rendering context: - is dict - leafs are string, boolean, None, int, or float (or empty list) - child nodes are leaf, set or list - list items must be dict - keys are strings - keys do *not* contain dot (.) - keys do *not* begin with underscore (_) -> reserved for preprocessor

Parameters
`context` – object to be checked

Raises
`Exception` – on check failure

`static get_context_preprocessed`(*context: dict*) → dict

preprocess a context object

Applies the following preproccessing to an object: - list items have property "_first" and "_last" (boolean) added - numbers are translated to C++-compatible strings

rejects unchecked rendering contexts

Parameters
`context` – context to be preprocessed

Returns
preprocessed copy of context dict

`static get_rendered_template`(*context: dict*, *template: str*) → str

get rendered version of the provided string

Renders a given string using the given context object.

Parameters

- `context` – (checked and preprocessed) rendering context
- `template` – string containing template to be rendered

Returns
rendered template

`static render_directory`(*context: dict*, *path: str*) → None

Render all templates inside a given directory and remove the templates

Recursively find all files inside given path and render the files ending in ".mustache" to the same name without the ending. The original ".mustache" files are renamed with a dot "." prefix.

Parameters

- `context` – (checked and preprocessed) rendering context
- `path` – directory containing ".mustache" files

`class picongpu.pypicongpu.rendering.RenderedObject`

Class to be inherited from for rendering context generation

Every object that intends to generate a rendering context (typically as a dict) must inherit from this class and implement _get_serialized().

It is expected that _get_serialized() fully encodes the internal state, i.e. two equal (==) objects also return an equal result for _get_serialized().

For external usage, the method get_rendering_context() is provided. It passes _get_serialized() through, checking its result against a predefined json schema. If this schema is not fullfilled/not available, an error is raised.

`get_rendering_context`() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static check_context_for_type(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

53.1.3 picongpu.pypicongpu.species

Data structures to specify and initialize particle species.

Note that the data structure (the classes) here use a different architecure than both (!) PIConGPU and PICMI.

Please refer to the documentation for a deeper discussion.

Subpackages

picongpu.pypicongpu.species.attribute

Submodules

picongpu.pypicongpu.species.attribute.attribute

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

Module Contents

Classes

Attribute	attribute of a species

class picongpu.pypicongpu.species.attribute.attribute.**Attribute**

attribute of a species

Property of individual macroparticles (i.e. can be different from macroparticle to macroparticle). Can change over time (not relevant for initialization here).

Owned by exactly one species.

Set by exactly one operation (an operation may define multiple attributes even across multiple species though).

Identified by its PIConGPU name.

PIConGPU term: “particle attributes”

PICONGPU_NAME: str

C++ Code implementing this attribute

picongpu.pypicongpu.species.attribute.boundelectrons

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

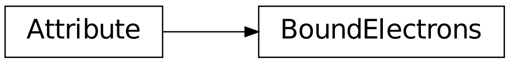
Module Contents

Classes

BoundElectrons	Position of a macroparticle

class picongpu.pypicongpu.species.attribute.boundelectrons.**BoundElectrons**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*



Position of a macroparticle

PICONGPU_NAME = 'boundElectrons'

picongpu.pypicongpu.species.attribute.momentum

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

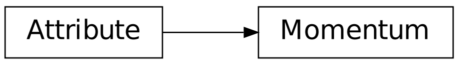
Module Contents

Classes

Momentum	Position of a macroparticle

class picongpu.pypicongpu.species.attribute.momentum.**Momentum**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*



Position of a macroparticle

PICONGPU_NAME = 'momentum'

`picongpu.pypicongpu.species.attribute.position`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

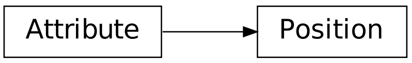
Module Contents

Classes

Position	Position of a macroparticle

class picongpu.pypicongpu.species.attribute.position.**Position**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*



Position of a macroparticle

PICONGPU_NAME = 'position<position_pic>'

`picongpu.pypicongpu.species.attribute.weighting`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

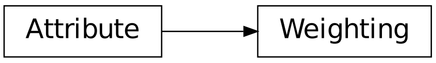
Module Contents

Classes

Weighting	Position of a macroparticle

class picongpu.pypicongpu.species.attribute.weighting.**Weighting**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*



Position of a macroparticle

PICONGPU_NAME = 'weighting'

Package Contents

Classes

Attribute	attribute of a species
Position	Position of a macroparticle
Weighting	Position of a macroparticle
Momentum	Position of a macroparticle
BoundElectrons	Position of a macroparticle

class picongpu.pypicongpu.species.attribute.**Attribute**

attribute of a species

Property of individual macroparticles (i.e. can be different from macroparticle to macroparticle). Can change over time (not relevant for initialization here).

Owned by exactly one species.

Set by exactly one operation (an operation may define multiple attributes even across multiple species though).

Identified by its PIConGPU name.

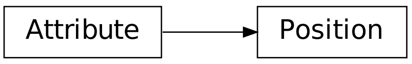
PIConGPU term: “particle attributes”

PICONGPU_NAME: str

C++ Code implementing this attribute

class picongpu.pypicongpu.species.attribute.**Position**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*

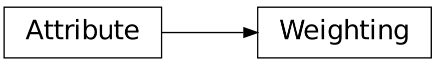


Position of a macroparticle

PICONGPU_NAME = 'position<position_pic>'

class picongpu.pypicongpu.species.attribute.**Weighting**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*

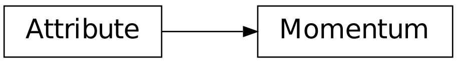


Position of a macroparticle

PICONGPU_NAME = 'weighting'

class picongpu.pypicongpu.species.attribute.**Momentum**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*

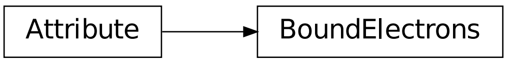


Position of a macroparticle

PICONGPU_NAME = 'momentum'

class picongpu.pypicongpu.species.attribute.**BoundElectrons**

Bases: *picongpu.pypicongpu.species.attribute.attribute.Attribute*



Position of a macroparticle

PICONGPU_NAME = 'boundElectrons'

picongpu.pypicongpu.species.constant

Submodules

picongpu.pypicongpu.species.constant.charge

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

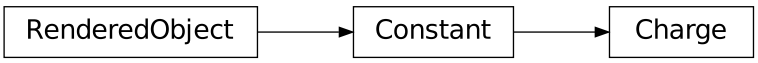
Module Contents

Classes

Charge	charge of a physical particle

class picongpu.pypicongpu.species.constant.charge.**Charge**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*



charge of a physical particle

charge_si

charge in C of an individual particle

check() → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

get_species_dependencies()

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

get_attribute_dependencies() → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation

- `RuntimeError` – on schema not found

`picongpu.pypicongpu.species.constant.constant`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

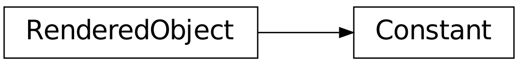
Module Contents

Classes

Constant	unmodifiable property of a species

`class picongpu.pypicongpu.species.constant.constant.`**`Constant`**

Bases: *`picongpu.pypicongpu.rendering.RenderedObject`*



unmodifiable property of a species

Equal for all macroparticles, e.g. mass, charge, ionization levels etc. Owned by **exactly one** species.

PIConGPU term: “particle flags”

Parent class for actual SpeciesConstants

A Constant can have the following dependencies:

- species objects
  - dependency species must be available for the init manager
  - code for dependency species will be generated *before* the code for this species
- attributes (types)
  - dependency attributes must be generated by *any* property on the species with this constant
  - there are no further checks employed, the constant is never shown the concrete attribute object of its species
- constants (types)
  - dependency constants must be present before the initmanager performs any actions
  - checks on the value of constants are not possible, only if a constant of a given type is present
  - (the concrete constant that is being checked is never passed to this constant)

`abstract check()` → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

`abstract get_species_dependencies()` → List

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

`abstract get_attribute_dependencies()` → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

`abstract get_constant_dependencies()` → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

picongpu.pypicongpu.species.constant.densityratio

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

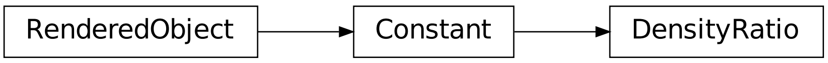
Module Contents

Classes

DensityRatio	factor for weighting when using profiles/deriving

class picongpu.pypicongpu.species.constant.densityratio.**DensityRatio**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*



factor for weighting when using profiles/deriving

ratio

factor for weighting calculation

check() → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

get_species_dependencies()

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

get_attribute_dependencies() → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.constant.elementproperties

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

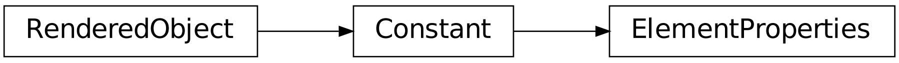
Module Contents

Classes

ElementProperties	represents constants associated to a chemical element

class picongpu.pypicongpu.species.constant.elementproperties.**ElementProperties**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*



represents constants associated to a chemical element

Produces PIConGPU atomic number and ionization energies.

Note: Not necessarily all of the generated properties will be required during runtime. However, this is left to the compiler to optimize (which is a core concept of PIConGPU).

element

represented chemical element

check()

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

`get_species_dependencies()`

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

`get_attribute_dependencies()` → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

`get_constant_dependencies()` → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

`picongpu.pypicongpu.species.constant.ionizers`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

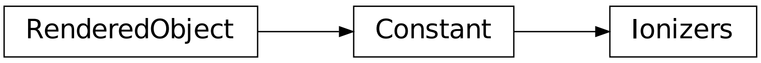
Module Contents

Classes

Ionizers	ionizers describing the ionization methods

class picongpu.pypicongpu.species.constant.ionizers.**Ionizers**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*



ionizers describing the ionization methods

Currently the selected ionizers are fixed by the code generation. When they are selectable by the user, they can be added here.

electron_species

species to be used as electrons

check() → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

get_species_dependencies()

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

get_attribute_dependencies() → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.constant.mass

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

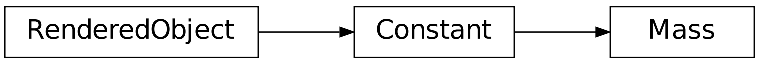
Module Contents

Classes

Mass	mass of a physical particle

class picongpu.pypicongpu.species.constant.mass.**Mass**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*



mass of a physical particle

mass_si

mass in kg of an individual particle

check() → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

get_species_dependencies()

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

get_attribute_dependencies() → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

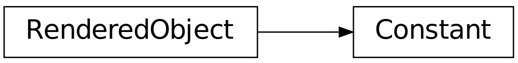
Package Contents

Classes

Constant	unmodifiable property of a species
Mass	mass of a physical particle
Charge	charge of a physical particle
DensityRatio	factor for weighting when using profiles/deriving
Ionizers	ionizers describing the ionization methods
ElementProperties	represents constants associated to a chemical element

class picongpu.pypicongpu.species.constant.**Constant**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



unmodifiable property of a species

Equal for all macroparticles, e.g. mass, charge, ionization levels etc. Owned by **exactly one** species.

PIConGPU term: "particle flags"

Parent class for actual SpeciesConstants

A Constant can have the following dependencies:

- species objects
  - dependency species must be available for the init manager
  - code for dependency species will be generated *before* the code for this species
- attributes (types)
  - dependency attributes must be generated by *any* property on the species with this constant
  - there are no further checks employed, the constant is never shown the concrete attribute object of its species
- constants (types)
  - dependency constants must be present before the initmanager performs any actions
  - checks on the value of constants are not possible, only if a constant of a given type is present
  - (the concrete constant that is being checked is never passed to this constant)

`abstract check()` → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

`abstract get_species_dependencies()` → List

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

`abstract get_attribute_dependencies()` → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

`abstract get_constant_dependencies()` → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

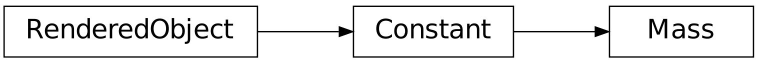
Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.constant.**Mass**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*



mass of a physical particle

mass_si

mass in kg of an individual particle

check() → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

get_species_dependencies()

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

get_attribute_dependencies() → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

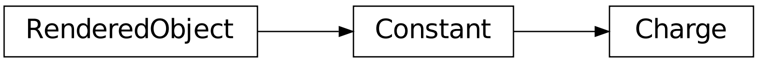
Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

`class picongpu.pypicongpu.species.constant.`**`Charge`**

Bases: *`picongpu.pypicongpu.species.constant.constant.Constant`*



charge of a physical particle

`charge_si`

charge in C of an individual particle

`check``()` → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

`get_species_dependencies``()`

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

`get_attribute_dependencies``()` → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.constant.**DensityRatio**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*

RenderedObject
Constant
DensityRatio

factor for weighting when using profiles/deriving

ratio

factor for weighting calculation

check() → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

get_species_dependencies()

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

get_attribute_dependencies() → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

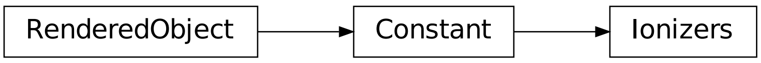
Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.constant.**Ionizers**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*



ionizers describing the ionization methods

Currently the selected ionizers are fixed by the code generation. When they are selectable by the user, they can be added here.

electron_species

species to be used as electrons

check() → None

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

get_species_dependencies()

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

get_attribute_dependencies() → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

get_constant_dependencies() → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.constant.**ElementProperties**

Bases: *picongpu.pypicongpu.species.constant.constant.Constant*

RenderedObject
Constant
ElementProperties

represents constants associated to a chemical element

Produces PIConGPU atomic number and ionization energies.

Note: Not necessarily all of the generated properties will be required during runtime. However, this is left to the compiler to optimize (which is a core concept of PIConGPU).

`element`

represented chemical element

`check()`

ensure validity of self

If ok passes silently, else raises. Intendend to check for invalid value (ranges), perhaps types etc.

Must be overwritten in child implementation.

`get_species_dependencies()`

get dependencies for definition

Returns a list of species which this flags requires being present. Mainly intended for ionization flags, i.e. should typically return [].

`get_attribute_dependencies()` → List[type]

get required attributes (during execution)

During execution some constants require an attribute to be present, e.g. the pusher required the Momentum attribute

This method returns a list of attribute types which it requires on its species.

`get_constant_dependencies()` → List[type]

get required constants (during execution)

Some constants (e.g. those selecting algorithms) may require other constants to be present.

This method returns the types of constants that must be present for this constant to be valid. Checking the value of these dependency constants is **NOT** possible.

Dependencies between constants **MAY** be circular. Rationale: Only presence of constants is checked, a reordering (as for inter-species-dependencies) is not performed, hence circular dependencies between constants can be handled.

However, as self-references do not make sense (theoretically speaking are always true), they are considered a programmer error and therefore self-references are **NOT** allowed.

This has no influence on the order of code generation.

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

`picongpu.pypicongpu.species.operation`

Subpackages

`picongpu.pypicongpu.species.operation.densityprofile`

Subpackages

`picongpu.pypicongpu.species.operation.densityprofile.plasmaramp`

Submodules

`picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.exponential`

This file is part of the PIConGPU. Copyright 2023 PIConGPU contributors Authors: Kristin Tippey, Brian Edward Marre License: GPLv3+

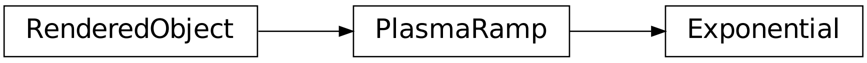
Module Contents

Classes

`Exponential`	exponential plasma ramp, either up or down

class `picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.exponential.`**`Exponential`**(*Plasma float*, *Plasma-Cutoff: float*)

Bases: *`picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.plasmaramp.PlasmaRamp`*



exponential plasma ramp, either up or down

`check`() → None

check self, overwritten by child class

Perform checks if own parameters are valid. passes silently if everything is okay

`get_generic_profile_rendering_context`() → dict

retrieve a context valid for "any profile"

Problem: Plasma Ramps are polymorph, there are several distinct implementations, each of them has its respective schema.

In the template we need know which exact sub type of the general abstract type was used to generate the correct code.

This is difficult in JSON (particularly in a mustache-compatible way) since no type information is available in the schema.

Solution: We store which type was actually used in a wrapper in addition to actual data,

provided as usual by get_rendering_context() by the plasma ramp instance

If a generic plasma ramp is requested we us the wrapper schema for this class which contains the **type** meta information and the **data** content

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- **get_rendering_context()**
    - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
    - _get_serialized() implemented in *every plasma ramp*
    - checks against schema of respective plasma ramp
    - **returned context is a representation of**
      exactly this plasma ramp
    - (left empty == not implemented in parent PlasmaRamp)
- **get_generic_profile_rendering_context()**
    - implemented in parent PlasmaRamp
    - returned representation is generic for *any plasma ramp* (i.e. contains which type is actually used)
    - passes information from get_rendering_context() through
    - returned representation is designed for easy use with templating engine mustache

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.none

This file is part of the PIConGPU. Copyright 2023 PIConGPU contributors Authors: Kristin Tippey, Brian Edward Marre License: GPLv3+

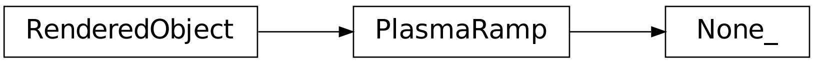
Module Contents

Classes

*None_*	no plasma ramp, either up or down

class picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.none.**None_**

Bases: *picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.plasmaramp.PlasmaRamp*



no plasma ramp, either up or down

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. passes silently if everything is okay

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Plasma Ramps are polymorph, there are several distinct implementations, each of them has its respective schema.

In the template we need know which exact sub type of the general abstract type was used to generate the correct code.

This is difficult in JSON (particularly in a mustache-compatible way) since no type information is available in the schema.

Solution: We store which type was actually used in a wrapper in addition to actual data,

provided as usual by get_rendering_context() by the plasma ramp instance

If a generic plasma ramp is requested we us the wrapper schema for this class which contains the **type** meta information and the **data** content

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- **get_rendering_context()**
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every plasma ramp*
  - checks against schema of respective plasma ramp
  - **returned context is a representation of**
    exactly this plasma ramp
  - (left empty == not implemented in parent PlasmaRamp)
- **get_generic_profile_rendering_context()**
  - implemented in parent PlasmaRamp
  - returned representation is generic for *any plasma ramp* (i.e. contains which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

`picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.plasmaramp`

This file is part of the PIConGPU. Copyright 2023 PIConGPU contributors Authors: Brian Edward Marre License: GPLv3+

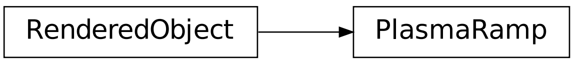
Module Contents

Classes

PlasmaRamp	abstract parent class for all plasma ramps

`class picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.plasmaramp.PlasmaRamp`

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



abstract parent class for all plasma ramps

A plasma ramp describes ramp up of an edge of an initial density distribution

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. passes silently if everything is okay

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Plasma Ramps are polymorph, there are several distinct implementations, each of them has its respective schema.

In the template we need know which exact sub type of the general abstract type was used to generate the correct code.

This is difficult in JSON (particularly in a mustache-compatible way) since no type information is available in the schema.

Solution: We store which type was actually used in a wrapper in addition to actual data,

provided as usual by get_rendering_context() by the plasma ramp instance

If a generic plasma ramp is requested we us the wrapper schema for this class which contains the **type** meta information and the **data** content

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- **get_rendering_context()**
    - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
    - _get_serialized() implemented in *every plasma ramp*
    - checks against schema of respective plasma ramp
    - **returned context is a representation of**
      exactly this plasma ramp
    - (left empty == not implemented in parent PlasmaRamp)
- **get_generic_profile_rendering_context()**

- implemented in parent PlasmaRamp
- returned representation is generic for *any plasma ramp* (i.e. contains which type is actually used)
- passes information from get_rendering_context() through
- returned representation is designed for easy use with templating engine mustache

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

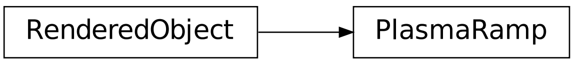
Package Contents

Classes

PlasmaRamp	abstract parent class for all plasma ramps
Exponential	exponential plasma ramp, either up or down
*None_*	no plasma ramp, either up or down

class picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.**PlasmaRamp**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



abstract parent class for all plasma ramps

A plasma ramp describes ramp up of an edge of an initial density distribution

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. passes silently if everything is okay

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Plasma Ramps are polymorph, there are several distinct implementations, each of them has its respective schema.

In the template we need know which exact sub type of the general abstract type was used to generate the correct code.

This is difficult in JSON (particularly in a mustache-compatible way) since no type information is available in the schema.

Solution: We store which type was actually used in a wrapper in addition to actual data,

> provided as usual by get_rendering_context() by the plasma ramp instance

If a generic plasma ramp is requested we us the wrapper schema for this class which contains the **type** meta information and the **data** content

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- **get_rendering_context()**
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every plasma ramp*
  - checks against schema of respective plasma ramp
  - **returned context is a representation of**
    exactly this plasma ramp
  - (left empty == not implemented in parent PlasmaRamp)
- **get_generic_profile_rendering_context()**
  - implemented in parent PlasmaRamp
  - returned representation is generic for *any plasma ramp* (i.e. contains which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

class picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.**Exponential**(*PlasmaLength: float*, *PlasmaCutoff: float*)

Bases: *picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.plasmaramp.PlasmaRamp*

RenderedObject
PlasmaRamp
Exponential

exponential plasma ramp, either up or down

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. passes silently if everything is okay

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Plasma Ramps are polymorph, there are several distinct implementations, each of them has its respective schema.

In the template we need know which exact sub type of the general abstract type was used to generate the correct code.

This is difficult in JSON (particularly in a mustache-compatible way) since no type information is available in the schema.

Solution: We store which type was actually used in a wrapper in addition to actual data,

provided as usual by get_rendering_context() by the plasma ramp instance

If a generic plasma ramp is requested we us the wrapper schema for this class which contains the **type** meta information and the **data** content

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- **get_rendering_context()**
    - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()

  - _get_serialized() implemented in *every plasma ramp*
  - checks against schema of respective plasma ramp
  - **returned context is a representation of**
    exactly this plasma ramp
  - (left empty == not implemented in parent PlasmaRamp)
- **get_generic_profile_rendering_context()**
  - implemented in parent PlasmaRamp
  - returned representation is generic for *any plasma ramp* (i.e. contains which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

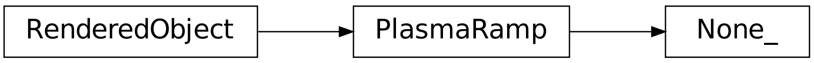
Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.**None_**

Bases: *picongpu.pypicongpu.species.operation.densityprofile.plasmaramp.plasmaramp.PlasmaRamp*



no plasma ramp, either up or down

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. passes silently if everything is okay

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Plasma Ramps are polymorph, there are several distinct implementations, each of them has its respective schema.

In the template we need know which exact sub type of the general abstract type was used to generate the correct code.

This is difficult in JSON (particularly in a mustache-compatible way) since no type information is available in the schema.

Solution: We store which type was actually used in a wrapper in addition to actual data,

provided as usual by get_rendering_context() by the plasma ramp instance

If a generic plasma ramp is requested we us the wrapper schema for this class which contains the **type** meta information and the **data** content

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- **get_rendering_context()**
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every plasma ramp*
  - checks against schema of respective plasma ramp
  - **returned context is a representation of**
    exactly this plasma ramp
  - (left empty == not implemented in parent PlasmaRamp)
- **get_generic_profile_rendering_context()**
  - implemented in parent PlasmaRamp
  - returned representation is generic for *any plasma ramp* (i.e. contains which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

Submodules

picongpu.pypicongpu.species.operation.densityprofile.densityprofile

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

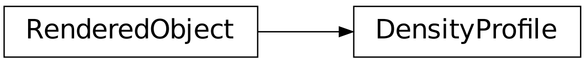
Module Contents

Classes

DensityProfile	(abstract) parent class of all density profiles

class picongpu.pypicongpu.species.operation.densityprofile.densityprofile.**DensityProfile**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



(abstract) parent class of all density profiles

A density profile describes the density in space.

abstract check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. On error raise, if everything is okay pass silently.

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Every profile has its respective schema, and it is difficult in JSON (particularly in a mustache-compatible way) to get the type of the schema.

Solution: The normal rendering of profiles get_rendering_context() provides **only their parameters**, i.e. there is **no meta information** on types etc.

If a generic profile is requested one can use the schema for "DensityProfile" (this class), for which this method returns the correct content, which includes metainformation and the data on the schema itself.

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- get_rendering_context()
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every profile*
  - checks against schema of respective profile
  - returned context is a representation of *exactly this profile*
  - (left empty == not implemented in parent ProfileDensity)
- get_generic_profile_rendering_context()
  - implemented in parent densityprofile
  - returned representation is generic for *any profile* (i.e. contains meta information which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.operation.densityprofile.foil

This file is part of the PIConGPU. Copyright 2023 PIConGPU contributors Authors: Kristin Tippey, Brian Edward Marre License: GPLv3+

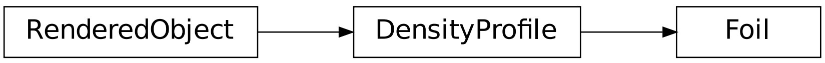
Module Contents

Classes

Foil	Directional density profile with thickness and pre- and

class picongpu.pypicongpu.species.operation.densityprofile.foil.**Foil**

Bases: *picongpu.pypicongpu.species.operation.densityprofile.densityprofile.DensityProfile*



Directional density profile with thickness and pre- and post-plasma lengths and cutoffs

density_si

density at every point in space (kg * m^-3)

y_value_front_foil_si

position of the front of the foil plateau (m)

thickness_foil_si

thickness of the foil plateau (m)

pre_foil_plasmaRamp

pre(lower y) foil-plateau ramp of density

post_foil_plasmaRamp

post(higher y) foil-plateau ramp of density

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. On error raise, if everything is okay pass silently.

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Every profile has its respective schema, and it is difficult in JSON (particularly in a mustache-compatible way) to get the type of the schema.

Solution: The normal rendering of profiles get_rendering_context() provides **only their parameters**, i.e. there is **no meta information** on types etc.

If a generic profile is requested one can use the schema for "DensityProfile" (this class), for which this method returns the correct content, which includes metainformation and the data on the schema itself.

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- get_rendering_context()
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every profile*

  - checks against schema of respective profile
  - returned context is a representation of *exactly this profile*
  - (left empty == not implemented in parent ProfileDensity)
- get_generic_profile_rendering_context()
  - implemented in parent densityprofile
  - returned representation is generic for *any profile* (i.e. contains meta information which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

`picongpu.pypicongpu.species.operation.densityprofile.uniform`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

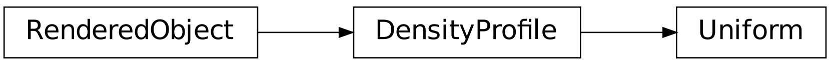
Module Contents

Classes

`Uniform`	globally constant density

`class picongpu.pypicongpu.species.operation.densityprofile.uniform.`**`Uniform`**

Bases: *`picongpu.pypicongpu.species.operation.densityprofile.densityprofile.DensityProfile`*



globally constant density

PIConGPU equivalent is the homogenous profile, but due to spelling ambiguities the PICMI name uniform is followed here.

`density_si`

density at every point in space (kg * m^-3)

`check``()` → None

check self, overwritten by child class

Perform checks if own parameters are valid. On error raise, if everything is okay pass silently.

`get_generic_profile_rendering_context``()` → dict

retrieve a context valid for "any profile"

Problem: Every profile has its respective schema, and it is difficult in JSON (particularly in a mustache-compatible way) to get the type of the schema.

Solution: The normal rendering of profiles get_rendering_context() provides **only their parameters**, i.e. there is **no meta information** on types etc.

If a generic profile is requested one can use the schema for "DensityProfile" (this class), for which this method returns the correct content, which includes metainformation and the data on the schema itself.

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- get_rendering_context()
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every profile*
  - checks against schema of respective profile
  - returned context is a representation of *exactly this profile*
  - (left empty == not implemented in parent ProfileDensity)
- get_generic_profile_rendering_context()
  - implemented in parent densityprofile
  - returned representation is generic for *any profile* (i.e. contains meta information which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static check_context_for_type(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

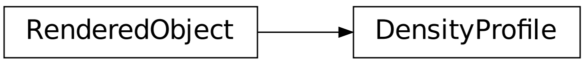
Package Contents

Classes

DensityProfile	(abstract) parent class of all density profiles
Uniform	globally constant density
Foil	Directional density profile with thickness and pre- and

class picongpu.pypicongpu.species.operation.densityprofile.**DensityProfile**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



(abstract) parent class of all density profiles

A density profile describes the density in space.

abstract check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. On error raise, if everything is okay pass silently.

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Every profile has its respective schema, and it is difficult in JSON (particularly in a mustache-compatible way) to get the type of the schema.

Solution: The normal rendering of profiles get_rendering_context() provides **only their parameters**, i.e. there is **no meta information** on types etc.

If a generic profile is requested one can use the schema for "DensityProfile" (this class), for which this method returns the correct content, which includes metainformation and the data on the schema itself.

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
```

(continues on next page)

(continued from previous page)

```
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- get_rendering_context()
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every profile*
  - checks against schema of respective profile
  - returned context is a representation of *exactly this profile*
  - (left empty == not implemented in parent ProfileDensity)
- get_generic_profile_rendering_context()
  - implemented in parent densityprofile
  - returned representation is generic for *any profile* (i.e. contains meta information which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

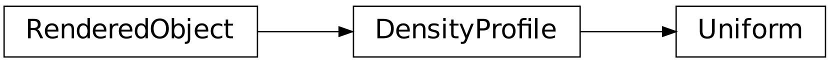
Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

`class picongpu.pypicongpu.species.operation.densityprofile.`**`Uniform`**

Bases: *`picongpu.pypicongpu.species.operation.densityprofile.densityprofile.DensityProfile`*



globally constant density

PICONGPU equivalent is the homogenous profile, but due to spelling ambiguities the PICMI name uniform is followed here.

density_si

density at every point in space (kg * m^-3)

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. On error raise, if everything is okay pass silently.

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Every profile has its respective schema, and it is difficult in JSON (particularly in a mustache-compatible way) to get the type of the schema.

Solution: The normal rendering of profiles get_rendering_context() provides **only their parameters**, i.e. there is **no meta information** on types etc.

If a generic profile is requested one can use the schema for "DensityProfile" (this class), for which this method returns the correct content, which includes metainformation and the data on the schema itself.

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- get_rendering_context()
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every profile*
  - checks against schema of respective profile
  - returned context is a representation of *exactly this profile*
  - (left empty == not implemented in parent ProfileDensity)
- get_generic_profile_rendering_context()
  - implemented in parent densityprofile
  - returned representation is generic for *any profile* (i.e. contains meta information which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static check_context_for_type(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

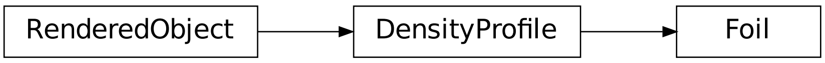
Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.operation.densityprofile.**Foil**

Bases: *picongpu.pypicongpu.species.operation.densityprofile.densityprofile.DensityProfile*



Directional density profile with thickness and pre- and post-plasma lengths and cutoffs

density_si

density at every point in space (kg * m^-3)

y_value_front_foil_si

position of the front of the foil plateau (m)

thickness_foil_si

thickness of the foil plateau (m)

pre_foil_plasmaRamp

pre(lower y) foil-plateau ramp of density

post_foil_plasmaRamp

post(higher y) foil-plateau ramp of density

check() → None

check self, overwritten by child class

Perform checks if own parameters are valid. On error raise, if everything is okay pass silently.

get_generic_profile_rendering_context() → dict

retrieve a context valid for "any profile"

Problem: Every profile has its respective schema, and it is difficult in JSON (particularly in a mustache-compatible way) to get the type of the schema.

Solution: The normal rendering of profiles get_rendering_context() provides **only their parameters**, i.e. there is **no meta information** on types etc.

If a generic profile is requested one can use the schema for "DensityProfile" (this class), for which this method returns the correct content, which includes metainformation and the data on the schema itself.

E.g.:

```
{
    "type": {
        "uniform": true,
        "gaussian": false,
```

(continues on next page)

(continued from previous page)

```
        ...
    },
    "data": DATA
}
```

where DATA is the serialization as returned by get_rendering_context().

There are *two* context serialization methods for density profiles:

- get_rendering_context()
  - provided by RenderedObject parent class, serialization ("context building") performed by _get_serialized()
  - _get_serialized() implemented in *every profile*
  - checks against schema of respective profile
  - returned context is a representation of *exactly this profile*
  - (left empty == not implemented in parent ProfileDensity)
- get_generic_profile_rendering_context()
  - implemented in parent densityprofile
  - returned representation is generic for *any profile* (i.e. contains meta information which type is actually used)
  - passes information from get_rendering_context() through
  - returned representation is designed for easy use with templating engine mustache

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.operation.momentum

Submodules

picongpu.pypicongpu.species.operation.momentum.drift

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

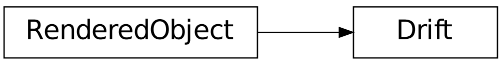
Module Contents

Classes

Drift	Add drift to a species (momentum)

`class picongpu.pypicongpu.species.operation.momentum.drift.`**`Drift`**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Add drift to a species (momentum)

Note that the drift is specified by a direction (normalized velocity vector) and gamma. Helpers to load from other representations (originating from PICMI) are provided.

`direction_normalized`

direction of drift, length of one

`gamma`

gamma, the physicists know

`check``()` → None

check attributes for correctness

pass silently if everything is OK, throw error otherwise

`fill_from_velocity``(`*velocity: Tuple[float, float, float]*`)` → None

set attributes to represent given velocity vector

computes gamma and direction_normalized for self

Parameters

`velocity` – velocity given as vector

`fill_from_gamma_velocity``(`*gamma_velocity: Tuple[float, float, float]*`)` → None

set attributes to represent given velocity vector multiplied with gamma

computes gamma and direction_normalized for self

Parameters

`velocity` – velocity given as vector multiplied with gamma

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.operation.momentum.temperature

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

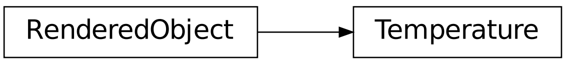
Module Contents

Classes

Temperature	Initialize momentum from temperature

class picongpu.pypicongpu.species.operation.momentum.temperature.**Temperature**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Initialize momentum from temperature

temperature_kev

temperature to use in keV

check() → None

check validity of self

pass silently if okay, raise on error

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static check_context_for_type(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

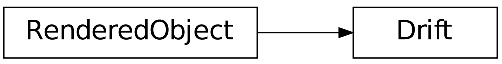
Package Contents

Classes

Drift	Add drift to a species (momentum)
Temperature	Initialize momentum from temperature

class picongpu.pypicongpu.species.operation.momentum.**Drift**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Add drift to a species (momentum)

Note that the drift is specified by a direction (normalized velocity vector) and gamma. Helpers to load from other representations (originating from PICMI) are provided.

direction_normalized

direction of drift, length of one

gamma

gamma, the physicists know

check() → None

check attributes for correctness

pass silently if everything is OK, throw error otherwise

fill_from_velocity(*velocity: Tuple[float, float, float]*) → None

set attributes to represent given velocity vector

computes gamma and direction_normalized for self

Parameters
velocity – velocity given as vector

fill_from_gamma_velocity(*gamma_velocity: Tuple[float, float, float]*) → None

set attributes to represent given velocity vector multiplied with gamma

computes gamma and direction_normalized for self

Parameters
velocity – velocity given as vector multiplied with gamma

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static check_context_for_type(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

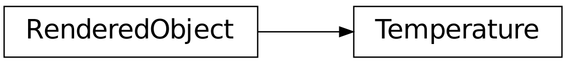
Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

`class picongpu.pypicongpu.species.operation.momentum.`**`Temperature`**

Bases: *`picongpu.pypicongpu.rendering.RenderedObject`*



Initialize momentum from temperature

`temperature_kev`

temperature to use in keV

`check``()` → None

check validity of self

pass silently if okay, raise on error

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

Submodules

`picongpu.pypicongpu.species.operation.densityoperation`

This file is part of the PIConGPU. Copyright 2023-2023 PIConGPU contributors Authors: Brian Edward Marre License: GPLv3+

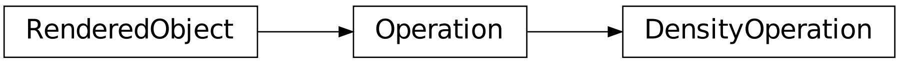
Module Contents

Classes

DensityOperation	common interface for all operations that create density

class picongpu.pypicongpu.species.operation.densityoperation.**DensityOperation**

Bases: *picongpu.pypicongpu.species.operation.operation.Operation*



common interface for all operations that create density
and the not-placed operation

attributes_by_species

attributes (exclusively) initialized by this operation

abstract check_preconditions() → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

abstract prebook_species_attributes() → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

bake_species_attributes() → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned

- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.operation.noboundelectrons

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

Module Contents

Classes

NoBoundElectrons	assigns a BoundElectrons attribute, but leaves it a 0

class picongpu.pypicongpu.species.operation.noboundelectrons.**NoBoundElectrons**

Bases: *picongpu.pypicongpu.species.operation.operation.Operation*

RenderedObject
Operation
NoBoundElectrons

assigns a BoundElectrons attribute, but leaves it a 0

Intended use for fully ionized ions, which do CAN be ionized. (Fully ionized ions which can NOT be ionized do not require a BoundElectrons attribute, and therefore no operation to assign it.)

`species`
species which will have boundElectorns set to 0

`attributes_by_species`
attributes (exclusively) initialized by this operation

`check_preconditions()` → None
check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`prebook_species_attributes()` → None
fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None
applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

`get_rendering_context()` → dict
get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static check_context_for_type(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.operation.notplaced

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

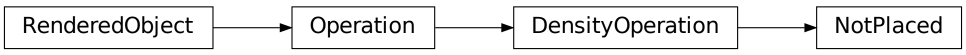
Module Contents

Classes

NotPlaced	assigns a position attribute, but does not place a species

class picongpu.pypicongpu.species.operation.notplaced.**NotPlaced**

Bases: *picongpu.pypicongpu.species.operation.densityoperation.DensityOperation*



assigns a position attribute, but does not place a species

Intended for electrons which do not have a profile, but are used in pre-ionization.

Provides attributes Position & Weighting.

species

species which will not be placed

ppc = 0

attributes_by_species

attributes (exclusively) initialized by this operation

check_preconditions() → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`prebook_species_attributes()` → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

picongpu.pypicongpu.species.operation.operation

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

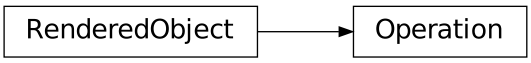
Module Contents

Classes

Operation	Defines the initialization of a set of attributes across multiple species

class picongpu.pypicongpu.species.operation.operation.**Operation**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Defines the initialization of a set of attributes across multiple species

One attribute of one species is initialized by exactly one Operation.

May require the species to have certain constants.

Operations are treated strictly independently from one another. Dependencies are thus handled inside of a single Operation. (Note: The code generation may impose a custom order between the types of operations.)

All Operations combined generate the PIConGPU init pipeline.

This leads to some notable cases:
- Species that are to be placed together (ion+electrons) must be initialized by the **same** Operation.
- Dependent attributes must be initialized by the **same** Operation.

The typical lifecycle of an Operation object is as follows:
1. Get created (i.e. __init__() is called)
   - performed from outside (PICMI interface)
   - affected species & parameters passed from outside (store as class attribute)
2. check preconditions for passed species
   - params set correctly
   - required constants present in species
   - At the point of checking the given species object are not fully defined yet: **DO NOT CHECK FOR OTHER ATTRIBUTES, THEY WILL NOT BE THERE!!!**
   - operation-implemented in self.check_preconditions()
3. prebook attributes (for each species)
   - generate attribute objects
   - "bookmark" that this operation will add them to the species, but does **NOT** assign the attributes (see 4.)
   - stored inside self.attributes_by_species

   - operation-implemented in self.prebook_species_attributes()
4. associate generated attributes with their respective species
   - invoked by the initmanager (do **not** do this in this class)
   - based on self.attributes_by_species
   - (performed by pre-defined self.bake_species_attributes())

So to define your own operator:

- inherit from this class
- accept species list & additional params from users (and store in attributes of self) -> no pre-defined interface, define your own
- implement self.check_preconditions() -> check for required constants etc.
- implement self.prebook_species_attributes() -> fill self.attributes_by_species

Then use your operator somewhere and register it with the InitializationManager, which will make calls to the functions above, check for incompatibilities to other operators and finally associates the generated attributes to their species.

`attributes_by_species`

attributes (exclusively) initialized by this operation

`abstract check_preconditions()` → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`abstract prebook_species_attributes()` → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.operation.setboundelectrons

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

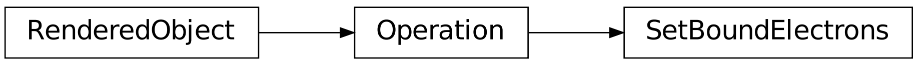
Module Contents

Classes

SetBoundElectrons	assigns and set the boundElectrons attribute

class picongpu.pypicongpu.species.operation.setboundelectrons.**SetBoundElectrons**

Bases: *picongpu.pypicongpu.species.operation.operation.Operation*



assigns and set the boundElectrons attribute

Standard attribute for pre-ionization.

species

species which will have boundElectrons set

bound_electrons

number of bound electrons to set

`attributes_by_species`

attributes (exclusively) initialized by this operation

`check_preconditions()` → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`prebook_species_attributes()` → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.operation.simpledensity

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

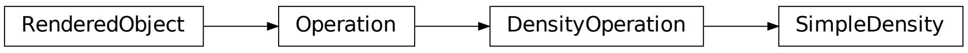
Module Contents

Classes

SimpleDensity	Place a set of species together, using the same density profile

class picongpu.pypicongpu.species.operation.simpledensity.**SimpleDensity**

Bases: *picongpu.pypicongpu.species.operation.densityoperation.DensityOperation*



Place a set of species together, using the same density profile

These species will have **the same** macroparticle placement.

For this operation, only the random layout is supported.

parameters:

- ppc: particles placed per cell
- profile: density profile to use
- species: species to be placed with the given profile note that their density ratios will be respected

ppc

particles per cell (random layout), >0

profile

density profile to use, describes the actual density

species

species to be placed

attributes_by_species

attributes (exclusively) initialized by this operation

check_preconditions() → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`prebook_species_attributes()` → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

picongpu.pypicongpu.species.operation.simplemomentum

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

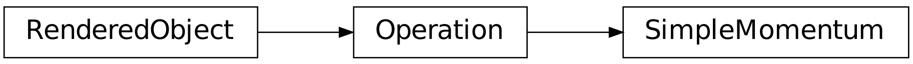
Module Contents

Classes

SimpleMomentum	provides momentum to a species

class picongpu.pypicongpu.species.operation.simplemomentum.**SimpleMomentum**

Bases: *picongpu.pypicongpu.species.operation.operation.Operation*



provides momentum to a species

specified by:
- temperature
- drift

Both are optional. If both are missing, momentum **is still provided**, but left at 0 (default).

species

species for which momentum will be set

temperature

temperature of particles (if any)

drift

drift of particles (if any)

attributes_by_species

attributes (exclusively) initialized by this operation

check_preconditions() → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

prebook_species_attributes() → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

bake_species_attributes() → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

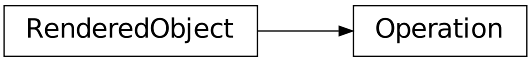
Package Contents

Classes

Operation	Defines the initialization of a set of attributes across multiple species
DensityOperation	common interface for all operations that create density
SimpleDensity	Place a set of species together, using the same density profile
NotPlaced	assigns a position attribute, but does not place a species
SimpleMomentum	provides momentum to a species
NoBoundElectrons	assigns a BoundElectrons attribute, but leaves it a 0
SetBoundElectrons	assigns and set the boundElectrons attribute

`class picongpu.pypicongpu.species.operation.Operation`

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Defines the initialization of a set of attributes across multiple species

One attribute of one species is initialized by exactly one Operation.

May require the species to have certain constants.

Operations are treated strictly independently from one another. Dependencies are thus handled inside of a single Operation. (Note: The code generation may impose a custom order between the types of operations.)

All Operations combined generate the PIConGPU init pipeline.

This leads to some notable cases:

- Species that are to be placed together (ion+electrons) must be initialized by the **same** Operation.
- Dependent attributes must be initialized by the **same** Operation.

The typical lifecycle of an Operation object is as follows:

1. Get created (i.e. __init__() is called)
    - performed from outside (PICMI interface)
    - affected species & parameters passed from outside (store as class attribute)
2. check preconditions for passed species
    - params set correctly
    - required constants present in species
    - At the point of checking the given species object are not fully defined yet: **DO NOT CHECK FOR OTHER ATTRIBUTES, THEY WILL NOT BE THERE!!!**
    - operation-implemented in self.check_preconditions()
3. prebook attributes (for each species)
    - generate attribute objects
    - "bookmark" that this operation will add them to the species, but does **NOT** assign the attributes (see 4.)

   - stored inside self.attributes_by_species
   - operation-implemented in self.prebook_species_attributes()
4. associate generated attributes with their respective species
   - invoked by the initmanager (do **not** do this in this class)
   - based on self.attributes_by_species
   - (performed by pre-defined self.bake_species_attributes())

So to define your own operator:

- inherit from this class
- accept species list & additional params from users (and store in attributes of self) -> no pre-defined interface, define your own
- implement self.check_preconditions() -> check for required constants etc.
- implement self.prebook_species_attributes() -> fill self.attributes_by_species

Then use your operator somewhere and register it with the InitializationManager, which will make calls to the functions above, check for incompatibilities to other operators and finally associates the generated attributes to their species.

`attributes_by_species`

attributes (exclusively) initialized by this operation

`abstract check_preconditions()` → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`abstract prebook_species_attributes()` → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.operation.**DensityOperation**

Bases: *picongpu.pypicongpu.species.operation.operation.Operation*

RenderedObject
Operation
DensityOperation

common interface for all operations that create density
and the not-placed operation

attributes_by_species

attributes (exclusively) initialized by this operation

abstract **check_preconditions**() → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

abstract **prebook_species_attributes**() → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes``()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

`get_rendering_context``()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None

check if the given context is valid for the given type

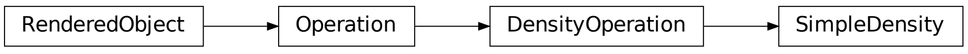
Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

`class picongpu.pypicongpu.species.operation.`**`SimpleDensity`**

Bases: *`picongpu.pypicongpu.species.operation.densityoperation.DensityOperation`*



Place a set of species together, using the same density profile

These species will have **the same** macroparticle placement.

For this operation, only the random layout is supported.

parameters:
- ppc: particles placed per cell
- profile: density profile to use
- species: species to be placed with the given profile note that their density ratios will be respected

`ppc`

particles per cell (random layout), >0

`profile`

density profile to use, describes the actual density

`species`

species to be placed

`attributes_by_species`

attributes (exclusively) initialized by this operation

`check_preconditions()` → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`prebook_species_attributes()` → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

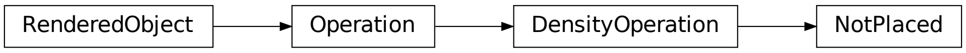
Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.operation.**NotPlaced**

Bases: *picongpu.pypicongpu.species.operation.densityoperation.DensityOperation*



assigns a position attribute, but does not place a species

Intended for electrons which do not have a profile, but are used in pre-ionization.

Provides attributes Position & Weighting.

species

species which will not be placed

ppc = 0

attributes_by_species

attributes (exclusively) initialized by this operation

check_preconditions() → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

prebook_species_attributes() → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

`class picongpu.pypicongpu.species.operation.`**`SimpleMomentum`**

Bases: *`picongpu.pypicongpu.species.operation.operation.Operation`*

RenderedObject
Operation
SimpleMomentum

provides momentum to a species

specified by:
- temperature
- drift

Both are optional. If both are missing, momentum **is still provided**, but left at 0 (default).

`species`
species for which momentum will be set

`temperature`
temperature of particles (if any)

`drift`
drift of particles (if any)

`attributes_by_species`
attributes (exclusively) initialized by this operation

`check_preconditions()` → None
check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`prebook_species_attributes()` → None
fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None
applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

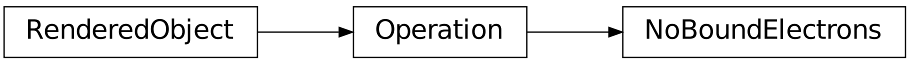
Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.operation.**NoBoundElectrons**

Bases: *picongpu.pypicongpu.species.operation.operation.Operation*



assigns a BoundElectrons attribute, but leaves it a 0

Intended use for fully ionized ions, which do CAN be ionized. (Fully ionized ions which can NOT be ionized do not require a BoundElectrons attribute, and therefore no operation to assign it.)

species

species which will have boundElectorns set to 0

attributes_by_species

attributes (exclusively) initialized by this operation

check_preconditions() → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

prebook_species_attributes() → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

bake_species_attributes() → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

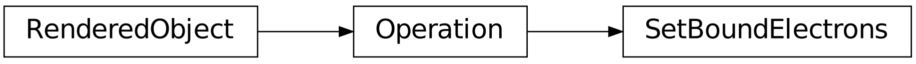
Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

class picongpu.pypicongpu.species.operation.**SetBoundElectrons**

Bases: *picongpu.pypicongpu.species.operation.operation.Operation*



assigns and set the boundElectrons attribute

Standard attribute for pre-ionization.

species

species which will have boundElectrons set

bound_electrons

number of bound electrons to set

`attributes_by_species`

attributes (exclusively) initialized by this operation

`check_preconditions()` → None

check own parameters and species this operator will be applied to

Must be implemented (overwritten) by the operator.

Throws if a precondition is not met, and passes silently if everything is okay.

If this operator relies on certain constants of classes to be set, (e.g. mass, charge) this must be checked here.

All parameters, including the species this Operator is to be applied to must be passed beforehand. Note: there is no unified way of passing parameters, please define your own. (Rationale: The structure of parameters is very heterogeneous.)

`prebook_species_attributes()` → None

fills attributes_by_species

Must be implemented (overwritten) by the operator.

Generates the Attribute objects and pre-books them together with the species they are to be initialized for in self.attributes_by_species.

Will only be run after self.check_preconditions() has passed.

Note: Additional checks are not required, compatibility to other operations will be ensured from outside.

MUST **NOT** MODIFY THE SPECIES.

`bake_species_attributes()` → None

applies content of attributes_by_species to species

For each species in attributes_by_species.keys() assigns the attributes to their respective species, precisely appends to the list Species.attributes

Expects check_preconditions() and prebook_species_attributes() to have passed previously (without error).

Additionally, performs the following sanity checks:

- at least one attribute is assigned
- the species does not already have an attribute of the same type
- every attribute is assigned exclusively to one species

Intended usage:

1. check for dependencies in used species
2. fill self.attributes_by_species (with newly generated objects) must be performed by self.prebook_species_attributes()
3. call self.bake_species_attributes() (this method) to set species.attributes accordingly

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violation
- `RuntimeError` – on schema not found

`picongpu.pypicongpu.species.util`

Submodules

`picongpu.pypicongpu.species.util.element`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

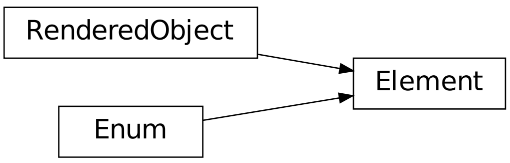
Module Contents

Classes

`Element`	Denotes an element from the periodic table of elements

`class picongpu.pypicongpu.species.util.element.`**`Element`**`(*args, **kwds)`

Bases: *`picongpu.pypicongpu.rendering.RenderedObject`*, `enum.Enum`



Denotes an element from the periodic table of elements

Used to provide fundamental constants for elements, and to map them in a type-safe way to PIConGPU.

The number associated is the number of protons. Note: Spelling follows periodic table, e.g. "Na", "C", "He"

Note that these denote Elements, but when initialized in a species *only* represent the core, i.e. there are no electrons. To make an atom also initialize an appropriate ionization.

`H = 1`

hydrogen

`He = 2`

helium

`N = 7`

nitrogen

`static get_by_openpmd_name`(*openpmd_name: str*) → *Element*

get the correct substance implementation from a openPMD type name

Names are (case-sensitive) element symbols (e.g. “H”, “He”, “N”).

Parameters
`openpmd_name` – single species following openPMD species extension

Returns
object representing the given species

`get_picongpu_name`() → str

get name as used in PIConGPU

Used for type name lookups

`get_mass_si`() → float

Get mass of an individual particle

Calculated based of “conventional atomic weight” from the periodic table of elements

Returns
mass in kg

`get_charge_si`() → float

Get charge of an individual particle *without* electrons

Calculated from atomic number, applies with no electrons present.

Returns
charge in C

`get_rendering_context`() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

`__repr__`()

Return repr(self).

`__str__`()

Return str(self).

`__dir__`()

Returns all members and all public methods

`__format__`(*format_spec*)

Default object formatter.

`__hash__`()

Return hash(self).

`__reduce_ex__`(*proto*)

Helper for pickle.

__deepcopy__(*memo*)

__copy__()

name()

The name of the Enum member.

value()

The value of the Enum member.

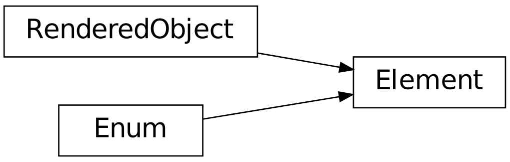
Package Contents

Classes

Element	Denotes an element from the periodic table of elements

class picongpu.pypicongpu.species.util.**Element**(**args*, ***kwds*)

Bases: *picongpu.pypicongpu.rendering.RenderedObject*, enum.Enum



Denotes an element from the periodic table of elements

Used to provide fundamental constants for elements, and to map them in a type-safe way to PIConGPU.

The number associated is the number of protons. Note: Spelling follows periodic table, e.g. “Na”, “C”, “He”

Note that these denote Elements, but when initialized in a species *only* represent the core, i.e. there are no electrons. To make an atom also initialize an appropriate ionization.

H = 1

hydrogen

He = 2

helium

N = 7

nitrogen

static **get_by_openpmd_name**(*openpmd_name: str*) → *Element*

get the correct substance implementation from a openPMD type name

Names are (case-sensitive) element symbols (e.g. “H”, “He”, “N”).

Parameters

openpmd_name – single species following openPMD species extension

Returns

object representing the given species

`get_picongpu_name()` → str

get name as used in PIConGPU

Used for type name lookups

`get_mass_si()` → float

Get mass of an individual particle

Calculated based of "conventional atomic weight" from the periodic table of elements

Returns
mass in kg

`get_charge_si()` → float

Get charge of an individual particle *without* electrons

Calculated from atomic number, applies with no electrons present.

Returns
charge in C

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

`__repr__()`

Return repr(self).

`__str__()`

Return str(self).

`__dir__()`

Returns all members and all public methods

`__format__`(*format_spec*)

Default object formatter.

`__hash__()`

Return hash(self).

`__reduce_ex__`(*proto*)

Helper for pickle.

`__deepcopy__`(*memo*)

`__copy__()`

`name()`

The name of the Enum member.

`value()`

The value of the Enum member.

Submodules

`picongpu.pypicongpu.species.initmanager`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

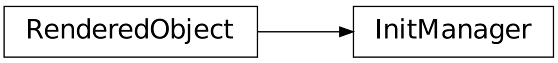
Module Contents

Classes

`InitManager`	Helper to manage species translation to PIConGPU

`class picongpu.pypicongpu.species.initmanager.`**`InitManager`**

Bases: *`picongpu.pypicongpu.rendering.RenderedObject`*



Helper to manage species translation to PIConGPU

Collects all species to be initialized and the operations initializing them.

Invokes the methods of the Operation lifecycle (check, prebook, bake).

Workflow:

1. Fill InitManager (from outside) with
   - Species and their constants (no attributes!)
   - Operations, fully initialized (all params set)
2. invoke InitManager.bake(), which
   - checks the Species for conflicts (name…)
   - performs dependency checks, possibly reorders species
   - invokes the Operation lifecycle (check, prebook, bake)
   - sanity-checks the results
3. retrieve rendering context
   - organizes operations into lists

Note: The InitManager manages a lifecycle, it does not perform deep checks
-> most errors have to be caught by delegated checks.

`all_species`

all species to be initialized

`all_operations`

all species to be initialized

`bake``()` → None

apply operations to species

get_typical_particle_per_cell() → int

get typical number of macro particles per cell(ppc) of simulation

@returns middle value of ppc-range of all operations, minimum 1

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

picongpu.pypicongpu.species.species

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

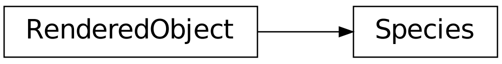
Module Contents

Classes

Species	PyPIConGPU species definition

class picongpu.pypicongpu.species.species.**Species**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



PyPIConGPU species definition

A "species" is a set of particles, which is defined by:
- A set of species constants (mass, charge, etc.),
- a set of species attributes (position, number of bound electrons), and
- a set of operations which collectively initialize these attributes, where one attribute is initializated by exactly one operation.
- (and a name)

Note that some of the species attributes or constants are considered mandatory. Each species constant or attribute may only be defined once.

constants

PIConGPU particle flags

attributes

PIConGPU particle attributes

name

name of the species

get_cxx_typename() → str

get (standalone) C++ name for this species

check() → None

sanity-check self, if ok pass silently

Ensure that:

- species has valid name
- constants have unique types
- attributes have unique types

get_constant_by_type(*needle_type: Type[*picongpu.pypicongpu.species.constant.Constant*]*) → *picongpu.pypicongpu.species.constant.Constant*

retrieve constant of given type, raise if not found

Searches through constants of this species and returns the constant of the given type if found. If no constant of this type is found, an error is raised.

Parameters

`needle_type` – constant type to look for

Raises

`RuntimeError` – on failure to find constant of given type

Returns

constant of given type

has_constant_of_type(*needle_type: Type[*picongpu.pypicongpu.species.constant.Constant*]*) → bool

lookup if constant of given type is present

Searches through constants of this species and returns true iff a constant of the given type is present.

Parameters

`needle_type` – constant type to look for

Returns

whether constant of needle_type exists

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

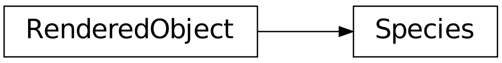
Package Contents

Classes

`Species`	PyPIConGPU species definition
`InitManager`	Helper to manage species translation to PIConGPU

`class picongpu.pypicongpu.species.`**`Species`**

Bases: `picongpu.pypicongpu.rendering.RenderedObject`



PyPIConGPU species definition

A "species" is a set of particles, which is defined by:

- A set of species constants (mass, charge, etc.),
- a set of species attributes (position, number of bound electrons), and
- a set of operations which collectively initialize these attributes, where one attribute is initializated by exactly one operation.
- (and a name)

Note that some of the species attributes or constants are considered mandatory. Each species constant or attribute may only be defined once.

`constants`

PIConGPU particle flags

`attributes`

PIConGPU particle attributes

`name`

name of the species

`get_cxx_typename``()` → str

get (standalone) C++ name for this species

`check``()` → None

sanity-check self, if ok pass silently

Ensure that:

- species has valid name
- constants have unique types
- attributes have unique types

`get_constant_by_type``(`*needle_type: Type[*picongpu.pypicongpu.species.constant.Constant*]*`)` → *picongpu.pypicongpu.species.constant.Constant*

retrieve constant of given type, raise if not found

Searches through constants of this species and returns the constant of the given type if found. If no constant of this type is found, an error is raised.

Parameters

`needle_type` – constant type to look for

Raises
`RuntimeError` – on failure to find constant of given type

Returns
constant of given type

has_constant_of_type(*needle_type: Type[picongpu.pypicongpu.species.constant.Constant]*) → bool

lookup if constant of given type is present

Searches through constants of this species and returns true iff a constant of the given type is present.

Parameters
`needle_type` – constant type to look for

Returns
whether constant of needle_type exists

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

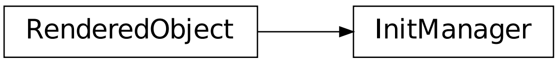
Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

class picongpu.pypicongpu.species.**InitManager**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Helper to manage species translation to PIConGPU

Collects all species to be initialized and the operations initializing them.

Invokes the methods of the Operation lifecycle (check, prebook, bake).

Workflow:

1. Fill InitManager (from outside) with
   - Species and their constants (no attributes!)
   - Operations, fully initialized (all params set)
2. invoke InitManager.bake(), which
   - checks the Species for conflicts (name…)
   - performs dependency checks, possibly reorders species
   - invokes the Operation lifecycle (check, prebook, bake)
   - sanity-checks the results
3. retrieve rendering context

- organizes operations into lists

Note: The InitManager manages a lifecycle, it does not perform deep checks
-> most errors have to be caught by delegated checks.

`all_species`
all species to be initialized

`all_operations`
all species to be initialized

`bake``()` → None
apply operations to species

`get_typical_particle_per_cell``()` → int
get typical number of macro particles per cell(ppc) of simulation

@returns middle value of ppc-range of all operations, minimum 1

`get_rendering_context``()` → dict
get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**`(`*type_to_check: type*, *context: dict*`)` → None
check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

53.2 Submodules

53.2.1 `picongpu.pypicongpu.grid`

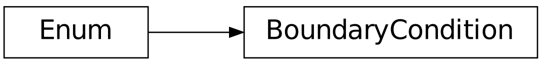
This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre, Richard Pausch License: GPLv3+

Module Contents

Classes

`BoundaryCondition`	Boundary Condition of PIConGPU
`Grid3D`	PIConGPU 3 dimensional (cartesian) grid

`class picongpu.pypicongpu.grid.`**`BoundaryCondition`**`(`**args*, ***kwds*`)`
Bases: `enum.Enum`



Boundary Condition of PIConGPU

Defines how particles that pass the simulation bounding box are treated.

TODO: implement the other methods supported by PIConGPU (reflecting, thermal)

PERIODIC = 1

ABSORBING = 2

get_cfg_str() → str

Get string equivalent for cfg files :return: string for –periodic

__repr__()

Return repr(self).

__str__()

Return str(self).

__dir__()

Returns all members and all public methods

__format__(*format_spec*)

Default object formatter.

__hash__()

Return hash(self).

__reduce_ex__(*proto*)

Helper for pickle.

__deepcopy__(*memo*)

__copy__()

name()

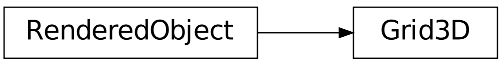
The name of the Enum member.

value()

The value of the Enum member.

class picongpu.pypicongpu.grid.**Grid3D**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



PIConGPU 3 dimensional (cartesian) grid

Defined by the dimensions of each cell and the number of cells per axis.

The bounding box is implicitly given as TODO.

cell_size_x_si
: Width of individual cell in X direction

cell_size_y_si
: Width of individual cell in Y direction

cell_size_z_si
: Width of individual cell in Z direction

cell_cnt_x
: total number of cells in X direction

cell_cnt_y
: total number of cells in Y direction

cell_cnt_z
: total number of cells in Z direction

boundary_condition_x
: behavior towards particles crossing the X boundary

boundary_condition_y
: behavior towards particles crossing the Y boundary

boundary_condition_z
: behavior towards particles crossing the Z boundary

n_gpus
: number of GPUs in x y and z direction as 3-integer tuple

get_rendering_context() → dict
: get rendering context representation of this object

  delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None
: check if the given context is valid for the given type

  Raises on error, passes silently if okay.

  Raises

  - **ValidationError** – on schema violiation
  - **RuntimeError** – on schema not found

53.2.2 picongpu.pypicongpu.laser

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre, Alexander Debus License: GPLv3+

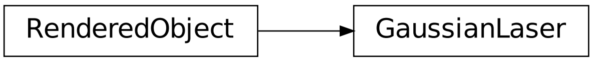
Module Contents

Classes

GaussianLaser	PIConGPU Gaussian Laser

class picongpu.pypicongpu.laser.**GaussianLaser**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*

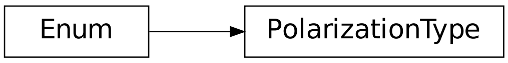


PIConGPU Gaussian Laser

Holds Parameters to specify a gaussian laser

class PolarizationType(**args*, ***kwds*)

Bases: `enum.Enum`



represents a polarization of a laser (for PIConGPU)

LINEAR = 1

CIRCULAR = 2

get_cpp_str() → str

retrieve name as used in c++ param files

__repr__()

Return repr(self).

__str__()

Return str(self).

__dir__()

Returns all members and all public methods

__format__(*format_spec*)

Default object formatter.

__hash__()

Return hash(self).

__reduce_ex__(*proto*)

Helper for pickle.

`__deepcopy__`(*memo*)

`__copy__`()

`name`()
The name of the Enum member.

`value`()
The value of the Enum member.

`wavelength`
wave length in m

`waist`
beam waist in m

`duration`
duration in s (1 sigma)

`focus_pos`
focus position vector in m

`phase`
phase in rad, periodic in 2*pi

`E0`
E0 in V/m

`pulse_init`
laser will be initialized pulse_init times of duration (unitless)

`propagation_direction`
propagation direction(normalized vector)

`polarization_type`
laser polarization

`polarization_direction`
direction of polarization(normalized vector)

`laguerre_modes`
array containing the magnitudes of radial Laguerre-modes

`laguerre_phases`
array containing the phases of radial Laguerre-modes

`huygens_surface_positions`
Position in cells of the Huygens surface relative to start/ edge(negative numbers) of the total domain

`get_rendering_context`() → dict
get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static` **`check_context_for_type`**(*type_to_check: type*, *context: dict*) → None
check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **`ValidationError`** – on schema violiation
- **`RuntimeError`** – on schema not found

53.2.3 `picongpu.pypicongpu.runner`

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre, Richard Pausch License: GPLv3+

Module Contents

Classes

Runner	Accepts a PyPIConGPU Simulation and runs it

Functions

runArgs(name, args)	
get_tmpdir_with_name(name[, parent])	returns a not existing temporary directory path,

`picongpu.pypicongpu.runner.`**`runArgs`**`(`*name*, *args*`)`

`picongpu.pypicongpu.runner.`**`get_tmpdir_with_name`**`(`*name*, *parent: str = None*`)`

returns a not existing temporary directory path, which contains the given name :param name: part of the newly created directory name :param parent: if given: create the tmpdir there :return: not existing path to directory

`class picongpu.pypicongpu.runner.`**`Runner`**`(`*sim*, *pypicongpu_template_dir: str | None = None*, *scratch_dir: str | None = None*, *setup_dir: str | None = None*, *run_dir: str | None = None*`)`

Accepts a PyPIConGPU Simulation and runs it

Manages 2 basic parts:
- *where* which data is stored (various `..._dir` options)
- *what* is done (generate, build, run)

Where:
- scratch_dir: (optional) directory where many simulation results can be stored
- run_dir: directory where data for an execution is stored
- setup_dir: directory where data is generated to and the simulation executable is built

These dirs are either copied from params or guessed. See __init__() for a detailed description.

The initialization of the dirs happens only once (!) inside __init__(). Any changes performed after that will be accepted and might lead to broken builds.

What:
- generate(): create a setup (directory) which represents the parameters given
- build(): run pic-build
- run(): run tbg

Typically these can only be performed in that order, and each once. Whether a step can be started is determined by some sanity checks: Are the inputs (e.g. the setup dir, the `.build` dir) ready, and is the output location empty (e.g. the run dir). **If those sanity checks pass, the respective process is launched.** If this launched program (e.g. pic-build) fails, the process output (stdout & stderr) is printed. While a process is running, all output is silenced (and collected into an internal buffer).

`SCRATCH_ENV_NAME = 'SCRATCH'`

name of the environment variable where the scratch dir defaults to

setup_dir
directory containing the experiment setup (scenario)

pic-build is called here

scratch_dir
directory where run directories can be store

run_dir
directory where run results (and a copy of the inputs) will be stored

sim
the picongpu simulation to be run

generate(*printDirToConsole=False*)
generate the picongpu-compatible input files

build()
build (compile) picongpu-compatible input files

run()
run compiled picongpu simulation

53.2.4 picongpu.pypicongpu.simulation

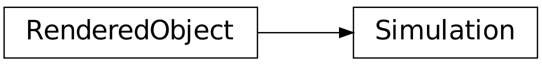
This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

Module Contents

Classes

Simulation	Represents all parameters required to build & run a PIConGPU simulation.

class picongpu.pypicongpu.simulation.**Simulation**
Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Represents all parameters required to build & run a PIConGPU simulation.

Most of the individual parameters are delegated to other objects held as attributes.

To run a Simulation object pass it to the Runner (for details see there).

delta_t_si
Width of a single timestep, given in seconds.

time_steps
Total number of time steps to be executed.

grid
: Used grid Object

laser
: Used (gaussian) Laser

solver
: Used Solver

init_manager
: init manager holding all species & their information

typical_ppc
: typical number of macro particles spawned per cell, >=1

  used for normalization of units

get_rendering_context() → dict
: get rendering context representation of this object

  delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static **check_context_for_type**(*type_to_check: type*, *context: dict*) → None
: check if the given context is valid for the given type

  Raises on error, passes silently if okay.

  Raises

  - **ValidationError** – on schema violiation
  - **RuntimeError** – on schema not found

53.2.5 picongpu.pypicongpu.solver

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

Module Contents

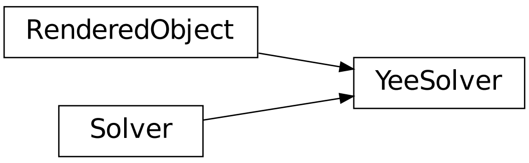
Classes

Solver	represents a field solver
YeeSolver	Yee solver as defined by PIConGPU

class picongpu.pypicongpu.solver.**Solver**
: represents a field solver

  Parent class for type safety, does not contain anything.

class picongpu.pypicongpu.solver.**YeeSolver**
: Bases: *Solver*, *picongpu.pypicongpu.rendering.RenderedObject*



Yee solver as defined by PIConGPU

note: has no parameters

get_rendering_context() → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

static check_context_for_type(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- **ValidationError** – on schema violiation
- **RuntimeError** – on schema not found

53.2.6 picongpu.pypicongpu.util

This file is part of the PIConGPU. Copyright 2021-2023 PIConGPU contributors Authors: Hannes Troepgen, Brian Edward Marre License: GPLv3+

Module Contents

Functions

build_typesafe_property(], name)	
unsupported(→ None)	Print a msg that the feature/parameter/thing is unsupported.

Attributes

attr_cnt	

`picongpu.pypicongpu.util.`**`attr_cnt`**` = 0`

`picongpu.pypicongpu.util.`**`build_typesafe_property`**`(`*type_: Union[type, type(typing.List[int])], name: str | None = None*`)` → property

`picongpu.pypicongpu.util.`**`unsupported`**`(`*name: str, value: Any = 1, default: Any = None*`)` → None

Print a msg that the feature/parameter/thing is unsupported.

If 2nd param (value) and 3rd param (default) are set: supress msg if value == default

If 2nd param (value) is set and 3rd is missing: supress msg if value is None

If only 1st param (name) is set: always print msg

Parameters

- **`name`** – name of the feature/parameter/thing that is unsupported
- **`value`** – If set: only print warning if this is not None
- **`default`** – If set: check value against this param instead of none

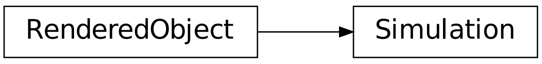
53.3 Package Contents

53.3.1 Classes

Simulation	Represents all parameters required to build & run a PIConGPU simulation.
Runner	Accepts a PyPIConGPU Simulation and runs it

class `picongpu.pypicongpu.`**`Simulation`**

Bases: *picongpu.pypicongpu.rendering.RenderedObject*



Represents all parameters required to build & run a PIConGPU simulation.

Most of the individual parameters are delegated to other objects held as attributes.

To run a Simulation object pass it to the Runner (for details see there).

`delta_t_si`

Width of a single timestep, given in seconds.

`time_steps`

Total number of time steps to be executed.

`grid`

Used grid Object

`laser`

Used (gaussian) Laser

`solver`

Used Solver

`init_manager`

init manager holding all species & their information

`typical_ppc`

typical number of macro particles spawned per cell, >=1

used for normalization of units

`get_rendering_context()` → dict

get rendering context representation of this object

delegates work to _get_serialized and invokes checks performed by check_context_for_type(). :raise ValidationError: on schema violiation :raise RuntimeError: on schema not found :return: self as rendering context

`static check_context_for_type`(*type_to_check: type*, *context: dict*) → None

check if the given context is valid for the given type

Raises on error, passes silently if okay.

Raises

- `ValidationError` – on schema violiation
- `RuntimeError` – on schema not found

`class picongpu.pypicongpu.Runner`(*sim*, *pypicongpu_template_dir: str | None = None*, *scratch_dir: str | None = None*, *setup_dir: str | None = None*, *run_dir: str | None = None*)

Accepts a PyPIConGPU Simulation and runs it

Manages 2 basic parts:

- *where* which data is stored (various `..._dir` options)
- *what* is done (generate, build, run)

Where:

- scratch_dir: (optional) directory where many simulation results can be stored
- run_dir: directory where data for an execution is stored
- setup_dir: directory where data is generated to and the simulation executable is built

These dirs are either copied from params or guessed. See __init__() for a detailed description.

The initialization of the dirs happens only once (!) inside __init__(). Any changes performed after that will be accepted and might lead to broken builds.

What:

- generate(): create a setup (directory) which represents the parameters given
- build(): run pic-build
- run(): run tbg

Typically these can only be performed in that order, and each once. Whether a step can be started is determined by some sanity checks: Are the inputs (e.g. the setup dir, the `.build` dir) ready, and is the output location empty (e.g. the run dir). **If those sanity checks pass, the respective process is launched.** If this launched program (e.g. pic-build) fails, the process output (stdout & stderr) is printed. While a process is running, all output is silenced (and collected into an internal buffer).

`SCRATCH_ENV_NAME = 'SCRATCH'`

name of the environment variable where the scratch dir defaults to

`setup_dir`
  directory containing the experiment setup (scenario)

  pic-build is called here

`scratch_dir`
  directory where run directories can be store

`run_dir`
  directory where run results (and a copy of the inputs) will be stored

`sim`
  the picongpu simulation to be run

`generate`(*printDirToConsole=False*)
  generate the picongpu-compatible input files

`build`()
  build (compile) picongpu-compatible input files

`run`()
  run compiled picongpu simulation

BIBLIOGRAPHY

[modules] J.L. Furlani, P.W. Osel. *Abstract Yourself With Modules*, Proceedings of the 10th USENIX conference on System administration (1996), http://modules.sourceforge.net

[Lmod] R. McLay and contributors. *Lmod: An Environment Module System based on Lua, Reads TCL Modules, Supports a Software Hierarchy*, https://github.com/TACC/Lmod

[CMake] Kitware Inc. *CMake: Cross-platform build management tool*, https://cmake.org/

[Burau2010] Burau, H., et al., *A fully relativistic particle-in-cell code for a GPU cluster.*, IEEE Transactions on Plasma Science, 38(10 PART 2), 2831–2839 (2010), https://doi.org/10.1109/TPS.2010.2064310

[Zuehl2011] Zühl, L., *Visualisierung von Laser-Plasma-Simulationen*, Diploma Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Informatiker" (2011), https://www.hzdr.de/db/Cms?pOid=35687

[Schneider2012a] Schneider, B., *Gestengesteuerte visuelle Datenanalyse einer Laser-Plasma-Simulation*, Student Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2012), https://www.hzdr.de/db/Cms?pOid=37242

[Schneider2012b] Schneider, B., *In Situ Visualization of a Laser-Plasma Simulation*, Diploma Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Informatiker" (2012), https://www.hzdr.de/db/Cms?pOid=40353

[Pausch2012] Pausch, R., *Electromagnetic Radiation from Relativistic Electrons as Characteristic Signature of their Dynamics*, Diploma Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2012), https://doi.org/10.5281/zenodo.843510

[Ungethuem2012] Ungethüm, A., *Simulation and visualisation of the electro-magnetic field around a stimulated electron*, Student Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2012), https://www.hzdr.de/db/Cms?pOid=38508

[Bussmann2013] Bussmann, M. et al., *Radiative signatures of the relativistic Kelvin-Helmholtz instability*, SC '13 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 5-1-5–12), https://doi.org/10.1145/2503210.2504564

[Huebl2014] Huebl, A. et al., *Visualizing the Radiation of the Kelvin-Helmholtz Instability*, IEEE Transactions on Plasma Science 42.10 (2014), https://doi.org/10.1109/TPS.2014.2327392

[Pausch2014] Pausch, R., Debus, A., Widera, R. et al., *How to test and verify radiation diagnostics simulations within particle-in-cell frameworks*, Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 740, 250–256 (2014), https://doi.org/10.1016/j.nima.2013.10.073

[Huebl2014] Huebl, A., *Injection Control for Electrons in Laser-Driven Plasma Wakes on the Femtosecond Time Scale*, Diploma Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2014), https://doi.org/10.5281/zenodo.15924

[Garten2015] Garten, M., *Modellierung und Validierung von Feldionisation in parallelen Particle-in-Cell-Codes*, Master Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2015), https://doi.org/10.5281/zenodo.202500

[Burau2016] Burau, H., *Entwicklung und Überprüfung eines Photonenmodells für die Abstrahlung durch hochenergetische Elektronen*, Diploma Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2016), https://doi.org/10.5281/zenodo.192116

[Matthes2016] Matthes, A., *In-Situ Visualisierung und Streaming von Plasmasimulationsdaten*, Diploma Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Informatiker" (2016), https://doi.org/10.5281/zenodo.55329

[Matthes2016b] Matthes, A., Huebl, A., Widera, R., Grottel, S., Gumhold, S., Bussmann, M., *In situ, steerable, hardware-independent and data-structure agnostic visualization with ISAAC*, Supercomputing Frontiers and Innovations, [S.l.], v. 3, n. 4, p. 30-48, oct. 2016, https://doi.org/10.14529/jsfi160403

[Pausch2017] Pausch, R., Bussmann, M., Huebl, A., Schramm, U., Steiniger, K., Widera, R. and Debus, A., *Identifying the linear phase of the relativistic Kelvin-Helmholtz instability and measuring its growth rate via radiation*, Phys. Rev. E 96, 013316 - Published 26 July 2017, https://doi.org/10.1103/PhysRevE.96.013316

[Couperus2017] Couperus, J. P. et al., *Demonstration of a beam loaded nanocoulomb-class laser wakefield accelerator*, Nature Communications volume 8, Article number: 487 (2017), https://doi.org/10.1038/s41467-017-00592-7

[Huebl2017] Huebl, A. et al., *On the Scalability of Data Reduction Techniques in Current and Upcoming HPC Systems from an Application Perspective*, ISC High Performance Workshops 2017, LNCS 10524, pp. 15-29 (2017), https://doi.org/10.1007/978-3-319-67630-2_2

[Obst2017] Obst, L., Göde, S., Rehwald, M. et al., *Efficient laser-driven proton acceleration from cylindrical and planar cryogenic hydrogen jets*, Sci Rep 7, 10248 (2017), https://doi.org/10.1038/s41598-017-10589-3

[Pausch2018] Pausch, R., Debus, A., Huebl, A. at al., *Quantitatively consistent computation of coherent and incoherent radiation in particle-in-cell codes — A general form factor formalism for macro-particles*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 909, 419–422 (2018), https://doi.org/10.1016/j.nima.2018.02.020

[Couperus2018] Couperus, J. P., *Optimal beam loading in a nanocoulomb-class laser wakefield accelerator*, PhD Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2018), https://doi.org/10.5281/zenodo.1463710

[Meyer2018] Meyer, F., *Entwicklung eines Partikelvisualisierers für In-Situ-Simulationen*, Bachelor Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2018), https://doi.org/10.5281/zenodo.1423296

[Hilz2018] Hilz, P, et al., *Isolated proton bunch acceleration by a petawatt laser pulse*, Nature Communications, 9(1), 423 (2018), https://doi.org/10.1038/s41467-017-02663-1

[Obst-Huebl2018] Obst-Huebl, L., Ziegler, T., Brack, FE. et al., *All-optical structuring of laser-driven proton beam profiles*, Nat Commun 9, 5292 (2018), https://doi.org/10.1038/s41467-018-07756-z

[Rudat2019] Rudat, S., *Laser Wakefield Acceleration Simulation as a Service*, Master Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3529741

[Pausch2019] Pausch, R., *Synthetic radiation diagnostics as a pathway for studying plasma dynamics from advanced accelerators to astrophysical observations*, PhD Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3616045

[Koehler2019] Köhler, A., *Transverse Electron Beam Dynamics in the Beam Loading Regime*, PhD Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3342589

[Zarini2019] Zarini, O., *Measuring sub-femtosecond temporal structures in multi-ten kiloampere electron beams*, PhD Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://nbn-resolving.org/urn:nbn:de:bsz:d120-qucosa2-339775

[Obst-Huebl2019] Obst-Hübl, L., *Achieving optimal laser-proton acceleration through multi-parameter interaction control*, PhD Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3252952

[Huebl2019] Huebl, A., *PIConGPU: Predictive Simulations of Laser-Particle Accelerators with Manycore Hardware*, PhD Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3266820

[Carstens2019] Carstens, F.-O., *Synthetic characterization of ultrashort electron bunches using transition radiation*, Bachelor Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3469663

[Ostermayr2020] Ostermayr, T. M., et al., *Laser-driven x-ray and proton micro-source and application to simultaneous single-shot bi-modal radiographic imaging*, Nature Communications volume 11, Article number: 6174 (2020), https://doi.org/10.1038/s41467-020-19838-y

[Huebl2020] Huebl, A. et al., *Spectral control via multi-species effects in PW-class laser-ion acceleration*, Plasma Phys. Control. Fusion 62 124003 (2020), https://doi.org/0.1088/1361-6587/abbe33

[Perera2020] Perera, A. et al., *Towards ultra-high gradient particle acceleration in carbon nanotubes*, Journal of Physics: Conference Series 1596 012028 (2020), https://doi.org/10.1088/1742-6596/1596/1/012028

[Kurz2021] Kurz, T. et al., *Demonstration of a compact plasma accelerator powered by laser-accelerated electron beams*, Nature Communications volume 12, Article number: 2895 (2021), https://doi.org/10.1038/s41467-021-23000-7

[Koehler2021] Koehler, A., Pausch, R., Bussmann, M., et al., *Restoring betatron phase coherence in a beam-loaded laser-wakefield accelerator*, Phys. Rev. Accel. Beams 24, 091302 – 20 September 2021, https://doi.org/10.1103/PhysRevAccelBeams.24.091302

[Bontoiu2023] Bontoiu, C., et al., *TeV/m catapult acceleration of electrons in graphene layers*, Scientific Reports volume 13, Article number: 1330 (2023), https://doi.org/10.1038/s41598-023-28617-w

[ClementiRaimondi1963] Clementi, E. and Raimondi, D., *Atomic Screening Constant from SCF Functions*, The Journal of Chemical Physics 38, 2686-2689 (1963), https://dx.doi.org/10.1063/1.1733573

[Keldysh] Keldysh, L.V., *Ionization in the field of a strong electromagnetic wave*, Soviet Physics JETP 20, 1307-1314 (1965), http://jetp.ac.ru/cgi-bin/dn/e_020_05_1307.pdf

[Yee1966] Yee, K., *Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media*, IEEE Transactions on Antennas and Propagation (Volume: 14, Issue: 3, May 1966), https://doi.org/10.1109/TAP.1966.1138693

[ClementiRaimondi1967] Clementi, E. and D. Raimondi, D., *Atomic Screening Constant from SCF Functions. II. Atoms with 37 to 86 Electrons*, The Journal of Chemical Physics 47, 1300-1307 (1967), https://dx.doi.org/10.1063/1.1712084

[Boris1970] Boris, J., *Relativistic Plasma Simulation - Optimization of a Hybrid Code*, Proc. 4th Conf. on Num. Sim. of Plasmas (1970), http://www.dtic.mil/docs/citations/ADA023511

[More1985] R. M. More. *Pressure Ionization, Resonances, and the Continuity of Bound and Free States*, Advances in Atomic, Molecular and Optical Physics Vol. 21 C, 305-356 (1985), https://dx.doi.org/10.1016/S0065-2199(08)60145-1

[Mulser1998] Mulser, P. et al., *Modeling field ionization in an energy conserving form and resulting nonstandard fluid dynamcis*, Physics of Plasmas 5, 4466 (1998), https://doi.org/10.1063/1.873184

[DeloneKrainov1998] Delone, N. B. and Krainov, V. P., *Tunneling and barrier-suppression ionization of atoms and ions in a laser radiation field*, Phys. Usp. 41 469–485 (1998), http://dx.doi.org/10.1070/PU1998v041n05ABEH000393

[BauerMulser1999] Bauer, D. and Mulser, P., *Exact field ionization rates in the barrier-suppression regime from numerical time-dependent Schrödinger-equation calculations*, Physical Review A 59, 569 (1999), https://dx.doi.org/10.1103/PhysRevA.59.569

[Ghrist2000] M. Ghrist, *High-Order Finite Difference Methods for Wave Equations*, PhD thesis (2000), Department of Applied Mathematics, University of Colorado

[Esirkepov2001] Esirkepov, T. Zh., *Exact charge conservation scheme for Particle-in-Cell simulation with an arbitrary form-factor*, Computer Physics Communications, Volume 135, Issue 2, 1 April 2001, Pages 144-153, https://doi.org/10.1016/S0010-4655(00)00228-9

[FLYCHK2005] *FLYCHK: Generalized population kinetics and spectral model for rapid spectroscopic analysis for all elements*, H.-K. Chung, M.H. Chen, W.L. Morgan, Yu. Ralchenko, and R.W. Lee, *High Energy Density Physics* v.1, p.3 (2005) http://nlte.nist.gov/FLY/

[Vay2008] Vay, J., *Simulation of beams or plasmas crossing at relativistic velocity*, Physics of Plasmas 15, 056701 (2008), https://doi.org/10.1063/1.2837054

[MulserBauer2010] Mulser, P. and Bauer, D., *High Power Laser-Matter Interaction*, Springer-Verlag Berlin Heidelberg (2010), https://dx.doi.org/10.1007/978-3-540-46065-7

[Perez2012] Pérez, F., Gremillet, L., Decoster, A., et al., *Improved modeling of relativistic collisions and collisional ionization in particle-in-cell codes*, Physics of Plasmas 19, 083104 (2012), https://doi.org/10.1063/1.4742167

[Lehe2013] Lehe, R., Lifschitz, A., Thaury, C., Malka, V. and Davoine, X., *Numerical growth of emittance in simulations of laser-wakefield acceleration*, Phys. Rev. ST Accel. Beams 16, 021301 – Published 28 February 2013, https://doi.org/10.1103/PhysRevSTAB.16.021301

[Gonoskov2015] Gonoskov, A., Bastrakov, S., Efimenko, E., et al., *Extended particle-in-cell schemes for physics in ultrastrong laser fields: Review and developments*, Phys. Rev. E 92, 023305 – Published 18 August 2015, https://doi.org/10.1103/PhysRevE.92.023305

[Vranic2016] Vranic, M., et al., *Classical radiation reaction in particle-in-cell simulations*, Computer Physics Communications, Volume 204, July 2016, Pages 141-151, https://doi.org/10.1016/j.cpc.2016.04.002

[Higuera2017] Higuera, A. V. and Cary, J. R., *Structure-preserving second-order integration of relativistic charged particle trajectories in electromagnetic fields*, Physics of Plasmas 24, 052104 (2017), https://doi.org/10.1063/1.4979989

[Higginson2020] Higginson, D. P. , Holod, I. , and Link, A., *A corrected method for Coulomb scattering in arbitrarily weighted particle-in-cell plasma simulations*, Journal of Computational Physics 413, 109450 (2020). https://doi.org/10.1016/j.jcp.2020.109450

[Matthes2016] A. Matthes, A. Huebl, R. Widera, S. Grottel, S. Gumhold, and M. Bussmann *In situ, steerable, hardware-independent and data-structure agnostic visualization with ISAAC*, Supercomputing Frontiers and Innovations 3.4, pp. 30-48, (2016), arXiv:1611.09048, DOI:10.14529/jsfi160403

[Huebl2014] A. Huebl. *Injection Control for Electrons in Laser-Driven Plasma Wakes on the Femtosecond Time Scale*, chapter 3.2, Diploma Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2014), https://doi.org/10.5281/zenodo.15924

[Pausch2012] Pausch, R., *Electromagnetic Radiation from Relativistic Electrons as Characteristic Signature of their Dynamics*, Diploma Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2012), https://doi.org/10.5281/zenodo.843510

[Pausch2014] Pausch, R., Debus, A., Widera, R. et al., *How to test and verify radiation diagnostics simulations within particle-in-cell frameworks*, Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 740, 250–256 (2014), https://doi.org/10.1016/j.nima.2013.10.073

[Pausch2018] Pausch, R., Debus, A., Huebl, A. at al., *Quantitatively consistent computation of coherent and incoherent radiation in particle-in-cell codes — A general form factor formalism for macro-particles*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 909, 419–422 (2018), https://doi.org/10.1016/j.nima.2018.02.020

[Pausch2019] Pausch, R., *Synthetic radiation diagnostics as a pathway for studying plasma dynamics from advanced accelerators to astrophysical observations*, PhD Thesis at Technische Universität Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3616045

[Rudat2019] S. Rudat. *Laser Wakefield Acceleration Simulation as a Service*, chapter 4.4, Master's thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), https://doi.org/10.5281/zenodo.3529741

[Huebl2014] A. Huebl. *Injection Control for Electrons in Laser-Driven Plasma Wakes on the Femtosecond Time Scale*, Diploma Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2014), DOI:10.5281/zenodo.15924

[Matthes2016] A. Matthes, A. Huebl, R. Widera, S. Grottel, S. Gumhold, and M. Bussmann *In situ, steerable, hardware-independent and data-structure agnostic visualization with ISAAC*, Supercomputing Frontiers and Innovations 3.4, pp. 30-48, (2016), arXiv:1611.09048, DOI:10.14529/jsfi160403

[Huebl2017] A. Huebl, R. Widera, F. Schmitt, A. Matthes, N. Podhorszki, J.Y. Choi, S. Klasky, and M. Bussmann. *On the Scalability of Data Reduction Techniques in Current and Upcoming HPC Systems from an Application Perspective.* ISC High Performance Workshops 2017, LNCS 10524, pp. 15-29 (2017), arXiv:1706.00522, DOI:10.1007/978-3-319-67630-2_2

[Pausch2012] R. Pausch. *Electromagnetic Radiation from Relativistic Electrons as Characteristic Signature of their Dynamics*, Diploma Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2012), DOI:10.5281/zenodo.843510

[Pausch2014] R. Pausch, A. Debus, R. Widera, K. Steiniger, A.Huebl, H. Burau, M. Bussmann, and U. Schramm. *How to test and verify radiation diagnostics simulations within particle-in-cell frameworks*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 740, pp. 250-256 (2014) DOI:10.1016/j.nima.2013.10.073

[Pausch2018] R. Pausch, A. Debus, A. Huebl, U. Schramm, K. Steiniger, R. Widera, and M. Bussmann. *Quantitatively consistent computation of coherent and incoherent radiation in particle-in-cell codes - a general form factor formalism for macro-particles*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 909, pp. 419-422 (2018) arXiv:1802.03972, DOI:10.1016/j.nima.2018.02.020

[Carstens2022] F.-O. Carstens, *Synthetic few-cycle shadowgraphy diagnostics in particle-in-cell codes for characterizing laser-plasma accelerators* Master Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2022) **`DOI:10.5281/zenodo.7755263<https://doi.org/10.5281/zenodo.7755263>`_**

[PauschDipl] Richard Pausch. *Electromagnetic Radiation from Relativistic Electrons as Characteristic Signature of their Dynamics*, Diploma Thesis TU Dresden (2012), https://www.hzdr.de/db/Cms?pOid=38997

[Pausch13] R. Pausch, A. Debus, R. Widera, K. Steiniger, A. Huebl, H. Burau, M. Bussmann, U. Schramm. *How to test and verify radiation diagnostics simulations within particle-in-cell frameworks*, Nuclear Instruments and Methods in Physics Research Section A (2013), http://dx.doi.org/10.1016/j.nima.2013.10.073

[Esarey93] E. Esarey, S. Ride, P. Sprangle. *Nonlinear Thomson scattering of intense laser pulses from beams and plasmas*, Physical Review E (1993), http://dx.doi.org/10.1103/PhysRevE.48.3003

[BirdsallLangdon] C.K. Birdsall, A.B. Langdon. *Plasma Physics via Computer Simulation*, McGraw-Hill (1985), ISBN 0-07-005371-5

[HockneyEastwood] R.W. Hockney, J.W. Eastwood. *Computer Simulation Using Particles*, CRC Press (1988), ISBN 0-85274-392-0

[TNSA] S.C. Wilks, A.B. Langdon, T.E. Cowan, M. Roth, M. Singh, S. Hatchett, M.H. Key, D. Pennington, A. MacKinnon, and R.A. Snavely. *Energetic proton generation in ultra-intense laser-solid interactions*, Physics of Plasmas **8**, 542 (2001), https://dx.doi.org/10.1063/1.1333697

[LCT] P.L. Poole, L. Obst, G.E. Cochran, J. Metzkes, H.-P. Schlenvoigt, I. Prencipe, T. Kluge, T.E. Cowan, U. Schramm, and D.W. Schumacher. *Laser-driven ion acceleration via target normal sheath acceleration in the relativistic transparency regime*, New Journal of Physics **20**, 013019 (2018), https://dx.doi.org/10.1088/1367-2630/aa9d47

[Alves12] E.P. Alves, T. Grismayer, S.F. Martins, F. Fiuza, R.A. Fonseca, L.O. Silva. *Large-scale magnetic field generation via the kinetic kelvin-helmholtz instability in unmagnetized scenarios*, The Astrophysical Journal Letters (2012), https://dx.doi.org/10.1088/2041-8205/746/2/L14

[Grismayer13] T. Grismayer, E.P. Alves, R.A. Fonseca, L.O. Silva. *dc-magnetic-field generation in unmagnetized shear flows*, Physical Reveview Letters (2013), https://doi.org/10.1103/PhysRevLett.111.015005

[Bussmann13] M. Bussmann, H. Burau, T.E. Cowan, A. Debus, A. Huebl, G. Juckeland, T. Kluge, W.E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, R. Widera. *Radiative Signatures of the Relativistic Kelvin-Helmholtz Instability*, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (2013), http://doi.acm.org/10.1145/2503210.2504564

[TajimaDawson] T. Tajima, J.M. Dawson. *Laser electron accelerator*, Physical Review Letters (1979), https://dx.doi.org/10.1103/PhysRevLett.43.267

[Modena] A. Modena, Z. Najmudin, A.E. Dangor, C.E. Clayton, K.A. Marsh, C. Joshi, V. Malka, C. B. Darrow, C. Danson, D. Neely, F.N. Walsh. *Electron acceleration from the breaking of relativistic plasma waves*, Nature (1995), https://dx.doi.org/10.1038/377606a0

[PukhovMeyerterVehn] A. Pukhov and J. Meyer-ter-Vehn. *Laser wake field acceleration: the highly non-linear broken-wave regime*, Applied Physics B (2002), https://dx.doi.org/10.1007/s003400200795

[Schroeder2004] Schroeder, C. B. and Esarey, E. and van Tilborg, J. and Leemans, W. P. *Theory of coherent transition radiation generated at a plasma-vacuum interface*, American Physical Society(2004), https://link.aps.org/doi/10.1103/PhysRevE.69.016501

[Downer2018] Downer, M. C. and Zgadzaj, R. and Debus, A. and Schramm, U. and Kaluza, M. C. *Diagnostics for plasma-based electron accelerators*, American Physical Society(2018), https://link.aps.org/doi/10.1103/RevModPhys.90.035002

[EulerLagrangeFrameOfReference] *Eulerian and Lagrangian specification of the flow field.* https://en.wikipedia.org/wiki/Lagrangian_and_Eulerian_specification_of_the_flow_field

[BirdsallLangdon] C.K. Birdsall, A.B. Langdon. *Plasma Physics via Computer Simulation*, McGraw-Hill (1985), ISBN 0-07-005371-5

[HockneyEastwood] R.W. Hockney, J.W. Eastwood. *Computer Simulation Using Particles*, CRC Press (1988), ISBN 0-85274-392-0

[Huebl2014] A. Huebl. *Injection Control for Electrons in Laser-Driven Plasma Wakes on the Femtosecond Time Scale*, Diploma Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf for the German Degree "Diplom-Physiker" (2014), DOI:10.5281/zenodo.15924

[Huebl2019] A. Huebl. *PIConGPU: Predictive Simulations of Laser-Particle Accelerators with Manycore Hardware*, PhD Thesis at TU Dresden & Helmholtz-Zentrum Dresden - Rossendorf (2019), DOI:10.5281/zenodo.3266820

[Esirkepov2001] T.Zh. Esirkepov, *Exact charge conservation scheme for particle-in-cell simulation with an arbitrary form-factor*, Computer Physics Communications 135.2 (2001): 144-153, https://doi.org/10.1016/S0010-4655(00)00228-9

[Ghrist2000] M. Ghrist, *High-Order Finite Difference Methods for Wave Equations*, PhD thesis (2000), Department of Applied Mathematics, University of Colorado

[Lehe2012] R. Lehe et al. *Numerical growth of emittance in simulations of laser-wakefield acceleration*, Physical Review Special Topics-Accelerators and Beams 16.2 (2013): 021301.

[Taflove2005] A. Taflove, S.C. Hagness *Computational electrodynamics: the finite-difference time-domain method* Artech house (2005)

[Yee1966] K.S. Yee, *Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media*, IEEE Trans. Antennas Propagat. 14, 302-307 (1966)

[Potter2017] M. Potter, J.-P. Berenger *A Review of the Total Field/Scattered Field Technique for the FDTD Method* FERMAT, Volume 19, Article 1 (2017)

[Taflove2005] A. Taflove, S.C. Hagness *Computational electrodynamics: the finite-difference time-domain method* Artech house (2005)

[Harrington2001] R.F. Harrington *Time-Harmonic Electromagnetic Fields* McGraw-Hill (2001)

[Balanis2012] C.A. Balanis *Advanced Engineering Electromagnetics* John Wiley & Sons (2012)

[Rengarajan2000] S.R. Rengarajan, Y. Rahmat-Samii *The field equivalence principle: illustration of the establishment of the non-intuitive null fields* IEEE Antennas and Propagation Magazine, Volume 42, No. 4 (2000)

[HockneyEastwood] R.W. Hockney, J.W. Eastwood. *Computer Simulation Using Particles*, CRC Press (1988), ISBN 0-85274-392-0

[Vranic2016] M. Vranic, J.L. Martins, R.A. Fonseca, L.O. Silva. *Classical radiation reaction in particle-in-cell simulations*, Computer Physics Communications 204, 114-151 (2016), https://dx.doi.org/10.1016/j.cpc.2016.04.002

[DeloneKrainov] N. B. Delone and V. P. Krainov. *Tunneling and barrier-suppression ionization of atoms and ions in a laser radiation field*, Phys. Usp. 41 469–485 (1998), http://dx.doi.org/10.1070/PU1998v041n05ABEH000393

[BauerMulser1999] D. Bauer and P. Mulser. *Exact field ionization rates in the barrier-suppression regime from numerical time-dependent Schrödinger-equation calculations*, Physical Review A 59, 569 (1999), https://dx.doi.org/10.1103/PhysRevA.59.569

[MulserBauer2010] P. Mulser and D. Bauer. *High Power Laser-Matter Interaction*, Springer-Verlag Berlin Heidelberg (2010), https://dx.doi.org/10.1007/978-3-540-46065-7

[Keldysh] L.V. Keldysh. *Ionization in the field of a strong electromagnetic wave*, Soviet Physics JETP 20, 1307-1314 (1965), http://jetp.ac.ru/cgi-bin/dn/e_020_05_1307.pdf

[ClementiRaimondi1963] E. Clementi and D. Raimondi. *Atomic Screening Constant from SCF Functions*, The Journal of Chemical Physics 38, 2686-2689 (1963) https://dx.doi.org/10.1063/1.1733573

[ClementiRaimondi1967] E. Clementi and D. Raimondi. *Atomic Screening Constant from SCF Functions. II. Atoms with 37 to 86 Electrons*, The Journal of Chemical Physics 47, 1300-1307 (1967) https://dx.doi.org/10.1063/1.1712084

[Mulser] P. Mulser et al. *Modeling field ionization in an energy conserving form and resulting nonstandard fluid dynamcis*, Physics of Plasmas 5, 4466 (1998) https://doi.org/10.1063/1.873184

[More1985] R. M. More. *Pressure Ionization, Resonances, and the Continuity of Bound and Free States*, Advances in Atomic, Molecular and Optical Physics Vol. 21 C, 305-356 (1985), https://dx.doi.org/10.1016/S0065-2199(08)60145-1

[FLYCHK2005] *FLYCHK: Generalized population kinetics and spectral model for rapid spectroscopic analysis for all elements*, H.-K. Chung, M.H. Chen, W.L. Morgan, Yu. Ralchenko, and R.W. Lee, *High Energy Density Physics* v.1, p.3 (2005) http://nlte.nist.gov/FLY/

[ClangTools] Online (2017), https://clang.llvm.org/docs/ClangTools.html

PYTHON MODULE INDEX

INDEX