IOStreamedHDF5

Thomas Radke

Date: 2004/03/13 18:31:47

Abstract

Thorn **IOStreamedHDF5** provides an I/O method to stream variables in HDF5 file format via live sockets to any connected clients. It also implements checkpointing/recovery functionality using HDF5.

1 Purpose

Thorn **IOStreamedHDF5** uses the standard I/O library HDF5 (Hierarchical Data Format version 5, see `http://hdf.ncsa.uiuc.edu/whatishdf5.html` for details) to output any type of CCTK grid variables (grid scalars, grid functions, and grid arrays of arbitrary dimension) in the HDF5 file format.

Output is done by invoking the `IOStreamedHDF5` I/O method which thorn **IOStreamedHDF5** registers with the flesh's I/O interface at startup.

Data is streamed as serialized HDF5 files over sockets to any connected client. Such datafiles can be used by appropriate programs for further postprocessing (eg. remote visualization).

Data is always written unchunked by processor 0, ie. the chunks of a distributed grid function or array will be collected from all other processors and streamed out as a single dataset. Parallel streaming from multiple processors is not supported yet.

2 IOStreamedHDF5 Parameters

Parameters to control the `IOStreamedHDF5` I/O method are:

- `IOStreamedHDF5::out_every` (steerable)
  How often to do periodic `IOStreamedHDF5` output. If this parameter is set in the parameter file, it will override the setting of the shared `IO::out_every` parameter. The output frequency can also be set for individual variables using the `out_every` option in an option string appended to the `IOStreamedHDF5::out_vars` parameter.
- `IOStreamedHDF5::out_vars` (steerable)
  The list of variables to output using the **IOStreamedHDF5** I/O method. The variables must be given by their fully qualified variable or group name. The special keyword *all* requests `IOStreamedHDF5` output for all variables. Multiple names must be separated by whitespaces.
  An option string can be appended in curly braces to a group/variable name. Supported options are `out_every` (to set the output frequency for individual variables) and hyperslab options (see section 3 for details).

3 Output of Hyperslab Data

By default, thorn **IOStreamedHDF5** outputs multidimensional Cactus variables with their full contents resulting in maximum data output. This can be changed for individual variables by specifying a hyperslab as a subset of the data within the N-dimensional volume. Such a subset (called a *hyperslab*) is generally defined as an orthogonal region into the multidimensional dataset, with an origin (lower left corner of the hyperslab), direction vectors (defining the number of hyperslab dimensions and spanning the hyperslab within the N-dimensional grid), an extent (the length of the hyperslab in each of its dimensions), and an optional downsampling factor.

Hyperslab parameters can be set for individual variables using an option string appended to the variables' full names in the `IOStreamedHDF5::out_vars` parameter.

Here is an example which outputs two 3D grid functions `Grid::r` and `Wavetoy::phi`. While the first is output with their full contents at every 5th iteration (overriding the `IOStreamedHDF5::out_every` parameter for this variable), a two-dimensional hyperslab is defined for the second grid function. This hyperslab defines a subvolume to output, starting with a 5 grid points offset into the grid, spanning in the yz-plane, with an extent of 10 and 20 grid points in y- and z-direction respectively. For this hyperslab, only every other grid point will be output.

```
IOStreamedHDF5::out_every = 1
IOStreamedHDF5::out_vars  = "Grid::x{ out_every = 5 }
                             Wavetoy::phi{ origin     = {4 4 4}
                                           direction  = {0 0 0
                                                          0 1 0
                                                          0 0 1}
                                           extent     = {10 20}
                                           downsample = {2 2}   }"
```

The hyperslab parameters which can be set in an option string are:

- `origin`
  This specifies the origin of the hyperslab. It must be given as an array of integer values with N elements. Each value specifies the offset in grid points in this dimension into the N-dimensional volume of the grid variable.
  If the origin for a hyperslab is not given, if will default to 0.

- `direction`
  The direction vectors specify both the directions in which the hyperslab should be spanned (each vector defines one direction of the hyperslab) and its dimensionality (= the total number of dimension vectors). The direction vectors must be given as a concatenated array of integer values. The direction vectors must not be a linear combination of each other or null vectors.
  If the direction vectors for a hyperslab are not given, the hyperslab dimensions will default to N, and its directions are parallel to the underlying grid.

- `extent`
  This specifies the extent of the hyperslab in each of its dimensions as a number of grid points. It must be given as an array of integer values with M elements (M being the number of hyperslab dimensions).
  If the extent for a hyperslab is not given, it will default to the grid variable's extent. Note that if the origin is set to a non-zero value, you should also set the hyperslab extent otherwise the default extent would possibly exceed the variable's grid extent.

- `downsample`
  To select only every so many grid points from the hyperslab you can set the downsample option. It must be given as an array of integer values with M elements (M being the number of hyperslab dimensions).
  If the downsample option is not given, it will default to the settings of the general downsampling parameters `IO::downsample_[xyz]` as defined by thorn **IOUtil**.

4 Checkpointing & Recovery

Thorn **IOStreamedHDF5** can also be used to create HDF5 checkpoints and stream them to another Cactus simulation which recovers from such a checkpoint at the same time.

Checkpoint routines are scheduled at several timebins so that you can save the current state of your simulation after the initial data phase, during evolution, or at termination. Checkpointing for thorn **IOStreamedHDF5** is enabled by setting the parameter `IOStreamedHDF5::checkpoint = "yes"`.

A recovery routine is registered with thorn **IOUtil** in order to restart a new simulation from a given HDF5 checkpoint.

Checkpointing and recovery are controlled by corresponding checkpoint/recovery parameters of thorn **IOUtil** (for a description of these parameters please refer to this thorn's documentation).

5 Building A Cactus Configuration with IOStreamedHDF5

The Cactus distribution does not contain the HDF5 header files and library which is used by thorn **IOStreamedHDF5**. So you need to configure it as an external software package via:

```
make <configuration>-config HDF5=yes
                            [HDF5_DIR=<path to HDF5 package>]
```

The configuration script will look in some default places for an installed HDF5 package. If nothing is found this way you can explicitly specify it with the `HDF5_DIR` configure variable.

Note that thorn **IOStreamedHDF5** uses the `Stream Virtual File Driver` of the HDF5 library as its low-level driver. This driver is not built into an HDF5 configuration by default. The configure script of **IOStreamedHDF5** will warn you if your HDF5 configuration doesn't contain this driver and stop the configuration process. Building an HDF5 library with `Stream` driver is very easy: just configure it with the `--enable-stream-vfd` option and build/install as usual.

Thorn **IOStreamedHDF5** inherits from **IOUtil** and **IOHDF5Util** so you need to include these thorns in your thorn list to build a configuration with **IOStreamedHDF5**.