CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 9: ADIOS, h5py, & VTK

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

https://sbyna.github.io

02/07/2023

Today's class

- Any questions?

- Class presentation topic

- Today's class –
  - ADIOS
  - h5py
  - VTK

ADIOS2

- ADaptable I/O System 2
- Development led by Oak Ridge National Laboratory

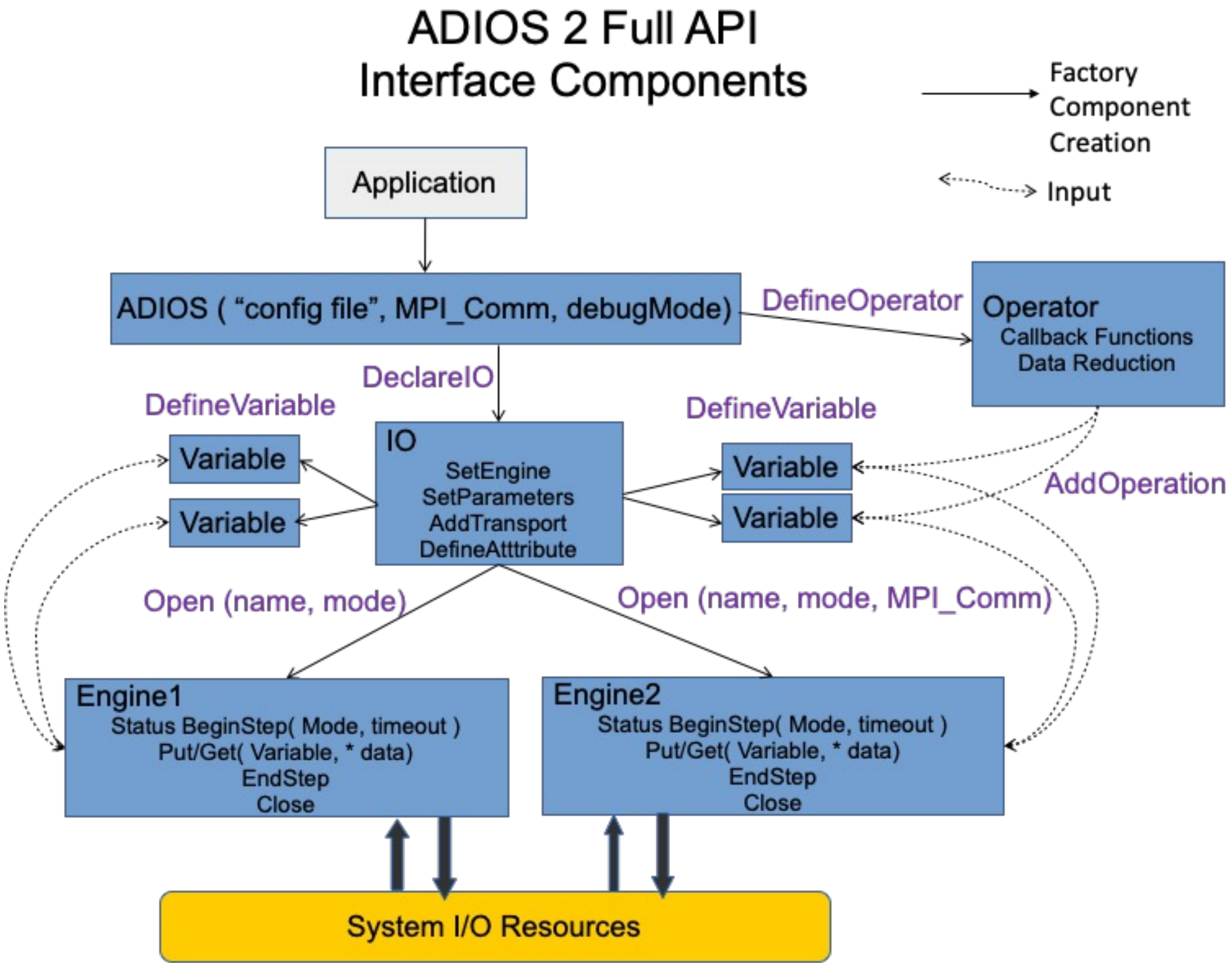

Image source: https://adios2.readthedocs.io/

ADIOS 2 internal software architecture layers

<table>
<tr><th>OSI Layer</th><th colspan="4">ADIOS2 classes</th></tr>
<tr><td>7 Application</td><td colspan="3">ADIOS, IO, Variable, Attribute, Operator</td><td rowspan="4">Engine</td></tr>
<tr><td>6 Presentation</td><td>Format: native binary-pack: BP3, BP4; json based: DataMan
Profiling: IOChrono, Timer
Compression lossy: ZFP, SZ, MGARD; lossless: BZIP2</td><td rowspan="3">Simplified Staging Transport:
FFS (layer 6),
SST (layer 5),
EVPath (layer 4)</td><td rowspan="3">Interoperability
HDF5</td></tr>
<tr><td>5 Session</td><td>Transport Managers: BP Manager, DataMan (WAN), InSituMPI
MPI_Aggregator : MPI_Chain</td></tr>
<tr><td>4 Transport</td><td>File: POSIX I/O, stdio, fstream
Network: MPI, RDMA (future)
WAN: ZMQ</td></tr>
<tr><td>3 Network
2 Data Link
1 Physical</td><td colspan="4">HARDWARE LAYERS (outside ADIOS 2)</td></tr>
</table>

Figure source:
https://www.sciencedirect.com/science/article/pii/S2352711019302560

ADIOS2 components

- ADIOS component
  - ADIOS object scope is through MPI Communicator
  - Optional runtime configuration file (XML format) to allow changing settings
- I/O component: bridge between the application specific settings, transports
  - Variables – the link between self-describing representation and data
  - Attributes – add extra information to the overall variables
  - Engines -- define the actual system executing the heavy IO tasks at
    - Open
    - BeginStep
    - Put
    - Get
    - EndStep
    - Close

Source: https://adios2.readthedocs.io/

ADIOS component

- ADIOS component
  - adios2::ADIOS adios("config.xml",
    MPI_COMM_WORLD);

```xml
<?xml version="1.0"?>
<adios-config>
  <io name="IONAME_1">

    <engine type="ENGINE_TYPE">

      <!-- Equivalent to IO::SetParameters-->
      <parameter key="KEY_1" value="VALUE_1"/>
      <parameter key="KEY_2" value="VALUE_2"/>
      <!-- ... -->
      <parameter key="KEY_N" value="VALUE_N"/>

    </engine>

    <!-- Equivalent to IO::AddTransport -->
    <transport type="TRANSPORT_TYPE">
      <!-- Equivalent to IO::SetParameters-->
      <parameter key="KEY_1" value="VALUE_1"/>
      <parameter key="KEY_2" value="VALUE_2"/>
      <!-- ... -->
      <parameter key="KEY_N" value="VALUE_N"/>
    </transport>
  </io>

  <io name="IONAME_2">
    <!-- ... -->
  </io>
</adios-config>
```

Image source: https://adios2.readthedocs.io/

IO component APIs – IO and variable

- DeclareIO
  - adios2::IO ADIOS::DeclareIO(const std::string ioName);

  - adios2::IO bpWriter = adios.DeclareIO("BPWriter");
  - adios2::IO bpReader = adios.DeclareIO("BPReader");

- Variable
  - adios2::Variable<T> DefineVariable<T>(const std::string name,

    const adios2::Dims &shape = {}, // Shape of global object

    const adios2::Dims &start = {}, // Where to begin writing

    const adios2::Dims &count = {}, // Where to end writing

    const bool constantDims = false);

Source: https://adios2.readthedocs.io/

IO component APIs - Attributes

- ```
  adios2::Attribute<T> DefineAttribute (
                          const std::string &name,
                          const T &value);
  ```

- ```
  adios2::Attribute<T> DefineAttribute (
                          const std::string &name,
                          const T *array, const size_t elements);
  ```

Source: https://adios2.readthedocs.io/

IO component APIs – Inquire variables

- Inquire about the status of variables and attributes when they have been previously defined

-
```
adios2::Variable<T> InquireVariable<T> (
        const std::string &name) noexcept;
```

-
```
adios2::Attribute<T> InquireAttribute<T> (
        const std::string &name) noexcept;
```

Source: https://adios2.readthedocs.io/

Engine component

- Engines execute the heavy operations in ADIOS2
- Each IO may select a type of Engine through the SetEngine function
- Available engines
  - BP4, BP5, or HDF5 → file
    - DEFAULT write/read ADIOS2 native bp files
    - write/read interoperability with HDF5 files
  - DataMan → write/read TCP/IP streams → Wide-area-network
  - SST → write/read to a "staging" area: *e.g.* RDMA → Staging

- void adios2::IO::SetEngine(const std::string engineType);
  - io.SetEngine("BP5");
  - adios2::Engine bpFile = io.Open("name", adios2::Mode::Write);

Source: https://adios2.readthedocs.io/

ADIOS Write example

Set dimensions

Open file stream

For each time step

```
#include <adios2.h>
...

int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

// Nx, Ny from application, std::size_t
const adios2::Dims shape{Nx, Ny * static_cast<std::size_t>(size)};
const adios2::Dims start{0, Ny * static_cast<std::size_t>(rank)};
const adios2::Dims count{Nx, Ny};

adios2::fstream oStream("cfd.bp", adios2::fstream::out, MPI_COMM_WORLD);

// NSteps from application
for (std::size_t step = 0; step < NSteps; ++step)
{
    if(rank == 0 && step == 0) // global variable
    {
        oStream.write<int32_t>("size", size);
    }

    // physicalTime double, <double> is optional
    oStream.write<double>( "physicalTime", physicalTime );
    // T and P are std::vector<float>
    oStream.write( "temperature", T.data(), shape, start, count );
    // adios2::endl will advance the step after writing pressure
    oStream.write( "pressure", P.data(), shape, start, count, adios2::end_step );

}

// Calling close is mandatory!
oStream.close();
```

Source: https://adios2.readthedocs.io/

ADIOS read example

```
#include <adios2.h>
...

int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

// Selection Window from application, std::size_t
const adios2::Dims start{0, 0};
const adios2::Dims count{SelX, SelY};

if( rank == 0)
{
   // if only one rank is active use MPI_COMM_SELF
   adios2::fstream iStream("cfd.bp", adios2::fstream::in, MPI_COMM_SELF);

   adios2::fstep iStep;
   while (adios2::getstep(iStream, iStep))
   {
       if( iStep.currentstep() == 0 )
       {
           const std::size_t sizeOriginal = iStep.read<std::size_t>("size");
       }
       const double physicalTime = iStream.read<double>( "physicalTime");
       const std::vector<float> temperature = iStream.read<float>( "temperature", start, count
       const std::vector<float> pressure = iStream.read<float>( "pressure", start, count );
   }
   // Don't forget to call close!
   iStream.close();
}
```

Source: https://adios2.readthedocs.io/

ADIOS BP5 optimizations

- Streaming through file
  - OpenTimeoutSecs, BeginStepPollingFrequencySecs
- Aggregation
  - **AggregationType**: *TwoLevelShm*, *EveryoneWritesSerial* and *EveryoneWrites*
  - **NumAggregators**
  - **AggregatorRatio**
  - **NumSubFiles**
  - **StripeSize**
  - **MaxShmSize**
- Buffering
- Managing steps
- Asynchronous writing I/O

More details:
https://www.sciencedirect.com/science/article/pii/S2352711019302560

Image source: https://adios2.readthedocs.io/

HDF5 for python – h5py

- H5py core concepts:
  - Groups work like dictionaries
  - Datasets work like NumPy arrays
- import h5py
- File object
  - f = h5py.File('mytestfile.hdf5', ‘r’)
  - f.close()

r	Readonly, file must exist (default)
r+	Read/write, file must exist
w	Create file, truncate if exists
w- or x	Create file, fail if exists
a	Read/write if exists, create otherwise

Source: https://docs.h5py.org/en/stable/

H5py – HDF5 file drivers

- f = h5py.File('myfile.hdf5', driver=<driver name>, <driver_kwds>)
- Virtual File Driver
  - maps the logical HDF5 address space to different storage mechanisms
  - sec2 → Unbuffered, optimized I/O using standard POSIX functions.
  - stdio → Buffered I/O using functions from stdio.h.
  - family → Store the file on disk as a series of fixed-length chunks.
  - fileobj → Store the data in a Python file-like object
  - split → Splits the meta data and raw data into separate files
  - ros3 → read-only access to HDF5 files in AWS S3 or S3 compatible object stores

h5py APIs – Groups, datasets

```
>>> grp = f.create_group("bar")
    >>> grp.name
    >> > '/bar'
>>> subgrp = grp.create_group("baz")
    >>> subgrp.name
    >>> '/bar/baz'

>>>  dset = f.create_dataset("default", (100,))
>>> dset = f.create_dataset("ints", (100,), dtype='i8')
>>> arr = np.arange(100)
>>> dset = f.create_dataset("init", data=arr)
```

Reading and writing datasets

- Datasets re-use the NumPy slicing syntax to read and write to the file
- Slice specifications are translated directly to HDF5 “hyperslab” selections
  - Indices: anything that can be converted to a Python long
  - Slices (i.e. [:] or [0:10])

- Write examples
  - dset = f.create_dataset("MyDataset", (10,10,10), 'f’)
  - dset[0,0,0] dset[0,2:10,1:9:3]
- Read examples
  - dset.fields("FieldA")[:10] # Read a single field
  - dset[:10]["FieldA"] # Read all fields, select in NumPy

Other h5py dataset APIs

- Chunked storage
  - dset = f.create_dataset("chunked", (1000, 1000), chunks=(100, 100))
- Auto-chunking
  - dset = f.create_dataset("autochunk", (1000, 1000), chunks=True)
- Resizable
  - dset = f.create_dataset("resizable", (10,10), maxshape=(500, 20))
  - dset = f.create_dataset("unlimited", (10, 10), maxshape=(None, 10))
- Filters
  - dset = f.create_dataset("zipped", (100, 100), compression="gzip")
  - dset = f.create_dataset("zipped_max", (100, 100), compression="gzip", compression_opts=9)

h5py – Parallel HDF5

- Example

```
from mpi4py import MPI
import h5py
rank = MPI.COMM_WORLD.rank # The process ID (integer 0-3 for 4-process run)
f = h5py.File ('parallel_test.hdf5', 'w', driver='mpio', comm=MPI.COMM_WORLD)
dset = f.create_dataset('test', (4,), dtype='i’)
dset[rank] = rank
f.close()
```

VTK

- The Visualization ToolKit (VTK)
  - open source,
  - Developed and maintained by Kitware
  - software system for 3D computer graphics, image processing, and visualization

  - Provides efficient implementations of a variety of visualization algorithms

  - C++ class library,
  - Several interpreted interface layers including Python, Tcl/Tk and Java

VTK file format

Part 1: Header	**Part 4:** Geometry/Topology. ***Type*** is one of STRUCTURED_POINTS STRUCTURED_GRID UNSTRUCTURED_GRID POLYDATA STRUCTURED_POINTS RECTILINEAR_GRID FIELD
Part 2: Title (256 characters maximum, terminated with newline \n character)	**Part 5:** Dataset attributes. The number of data items n of each type must match the number of points or cells in the dataset. (If *type* is FIELD, point and cell data should be omitted.
Part 3: Data type, either ASCII or BINARY	

# vtk DataFile Version 2.0	(1)
Really cool data	(2)
ASCII | BINARY	(3)
DATASET ***type*** ...	(4)
POINT_DATA ***type*** ... CELL_DATA ***type*** ...	(5)

Source: https://kitware.github.io/vtk-examples/site/VTKFileFormats/

VTK Dataset formats

- Supports five types of dataset formats
- Structured Points: 1D, 2D, and 3D structured point datasets

DATASET STRUCTURED_POINTS
DIMENSIONS n_x n_y n_z
ORIGIN x y z
SPACING s_x s_y y_z

- Structured Grid:

DATASET STRUCTURED_GRID
DIMENSIONS n_x n_y n_z
POINTS ***n dataType***
p_{0x} p_{0y} p_{0z}
p_{1x} p_{1y} p_{1z}
...
$p_{(n-1)x}$ $p_{(n-1)y}$ $p_{(n-1)z}$

Source: https://kitware.github.io/vtk-examples/site/VTKFileFormats/

VTK Dataset formats

- Rectilinear Grid: regular topology, and semi-regular geometry aligned along the x-y-z coordinate axes

DATASET RECTILINEAR_GRID
DIMENSIONS n_x n_y n_z
X_COORDINATES n_x ***dataType***
x_0 x_1 ... $x_{(nx-1)}$
Y_COORDINATES n_y ***dataType***
y_0 y_1 ... $y_{(ny-1)}$
Z_COORDINATES n_z ***dataType***
z_0 z_1 ... $z_{(nz-1)}$

- Polygonal data: consists of arbitrary combinations of surface graphics primitives vertices (and polyvertices), lines (and polylines), polygons (of various types), and triangle strips

DATASET POLYDATA
POINTS ***n dataType***
p_{0x} p_{0y} p_{0z}
p_{1x} p_{1y} p_{1z}
...
$p_{(n-1)x}$ $p_{(n-1)y}$ $p_{(n-1)z}$

VERTICES ***n size***
$numPoints_0$, i_0, j_0, k_0, ...
$numPoints_1$, i_1, j_1, k_1, ...
...
$numPoints_{n-1}$, i_{n-1}, j_{n-1}, k_{n-1}, ...

LINES ***n size***
$numPoints_0$, i_0, j_0, k_0, ...
$numPoints_1$, i_1, j_1, k_1, ...
...
$numPoints_{n-1}$, i_{n-1}, j_{n-1}, k_{n-1}, ...

POLYGONS ***n size***
$numPoints_0$, i_0, j_0, k_0, ...
$numPoints_1$, i_1, j_1, k_1, ...
...
$numPoints_{n-1}$, i_{n-1}, j_{n-1}, k_{n-1}, ...

TRIANGLE_STRIPS ***n size***
$numPoints_0$, i_0, j_0, k_0, ...
$numPoints_1$, i_1, j_1, k_1, ...
...
$numPoints_{n-1}$, i_{n-1}, j_{n-1}, k_{n-1}, ...

Source: https://kitware.github.io/vtk-examples/site/VTKFileFormats/

VTK Dataset formats

- Unstructured grid: arbitrary combinations of any possible cell type. Unstructured grids are defined by points, cells, and cell types.

DATASET UNSTRUCTURED_GRID
POINTS ***n dataType***

p_{0x} p_{0y} p_{0z}
p_{1x} p_{1y} p_{1z}
...
$p_{(n-1)x}$ $p_{(n-1)y}$ $p_{(n-1)z}$

CELLS ***n size***
$numPoints_0, i_0, j_0, k_0, \ldots$
$numPoints_1, i_1, j_1, k_1, \ldots$
$numPoints_2, i_2, j_2, k_2, \ldots$
...
$numPoints_{n-1}, i_{n-1}, j_{n-1}, k_{n-1}, \ldots$

CELL_TYPES ***n***
$type_0$
$type_1$
$type_2$
...
$type_{n-1}$

VTK HDF files – Image data

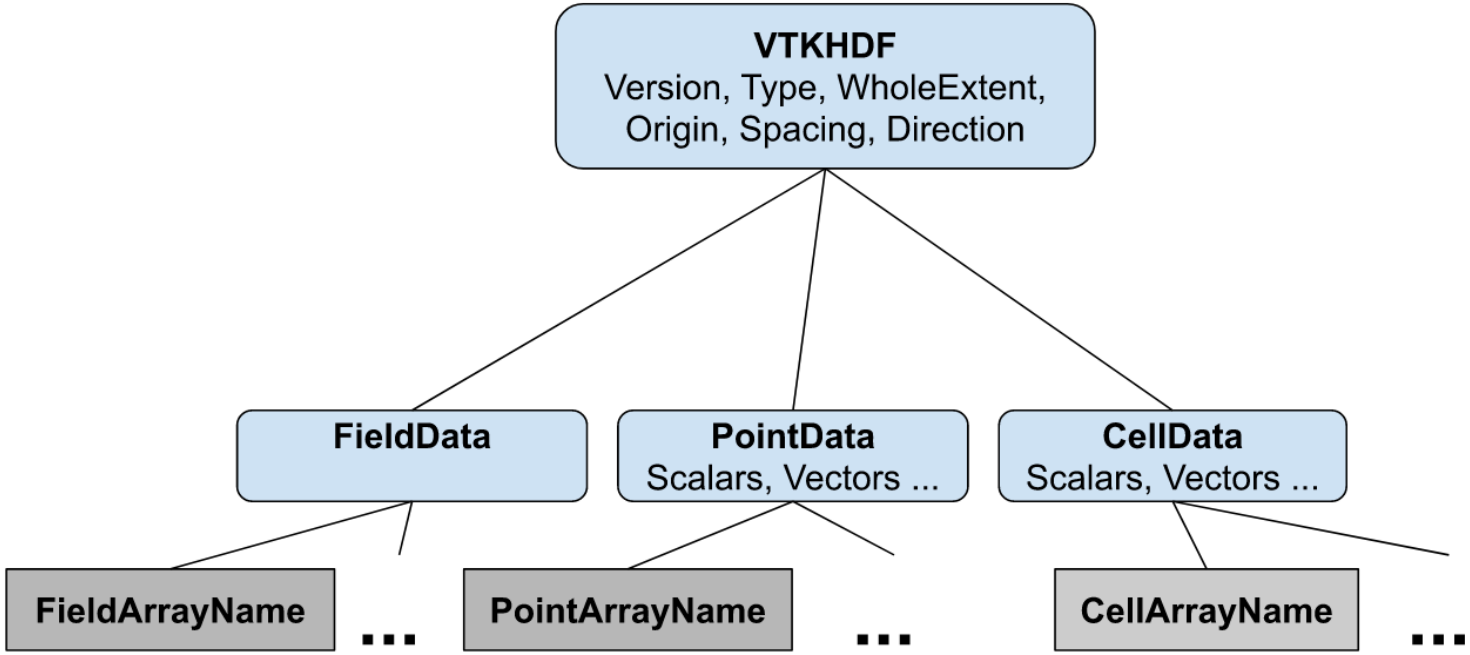

Figure 6. - Image data VTKHDF File Format

Source: https://kitware.github.io/vtk-examples/site/VTKFileFormats/#hdf-file-formats

VTK HDF files – Unstructured grid

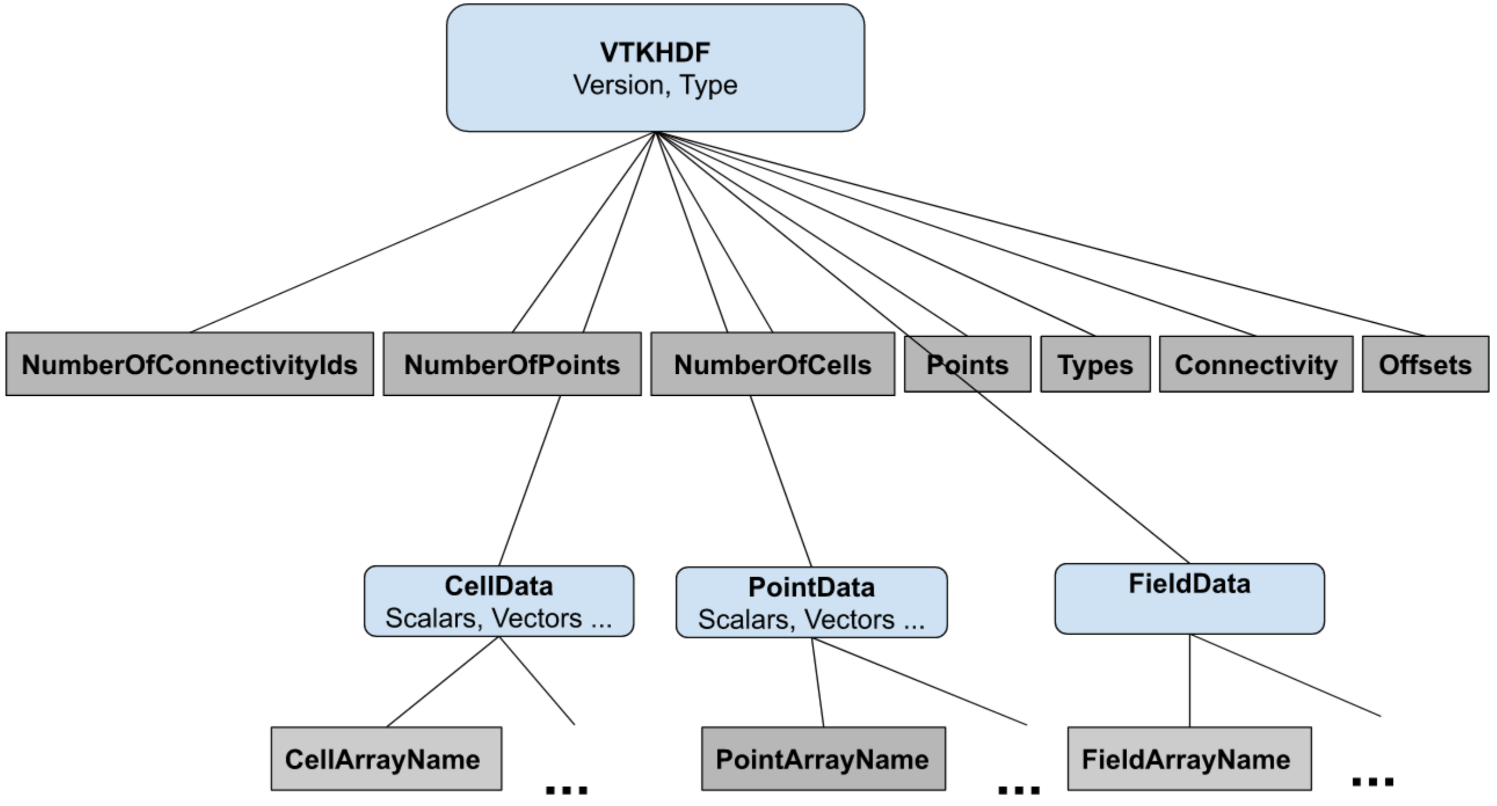

Figure 7. - Unstructured grid VTKHDF File Format

Source: https://kitware.github.io/vtk-examples/site/VTKFileFormats/#hdf-file-formats

Summary of today's class

- ADIOS, VTK, h5py

- Next Class – Parallel I/O in VTK and ParaView, I/O performance topics

- Homework: h5bench, PnetCDF examples