PROGRESS ON H5PART: A PORTABLE HIGH PERFORMANCE PARALLEL DATA INTERFACE FOR ELECTROMAGNETICS SIMULATIONS*

A. Adelmann, A. Gsell, B. Oswald, T. Schietinger, PSI, Villigen, Switzerland
W. Bethel, J.M. Shalf, C. Siegerist, K. Stockinger, LBNL/NERSC, Berkeley, California, USA

Abstract

Significant problems facing all experimental and computational sciences arise from growing data size and complexity. Common to all these problems is the need to perform efficient data I/O on diverse computer architectures. In our scientific application, the largest parallel particle simulations generate vast quantities of six-dimensional data. Such a simulation run produces data for an aggregate data size up to several TB per run. Motived by the need to address data I/O and access challenges, we have implemented H5Part, an open source data I/O API that simplifies the use of the Hierarchical Data Format v5 library (HDF5). HDF5 is an industry standard for high performance, cross-platform data storage and retrieval that runs on all contemporary architectures from large parallel supercomputers to laptops. H5Part, which is oriented to the needs of the particle physics and cosmology communities, provides support for parallel storage and retrieval of particles, structured and in the future unstructured meshes. In this paper, we describe recent work focusing on I/O support for particles and structured meshes and provide data showing performance on modern supercomputer architectures like the IBM POWER 5.

MOTIVATION

Modern large-scale parallel simulations produce data volumes that are on the orders of TBs. One of the main challenges is how to access this data efficiently and how to share it among scientists of specific communities. In order to address these problems we have developed H5Part [1], a high-performance data API that is particularly targeted for the accelerator modeling community.

H5Part uses HDF5 [2] as the underlying storage format which has the following benefits: 1) Machine independence: No byte-swapping is necessary for accessing binary data created on different machines. 2) Language independence: Data can, for instance, be written using the Fortran API and read using the C/C++ API. 3) Self-describing: Data is accessed by names rather than position. For instance, read the values of the dataset p_x. 4) High-performance: Data is stored in native binary format and is only automatically translated if the machine that reads the data requires a different format. 5) Parallel I/O: Data is written in parallel into a single file using MPI-I/O.

*This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098

DATA MODELS IN H5PART

H5Part supports two types of data models. One model (H5Part) stores time varying unstructured datasets with multiple variables per datum; The other model (H5Block) stores multidimensional, timevarying block-structured fields. Both data models support parallel reading and writing of the respective data. We will now explain these two data models in more detail.

H5Part: Particle Data

The data model for particle data allows storing multiple timesteps where each timestep can contain several datasets of the same length. By definition, each timestep must have the same number of datasets. Typical particle data consists of the 3-dimensional Cartesian positions of particles (x, y, z) as well as the corresponding 3-dimensional momenta (p_x, p_y, p_z). These 6 variables are stored as 6 HDF5 datasets. The type of the dataset can be either integer or real. H5Part also allows storing attribute information for the file and the timesteps.
A simplified pseudo code for storing particle data with n timesteps is shown below. Note that if a file is opened in parallel, the data is partitioned and written in parallel based on the number of particles.

```
if(not parallel);
  filehandle=OpenFile(filename,mode)
else
  filehandle=OpenFile(filename,mode,mpicomm)
SetNumberOfParticles(filehandle);
loop(step=1,NSteps);
  compute data
  SetStep(filehandle,step);
  WriteData(filehandle,fieldname1,data1);
                              ...
  WriteData(filehandle,fieldname<n>,data<n>);
CloseFile(filehandle);
```

The internal HDF5 file structure of the above example with 2 timesteps is as follows:

```
GROUP "/" {
   GROUP "Step#0" {
      DATASET "px" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
      }
      DATASET "py" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
      }
      DATASET "pz" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
      }
      DATASET "x" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
      }
      DATASET "y" {
         DATATYPE  H5T_IEEE_F64LE
```



```
      DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
    }
    DATASET "z" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 1000 ) / ( 1000 ) }
    }
  }
  GROUP "Step#1" {
    ... same information as in Step#0
  }
}
```

H5Block: Block-Structured Data

H5Block inherits all features of H5Part such as multiple timesteps, file and timestep attributes etc. but also supports scalar fields and vector fields. Examples of these two types of data are the *potential* and *electrical field* of a particle simulation. Since H5Block stores multidimensional datasets, the data partition strategy among the processors must be specified explicitly for each data dimension. Find below the pseudo code for partitioning a 3D scalar field with dimensionality 16, 16, 128 among two processors:

```
fh=OpenFile(filename,mode)
if processor 0
  Define3DFieldLayout(fh, 0, 15, 0, 15, 0, 63);
else
  Define3DFieldLayout(fh, 0, 15, 0, 15, 64, 127);
Write3DScalarField(fh,fieldname1,data1);
CloseFile(fh);
```

Below we show the internal HDF5 file structure of H5Block. Note that the 3D scalar field *potential* is stored as one 3-dimensional data set. On the other hand, the 3D vector field *electrical field* is stored as three 3-dimensional datasets.

```
GROUP "Block" {
    GROUP "Potential" {
       DATASET "0" {
          DATATYPE  H5T_IEEE_F64LE
          DATASPACE  SIMPLE { ( 128, 16, 16 ) / ( 128, 16, 16 ) }
       }
    }
    GROUP "Electrical Field" {
       DATASET "0" {
          DATATYPE  H5T_IEEE_F64LE
          DATASPACE  SIMPLE { ( 128, 16, 16 ) / ( 128, 16, 16 ) }
       }
       DATASET "1" {
          DATATYPE  H5T_IEEE_F64LE
          DATASPACE  SIMPLE { ( 128, 16, 16 ) / ( 128, 16, 16 ) }
       }
       DATASET "2" {
          DATATYPE  H5T_IEEE_F64LE
          DATASPACE  SIMPLE { ( 128, 16, 16 ) / ( 128, 16, 16 ) }
       }

 }
```

DATA MINING AND VISUALIZATION

Massive parallel applications in science generally scale at least linearly with the number of processors. The balance between generation of data and the capability to (post)process i.e. analyze them is of increasing importance. Data that cannot be looked at in an acceptable time frame are essentially lost.

Data Visualization with VisIt

VisIt [6] is an open source scientific visualization application that supports most of the common visualization techniques on structured and unstructured grids. One of its advantages is that it employs a distributed and parallel architecture in order to handle extremely large data sets interactively.

We have developed plugins for reading and visualizing both particle and field data. Figure 1 shows a visualization of the scalar field "potential" and the particle data "px" for timesteps 3 and 42 of our simulation data.

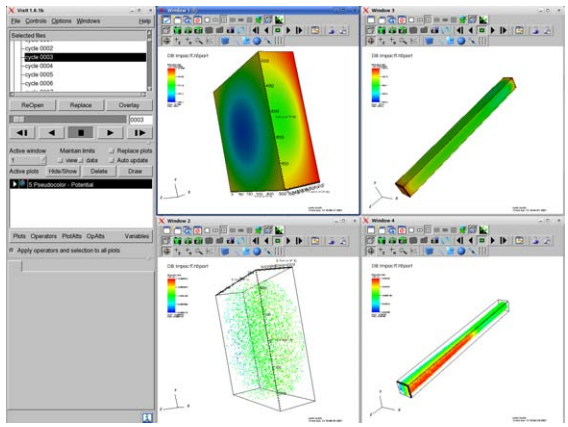

Figure 1: Visit is able to read and visualize H5Part data. Windows 1 (top left) and 3 (top right) display the scalar field "potential" for the timesteps 3 and 42. Windows 2 (bottom left) and 4 (bottom right) display the particle data "px" for the same timesteps.

H5PartROOT

H5PartROOT is a tool to visualize large data files produced with the H5Part interface to HDF5. It is based on the ROOT framework for data analysis [5] developed at CERN. The main Graphical User Interface (GUI) allows convenient navigation between time steps and between several datafiles for quick comparisons at the click of a mouse. The basic functionality of the tool ranges from plotting one-, two-, and three-dimensional particle distributions to line plots of step attributes such as emittance (projected, slice and screen), rms beam size and centroid position, which are either read in directly from the datafile or reconstructed from the particle distribution at the current time step. More sophisticated fitting procedures and moment determination are available via the corresponding ROOT classes.

The H5PartROOT software essentially consists of two classes, which both inherit from ROOT classes to ensure access to the full range of features provided by the ROOT framework, for both interactive sessions and compiled and linked executables. The first class creates the main GUI, the second one stores the data of one HDF5 datafile locally in ROOT-native data containers and supplies various methods to retrieve, plot and manipulate the data. The methods of the data class are connected to the GUI via the so-called signal/slot mechanism, a technique originally designed for the Qt toolkit and recently adopted by ROOT. While H5PartROOT provides a stand-alone executable, which launches a GUI ready for use, the H5PartROOT

classes may also be linked as a shared library to a standard ROOT session. This allows a user to take advantage of the full H5PartROOT functionality from within ROOT interactive sessions or macros. Figure 2 shows a screenshot of a typical H5PartROOT session.

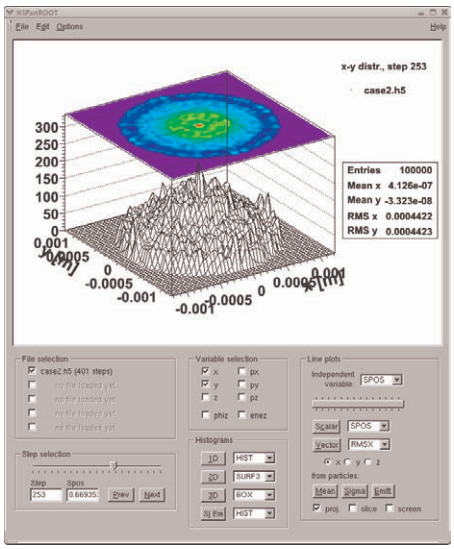

Figure 2: A snapshot of the H5PartROOT Graphical User Interface.

PERFORMANCE

In this section we will analyze the performance of H5Part and compare the results with MPI-I/O. All experiments are executed on an IBM p575 POWER 5 system using up to 8 nodes, where each node consists of 8 CPUs. In total, the benchmarks were run on 8 to 64 CPUs using the GPFS filesystem. One of the major tuning parameters of H5Part is whether to choose collective or non-collective I/O. In order to answer the question, we ran a large set of performance measurements based on the IOR-benchmark [3]. We have chosen the IOR benchmark since it allows one to study the performance of applications with different access patterns. These performance results suggest that non-collective I/O is significantly faster than collective I/O for the H5Part specific access pattern where the data sets are evenly partitioned among the parallel processors. Hence, all our performance benchmarks are based on non-collective I/O.

In our first set of benchmarks we used 10^8 particles with 6 attributes and 5 timesteps. In total, these experiments write some 24 GB of data. Figure 3 (left) shows the performance of H5Part, MPI-I/O and POSIX-file I/O using between 8 and 64 CPUs. Note that H5Part and MPI-I/O write a single large file whereas POSIX-I/O writes one file per processor. The results demonstrate that H5Part shows similar performance to MPI-I/O. This result confirms that use of H5Part does not introduce any significant performance overhead compared to direct use of MPI-I/O. We also see that POSIX-file I/O scales nearly linearly with the number of processors. As part of our future work we will investigate additional performance tuning possibilities of H5Part and MPI-I/O.

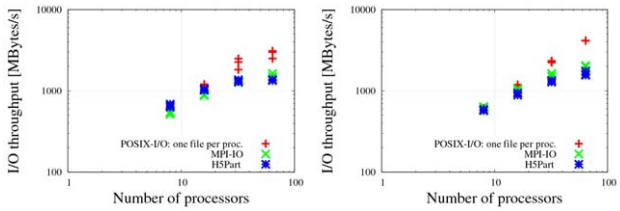

Figure 3: Performance of H5Part compared with POSIX and MPI-I/O, where the total size of the written data is constant (left plot), or increases with the number of processors (right plot).

In our next set of experiments we increased the data size with the number of processors. In other words, we increased the problem size as we increased the number of processors. Again, we used 10^8 particles with 6 attributes but we varied the number of timesteps between 4 and 32 for 8 to 64 CPUs. In total, the amount of data that is written is between 19 and 152 Gigabytes. The results of Fig. 3 (right) demonstrate the similar performance of H5Part and MPI-I/O.

The performance results show that H5Part is capable of writing extremely large files showing good performance. The advantage of writing one large file in parallel over writing many smaller files is better usability. Managing one file is clearly simpler than managing a large set of files.

CONCLUSIONS AND FUTURE WORK

We are currently working on integrating the FastBit bitmap indexing technology [4] for accelerating queries of data stored in H5Part. An example of such a query is *(p_x > 1.34) AND (potential* $< 2e7$*)*. A formal definition of an API for unstructured finite element data has been written, the implementation will start this summer.

REFERENCES

[1] A. Adelmann, R.D Ryne, C. J. Shalf, Siegerist, "H5Part: A Portable High Performance Parallel Data Interface for Particle Simulations", PAC 2005.

[2] HDF5 Home Page, `http://hdf.ncsa.uiuc.edu/HDF5`.

[3] Interleaved or Random (IOR) benchmarks. `http://www.llnl.gov/icc/lc/siop/downloads/download.html`

[4] L. Gosink, J. Shalf, K. Stockinger, K. Wu, W. Bethel, "HDF5-FastQuery: Accelerating Complex Queries on HDF Datasets using Fast Bitmap Indices", SSDBM 2006.

[5] ROOT – an Object-Oriented Data Analysis Framework, `http://root.cern.ch/`.

[6] VisIt – a free interactive parallel visualization and graphical analysis tool, `http://www.llnl.gov/visit/`.

