HDF5 and Diablo

Enrico Deusebio

February 10, 2015

This guide gives an overview of the HDF5 subroutines which have been implemented in DIABLO and provides a framework to handle IO operations.

What is HDF5

The Hierarchical Data Format 5 (HDF5) is a very flexible standard data format designed to handle large amount of data in a very structured fashion. The main tools for scientific computing, such as Python or Matlab, generally provide interfaces to read/write HDF5 files. Fortran and C codes need to be linked to HDF5 libraries (can be found here `http://www.hdfgroup.org/HDF5`). In the following we only provide an very general overview on HDF5 and how DIABLO HDF5 files are organized. For further information, we refer the interested reader to the userguide (`http://www.hdfgroup.org/HDF5/doc/index.html`).

Structure of HDF5

HDF5 allows to create files with a very flexible structure similar to a directory tree. The structure is composed by two basic primary elements

- *Groups* corresponding to directories, allow one to organise the data
- *Datasets* correponsing to files and constitutes the actual data contained in the file

Both groups and datasets can also be further described by attributes, which can be short information or data associated with them.

0.1 Advantages of HDF5

HDF5 has several advantages with respect to other types of IO operations and it is becoming increasingly used in several fields, especially in the context of High-Performance Computing.

- *Flexibilty.* HDF5 allows a large degree of flexibility, letting the user decide how to organize his/her own file in a tree-like structure. Compared to binary or formatted output data, there are no headers and/or no sequential reading/writing, thus making it very convenient for backwards compatibility when developing codes.

- *Large amount of data.* HDF5 also allows the handling of large amount of data which need not to be all read in the memory. It allows to read only part of the file and subsets of the datasets, making it possible to post-process big files in local machines if needed.
- *Parallel IO.* HDF5 features very efficient implementations for handling IO operations in parallel codes. The interface, available both in Fortran and C, automatically manage the processes and the IO operations on single file by multiple processes. This is very convinient as in many codes, parallel IO is done by either dumping one file for each process (overloading file systems for large number of processes) or communicating all the information to/from the master process that writes to/reads from the disk.

Perhaps the only drawback of HDF5 is that it requires to install the HDF5 library on local machines, although the process is usually smooth and effortless. In many HPC centres, due to its increasing use, HDF5 is already present and it is one of the recommended IO procedures for PRACE allocations.

HDF5 Procedures

In the following we give a quick overview of the HDF5 procedure implemented in Diablo. They are contained in two files: `hdf5s.f` to be used when the code is compiled serially; `hdf5.f` when the code is compiled for parallel use. The two files features exactly the same subroutines (same name and arguments) with slightly different implementation in order to account for the parallel output. This allows the core codes (*e.g.* `channel.f`) to be the same in parallel and serial run; and depending on the actual compilation rules (PARALLEL=TRUE or PARALLEL=FALSE) reference to the appropiate subroutine.

`WriteHDF(FNAME,SAVE_PRESSURE).` Writes out an entire flow field to disk.

Args:

- `FNAME : character*55`

  Name of the output file.

- `SAVE_PRESSURE : logical`

  Flag to save also the pressure field. This was done in order to assure perfect agreement between restarting simulations (recomputing the pressure when a file is read would otherwise introduce a small error).

`ReadHDF(FNAME).` Reads in a flow field from the disk. After reading the flow field overwrites the variable `U1`, `U2`, `U3` and `TH` and converts them to spectral space into the variable `CU1`, `CU2`, `CU3` and `CTH`. If the field `P` exists, it reads also the pressure.

Args:

- `FNAME : character*55`

  Name of the file to be read in.

`ReadGridHDF(FNAME,coord).` Reads in the grid (or the relevant portion of the grid in parallel cases) from a file.

Args:

- `FNAME : character*55`

  Name of the file containing the grid.

- `coord : integer`

  Integer referring to the coordinate (1-x, 2-y, 3-z) whose grid ought to be read. **Note:** At the present stage, only `coord=2` has been implemented.

`ReadGridHDF(FNAME,coord).` Reads in the grid (or the relevant portion of the grid in parallel cases) from a file.

Args:

- `FNAME : character*55`

  Name of the file containing the grid.

- `coord : integer`

  Integer referring to the coordinate (1-x, 2-y, 3-z) whose grid ought to be read. **Note:** At the present stage, only `coord=2` has been implemented.

`WriteHDF5_XYPlane(FNAME,gname,var2d).` Writes out a x-y plane.

Args:

- `FNAME : character*35`

  Name of the output file.

- `gname : character*10`

  Name of the group where the new plane will be placed. The planes are stored with an increasing id. The group has a property `SAMPLES` representing the number of planes saved. Each dataset has a property `Time` representing the simulation time at which the plane has been saved.

- `var2d : real*8, dimension(NX,NY)`

  2D plane to be saved. Note that in parallel cases `NY` is the number of vertical points per process.

`WriteHDF5_ZYPlane(FNAME,gname,var2d).` Writes out a y-z plane.

Args:

- `FNAME : character*35`

  Name of the output file.

- `gname : character*10`

  Name of the group where the new plane will be placed. The planes are stored with an increasing id. The group has a property `SAMPLES` representing the number of planes saved. Each dataset has a property `Time` representing the simulation time at which the plane has been saved.

- `var2d : real*8, dimension(NZ,NY)`

  2D plane to be saved. Note that in parallel cases `NY` is the number of vertical points per process.

`WriteHDF5_XZPlane(FNAME,gname,var2d).` Writes out a x-z plane.

Args:

- `FNAME : character*35`

  Name of the output file.

- `gname : character*10`

  Name of the group where the new plane will be placed. The planes are stored with an increasing id. The group has a property `SAMPLES` representing the number of planes saved. Each dataset has a property `Time` representing the simulation time at which the plane has been saved.

- `var2d : real*8, dimension(NX,NZ)`

  2D plane to be saved.

The next two subroutines are generally commented out and may be useful when bug-hunting.

`WriteHDF5_var_real(FNAME).` Writes out to the disk a *real* variable. The variable is written out to the `FNAME` file under the group `U`. The variable to be written out should be stored in a variable `real*8, dimension(0:NX+1,0:NZP+1,0:NY+1) tvar` to be declared in the header.

Args:

- `FNAME : character*35`

  Name of the output file.

`WriteHDF5_var_complex(FNAME).` Writes out to the disk a *complex* variable. The variable is written out to the `FNAME` file under the group `U`. The variable to be written out should be stored in a variable `complex*16, dimension(0:NXP,0:NZ+1,0:NY+1) tvar` to be declared in the header.

Args:

- `FNAME : character*35`

  Name of the output file.

Diablo HDF5 Files

In the following we provide a brief description of the HDF5 files used in DIABLO.

`grid.h5.`

File containing the grid. Now only the y grid is supported, altough this can be easily expanded to include grids in x and z as well.

Groups

- `/grids` contains the locations of the grids.

  Datasets

  - `y`. Location of the points in the full grid. The fractional grid is calculated within diablo.

`end.h5` and `end.h5`

Full flow fields in physical space.

Groups

- `/` Root with general information about the flow and the simulation

  Attributes

  - `Resolution`. Number of points in each direction.
  - `Data`. Data of creation of the file.
  - `Info`. Additional extra-information about the file

- `/Timestep`

  Group containing the flow field for a particular time step. Possibly, this could also be extended such that each file contains more flow fields in different group `Timestep1`,`Timestep2`, `Timestep3`, ...

  Attributes

  - `Time`. Simulation time

  Datasets

  - `U`. 3D flow field (streamwise velocity)
  - `V`. 3D flow field (vertical velocity)
  - `W`. 3D flow field (spanwise velocity)
  - `TH1`. 3D flow field (first scalar)
  - `TH?`. 3D flow field (# scalar)
  - `P`. 3D flow field (pressure)

`movie.h5`.

Files written with the `writeHDF5_xyplane` and `writeHDF5_zyplane` subroutines. The one below is just an example of a possible file

Groups

- `u_xy`

  Attributes

  - `SAMPLES` Number of planes in the group

  Datasets

  - `0000`. U x-y plane. An attribute `Time` specifies the time instant.
  - `0001`. U x-y plane. An attribute `Time` specifies the time instant.
  - $\cdots$ `SAMPLES`

`pixie.h5`.

PIXIE format files are HDF5 files which have a predefined structure and that can be imported straight away into visualisation tools such as Visit or Paraview. The contains information about the data to be visualised and its grid. This works for both structured (rectangular and curvilinear) and unstructured grids.

Groups

- `/node_coords`

  Group containing the location in x, y and z of the gridpoints.

  Datasets

  - `X`. Dataset of dimension `[Nxg,Nyg,Nzg]` containing x locations.
  - `Y`. Dataset of dimension `[Nxg,Nyg,Nzg]` containing y locations.
  - `Z`. Dataset of dimension `[Nxg,Nyg,Nzg]` containing z locations.

- `/Timestep`

  This group has all the data pertaining to a particular grid pointed by the attribute of the group

  Attributes

  - `coords`. Three element arrays of strings referring to the dataset containing the grid, *e.g.* `["/node_coords/X ", "/node_coords/Y ", "/node_coords/Z "]`
  - `Time`. Data simulation time stamp

  Datasets

  Here one can put as many datasets as he wants (with the same grid). There is a one-to-one map between datasets and grid coordinates.

  - `NameVar1`. Dataset of dimension `[Nxg,Nyg,Nzg]`.
  - $\cdots$ `NameVarN`

From version 1.8.0, HDF5 also supports external links, meaning that groups or datasets can also link to groups or data in other external files. Since it is often the case that several PIXIE files refer to the same grid, it could be efficient to store the grid only once and use external links to refer to it.

HDF5 and computing tools

Most of the tools used in scientific computing provide some interfaces to read/write HDF5 files. Here we provide a quick guide on some of them.

Matlab

Matlab supports a nice interface which allows to easily access HDF5. In order to have a glance of the HDF5 file just use

```
h5disp('myfile.h5')
```

For reading a particular attribute, e.g. time in 'end.h5', use

```
h5readatt('end.h5','/Timestep','Time')
```

whereas reading a dataset can be achieved by

```
h5readatt('end.h5','/Timestep/U')
```

Python

In order to use python to read HDF5 files, an additional package, `h5py`, must be installed (`http://www.h5py.org`). Moreover, I have developed an additional interface (module `h5` in the ext folder) which provides very similar commands as the one found in matlab.

Assuming that a `h5py` package is installed and working, import the `h5` module by

```
import h5
```

and then use the subroutines

```
h5.h5disp
```

```
h5.h5readatt
```

```
h5.h5read
```

similarly to what found in Matlab.

From the command line ...

When installing the HDF5 packages, some useful binaries are also installed. In the following, we outline some of them which can provides some insights into HDF5 files without the need of Python or Matlab.

In order to list the content of an HDF5 files use

```
h5ls [-r] myfile.h5
```

The flag `-r` recursively enters into the groups, listing the content.

Some further information can also be provided using

```
h5dump -H myfile.h5
```

which gives an overview on all the elements, along with their attribute, data type and size. `h5dump`, if the `-H` flag is removed, can also read and print out the actual values of attributes and datasets (although this is not recommended for large files).

Finally, if some further global informations on the file are needed (e.g. storage, number of groups, number of datasets, etc), refer to

```
h5stat myfile.h5
```

Note: HDF5 is a C library in which the order of the fastest element changing in a matrix is reversed with respect to Fortran. Thus, don't be surprised if when printing the size of matrices with bash, dimensions are left-to-right flipped.