mppnccombine-fast Documentation

Scott Wales

Jan 10, 2019

Contents:

`mppnccombine-fast` is an optimised implementation of the MOM tool `mppnccombine` for use with high-resolution NetCDF4-compressed datasets

CHAPTER 1

Installation

For installation instructions see the README at https://github.com/coecms/mppnccombine-fast

CHAPTER 2

Usage

`mppnccombine-fast` is a MPI program, and requires at least two MPI ranks to run:

```
mpirun -n 2 mppnccombine-fast --output output.nc input.nc.000 input.nc.001
```

Variables in the input files whos dimensions have a `domain_distribution` attribute will be collated. All other dimensions, variables and attributes will be copied from the first input file.

The `domain_distribution` values are expected to be in the format provided by the MOM model - an array of 4 integer values using 1-based array indices:

- First index of this dimension in the full dataset
- Last index of this dimension in the full dataset
- First index of this dimension in this file's data
- Last index of this dimension in this file's data

A `domain_distribution` of `[1, 10, 5, 10]` states that the full dimension has a length of 10, and this file contains the 5 values starting at offset `5`.

2.1 Globbing inputs

Input files may be listed either individually or as an escaped shell glob (both to reduce the `history` attribute in the output file as well as to avoid issues when there are thousands of input files):

```
mpirun -n 2 mppnccombine-fast --output output.nc input.nc.\*
```

2.2 Changing compression settings

Chunk size and compression settings will by default come from the first input file, though they can be overridden using flags. Note that the optimised copying routines can only be used when the compression settings of an input file matches those of the output, and when the input file's data chunks align with the chunks in the output file (e.g. if a variable in the output file has chunk sizes `[10, 15, 30]` then the input file's offset in the full dataset must be `[m*10, n*15, o*30]` where `m`, `n` and `o` are integers).

If only some of the chunks in the input file align with the output these chunks will use the fast path (so partial chunks on the edges of the dataset are fine).

CHAPTER 3

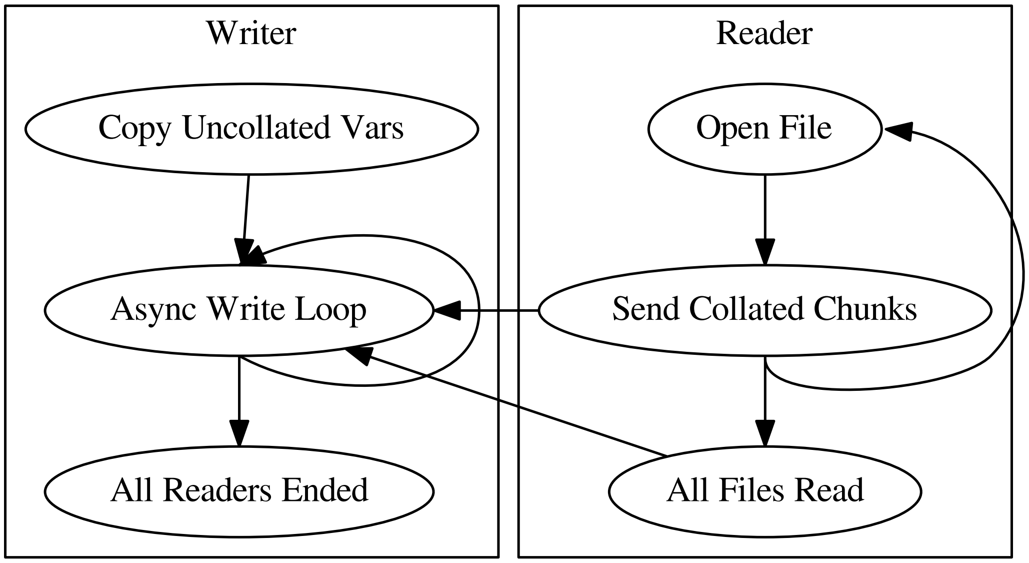
Implementation Overview

The basic outline of `mppnccombine-fast` consists of one "Writer" rank and one or more "Reader" ranks. The Writer rank handles all writing to the output file, while the Reader ranks read in data from the many files to be collated and send the data to the Writer rank.

The main slowdown in copying compressed variables is that the hdf5 library has to de-compress them during the read, and re-compress them during the write. `mppnccombine-fast` works around this by using HDF5 1.10.2's direct IO functions H5DOwrite_chunk() and H5DOread_chunk() to copy the compressed data from one file to the other directly, rather than going through the de-compress/re-compress cycle.

To get a even larger speedup MPI is used to have separate read and write processes, since HDF5 IO is a blocking function.

Since the NetCDF4 library is much nicer to use, but doesn't provide public access to the underlying HDF5 file, we need to do a bit of musical chairs with the files, swapping between NetCDF4 and HDF5 modes by re-opening the files.



3.1 Writer Rank

The Writer starts out by copying the dimensions, attributes and any uncollated variables from the first of the listed input files using the NetCDF API in `init()` and `copy_contiguous()`. It then re-opens the file using the HDF5 API and enters the 'Async Write Loop' in *`run_async_writer()`*.

This loop polls for any incoming MPI messages from the Reader processes then performs some action (e.g. write a compressed chunk directly to the file at some location). Once a Reader has finished reading all of its input files it sends a close message to the Writer rank, once all close messages have been received the Writer rank closes the output file and exits.

3.2 Reader Ranks

The Readers distribute input files amonst themselves using a shared atomic counter. When a Reader is ready for a new file it gets the next value from the counter, then in *`copy_chunked()`* opens that file using NetCDF to query its attributes and discover and copy collated variables.

Depending on the chunking and alignment of the file the Reader will decide to copy the data either in uncompressed form using NetCDF with `copy_netcdf_variable_chunks()` or directly copying the compressed chunks by re-opening the file in HDF5 mode with `copy_hdf5_variable_chunks()`.

Once all available files have been read the Reader sends a final close message to the Writer and exits.

3.3 The Async Write Loop

The Async write loop is set up to handle a number of messages that the Readers will send to the Writer

- `open_variable_async()`: Obtain a handle to a variable in the output file
- `variable_info_async()`: Obtain chunking and compression information for a variable
- `write_uncompressed_async()`: Write uncompressed data to a given logical location in the variable
- `write_chunk_async()`: Write a compressed chunk directly to the output file at a given chunk location
- `close_variable_async()`: Return the variable handle
- `close_async()`: Reports that the Reader will not send any more messages

The Writer asyncronously polls for these messages in `run_async_writer()`, then actions them in `receive_open_variable_async()` etc.

CHAPTER 4

API

4.1 async.h

Contains the async Writer loop and functions that the Reader process can use to communicate with it

4.1.1 Reader side

type varid_t

Remote handle to a variable in the output file on the Writer process

varid_t **open_variable_async** (**const** char **varname*, size_t *len*, int *async_writer_rank*)

Open a variable in the async writer.

Return a handle to the variable in the output file

Parameters

- `varname`: Variable name
- `len`: Length of varname, including the closing '/0'
- `async_writer_rank`: MPI rank of writer process

void **variable_info_async** (*varid_t varid*, size_t *ndims*, size_t *chunk*[], int **deflate*, int **deflate_level*, int **shuffle*, int *async_writer_rank*)

Get info about a variable in the output file.

For direct chunk writes to work the compression parameters must match between the input and output files. Use this function to query the parameters of the output file, then compare the results against the values returned by nc_inq_var_deflate() on the input file.

Parameters

- `varid`: Variable handle obtained with open_variable_async
- `ndims`: Number of dimensions

- `chunk[ndims]`: (out): Chunk shape
- `deflate`: (out): Compression enabled
- `deflate_level`: (out): Compression level
- `shuffle`: (out): Shuffle filter enabled
- `async_writer_rank`: MPI rank of writer process

void **`write_uncompressed_async`** (*varid_t* *varid*, size_t *ndims*, **`const`** size_t *offset*[], **`const`** size_t *shape*[], **`const`** void **buffer*, nc_type *type*, int *async_writer_rank*, MPI_Request **request*)

Write data to the file, using the dataset filters.

This writes uncompressed data, as obtained by nc_get_vara() on the input file. It is slower than direct chunk writes, as the data must be put through a compression filter, but is more flexible as it can be used to write partial or unaligned chunks.

request must be sent to MPI_Wait for the message to complete

Parameters

- `varid`: Variable handle obtained with open_variable_async
- `ndims`: Number of dimensions
- `offset[ndims]`: Offset of this data's origin in the collated dataset
- `shape[ndims]`: Shape of this data array
- `buffer`: Compressed chunk data
- `type`: NetCDF type of the data
- `async_writer_rank`: MPI rank of writer process
- `request`: (out): MPI request for the communication

void **`write_chunk_async`** (*varid_t* *varid*, size_t *ndims*, uint32_t *filter_mask*, **`const`** hsize_t *offset*[], size_t *data_size*, **`const`** void **buffer*, int *async_writer_rank*, MPI_Request **request*)

Write a compressed chunk directly to the file.

This writes compressed chunks, as obtained by opening the input file in HDF5 and calling H5DOread_chunk(). This is faster than copying uncompressed data, but the chunking and compression parameters must be identical on the input and output files, and the chunk must lay on the chunk boundary of the output file. variable_info_async() can be used to determine the chunk layout and compression settings of the variable in the output file.

request must be sent to MPI_Wait for the message to complete

Parameters

- `varid`: Variable handle obtained with open_variable_async
- `ndims`: Number of dimensions
- `filter_mask`: HDF5 filter information (must match the output file)
- `offset[ndims]`: Offset of this chunk's origin in the collated dataset (must be on a chunk boundary of the output file)
- `data_size`: Size of the compressed chunk in bytes
- `buffer`: Compressed chunk data
- `async_writer_rank`: MPI rank of writer process

- `request`: [out]: MPI request for the communication

void **`close_variable_async`**(*varid_t* *varid*, int *async_writer_rank*)

Close a variable in the async writer.

varid is no longer a valid handle after this call

Parameters

- `varid`: Variable handle obtained with open_variable_async
- `async_writer_rank`: MPI rank of writer process

void **`close_async`**(int *async_writer_rank*)

Close the async writer.

Parameters

- `async_writer_rank`: MPI rank of writer process

4.1.2 Writer side

size_t **`run_async_writer`**(**`const`** char **filename*)

Async runner to accept writes.

Called by the Writer to accept async messages sent by the Readers. Once all Readers have sent close_async() messages this will return

Return total size written

Parameters

- `filename`: Output filename

4.2 error.h

Functions for reporting errors from the various libraries used

`NCERR`(x)

NetCDF error handler.

If a NetCDF call has errored reports the error and exits

Parameters

- `x`: The return code of a NetCDF library call

`H5ERR`(x)

HDF5 error handler.

If a HDF5 call has errored reports the error and exits

Parameters

- `x`: The return code of a HDF5 library call

`CERR`(x, message)

C error handler.

If a C library call has errored reports the error and exits

Parameters

- `x`: The return code of a C library call
- `message`: Error message

void **`set_log_level`**(int *level*)

Set the output level for log messages.

Parameters

- `level`: Messages with a level less than or equal to this will be output

void **`log_message`**(int *level*, **`const`** char **message*, ...)

Send a message to the log.

Parameters

- `level`: Message log level
- `message`: Message (printf-like format string)
- `...`: Message arguments

Available log levels are:

`LOG_DEBUG`

`LOG_INFO`

`LOG_WARNING`

`LOG_ERROR`

4.3 read_chunked.h

Functions the Readers use to read chunks from the input files and send them to the Writer

bool **`is_collated`**(int *ncid*, int *varid*)

Check if any of the dimensions of a variable are collated.

Return true if any dimension of varid is collated, false otherwise

Parameters

- `ncid`: NetCDF4 file handle
- `varid`: NetCDF4 variable handle

bool **`get_collation_info`**(int *ncid*, int *varid*, size_t *out_offset*[], size_t *local_size*[], size_t *total_size*[], int *ndims*)

Get collation info from a variable.

Return true if any of the dimensions are collated

Parameters

- `ncid`: NetCDF4 file handle
- `varid`: NetCDF4 variable handle
- `out_offset[ndims]`: (out): The offset in the collated array of this file's data
- `local_size[ndims]`: (out): This file's data size
- `total_size[ndims]`: (out): The total collated size of this variable
- `ndims`: Variable dimensions

bool **`get_collated_dim_decomp`**(int *ncid*, **`const`** char **varname*, int *decomposition*[4])

size_t **`get_collated_dim_len`**(int *ncid*, **`const`** char **varname*)
Get the global length of a collated variable.

Return : Collated variable length

Parameters

- `ncid`: NetCDF4 file handle
- `varname`: Variable name

void **`copy_chunked`**(**`const`** char **filename*, int *async_writer_rank*)
Copy all collated variables to the Writer.

The main function for Reader processes. Iterates over all collated variables in the file, choosing to send each variable in compressed (HDF5) or uncompressed (NetCDF4) mode to the Writer.

Parameters

- `filename`: Input filename
- `async_writer_rank`: MPI rank of the writer process

CHAPTER 5

Indices and tables

- genindex

Index