# H5Part: A Portable High Performance Parallel Data Interface for Particle Simulations[*]

A. Adelmann, PSI, Villigen, Switzerland
R.D. Ryne, LBNL/AFR, Berkeley, California, USA
J.M. Shalf, C.Siegerist, LBNL/NERSC, Berkeley, California, USA

## *Abstract*

The very largest parallel particle simulations, for problems involving six dimensional phase space, generate vast quantities of data. It is desirable to store such enormous datasets efficiently and also to share data effortlessly between data analysis tools such as PartView [**?**] and extensions to AVS/Express among other groups who are working on particle-based accelerator simulations. We define a very simple file schema built on top of HDF5 [**?**] (Hierarchical Data Format version 5) as well as an API that simplifies the reading/writing of the data to the HDF5 file format. HDF5 offers a self-describing machine-independent binary file format that supports scalable parallel I/O performance for MPI codes on computer systems ranging from laptops to supercomputers. The sample H5Part API is available for C, C++, and Fortran codes. The common file format will enable groups that use completely different simulation implementations to transparently share datasets and custom data analysis tools like PartView. We will show examples and benchmark data for various platforms.

## MOTIVATION

The motivation for this undertaking is to create a file format that is suitable for large-scale parallel simulation codes. A suitable data format must have the following properties: it must be a machine-independent binary representation that is self-describing, easily extensible, language independent, efficient (both for serial and parallel), and produces files that are seamlessly sharable between different programs. In the following sections we describe the motivation for these features and how they can be accomplished using the proposed implementation.

### *Machine Independence*

Processor architectures use different binary representations for data. While the IEEE 754 standard has decreased the number of differing floating point number represenations, byte-order still remains a source of incompatibility. While it is trivial to byte-swap a file from a programming standpoint, it creates a number of long-term file management difficulties for groups that are sharing or maintaining a repository of simulation data. Given files that store data structures with differently sized elements, one must know the storage format precisely in order to apply byte-swapping properly. HDF5 does not suffer from these problems because the file format is completely self-describing and the internal binary structures are all machine independent. The HDF5 library is able to convert data that is stored in any native machine represenation in the data file into a native binary representation in memory as part of the reading process with little performance penalty.

### *Language Independence*

The three most common languages used for implementing applications in the arena of computational sciences are Fortran, C and C++. The file format and associated API must hide differences in the binary file-storage conventions of these languages as well as offering native API bindings for each of these programming languages.

For instance, Fortran unformatted binary files contain integer fields at the beginning and the end of each record that describe the length of the record. The size of these integer tags is usually 32-bits, but some fortran compiler implementations will use larger tags in order to represent larger record sizes. C and C++ binary files have no such convention for record-oriented storage. The language-dependent differences in binary storage layout conventions can cause difficulties for scientists who wish to share data files between Fortran and C/C++ implementations of a code, or with visualization tools that are primarily written in C/C++. The API bindings and underlying file format provided by the H5Part API and the underlying HDF5 file format are able to hide these differences in order to provide symmetric access via all languages.

### *Self-Describing*

The data is accessed by names, for example, one might ask for 'the column of data called $p_x$' – affording a layer of file-layout independence. In other words, self-describing data is not accessed by a position in a file but by name of the datasets. Various attributes of the data that may be necessary to using it are available. For example, one can ask "what are the units of column $p_x$?'.

There are a number of examples of self-describing file formats. Examples include HDF earlier HDF implementations leading up to HDF version 5 and the Unidata NetCDF format. Another very popular approach is the Self Describ-

[*] 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 and the DOE SciDAC project on "Advanced Computing for 21st Century Accelerator Science and Technology.".

ing Data Sets (SDDS) [**?**] although it is only serial in nature. HDF5 is a complete rewrite of the HDF file format that supports parallel I/O and offers a much leaner, more flexible interface. Because it is self-describing, the entire contents of an HDF5 file can be browsed and even converted to ASCII text, including XML syntax text files, using the built-in ’h5ls’ and ’h5dump’ tools without specific knowledge of the internal file format.

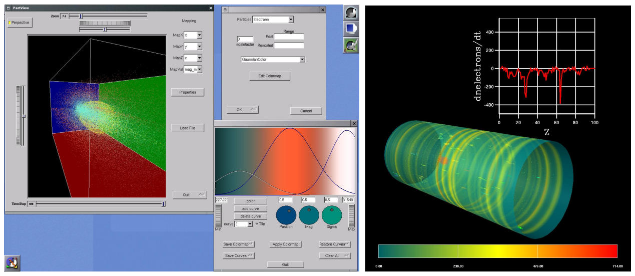

Figure 1: color: A common self-describing file format allows different codes to share a common set of visualization and data analysis tools. PartView and AVS/Express, pictured above, are able to read and display contents of an H5Part/HDF5 file written on any machine in any language, regardless of how many processors are used.

The primary advantage of accessing data and its attributes is that one can then construct more flexible data manipulation tools that are capable of surviving the natural evolution of file formats. Data formats can be extended to include additional information without breaking older file readers. Self-describing data contains all the information that analysis tools need to manipulate various types of data correctly. Two examples of such tools using the proposed file format are shown in Figure **??** As a result, data exchange between different simulations tools is much simpler, robust and better defined by using self-describing data sets.

### *High Performance*

The HDF5 file format allows data elements to be written to disk in the native binary representation. The file format stores a description of the native data representation of the machine that wrote the data so that it can be automatically translated to the native binary representation of the machine that reading the data (eg. if the byte order differs). In contrast to XDR, where the data always gets translated to/from an intermediate machine-neutral format, the HDF5 data conversion only occurs if the stored data represenation is different from the native binary representation of the machine that is reading the file, so there is no performance penalty if the machines have the compatible binary data formats. [ada: need work] In general, HDF5 offers performance that is very close to what can be achieved by writing an ad-hoc machine-dependent binary for F77 unformatted data file.

### *Parallel I/O*

HDF5 also supports parallel I/O capabilities for MPI programs. The naive approach to writing data from a parallel program is to write one file per processor. While this is simple to implement and very efficient on most cluster filesystems, it leads to file management headaches when it comes the time to analyze the data. One must either recombine these separate files into a single file or create ponderous user-interfaces that allow a data analysis application to read from a directory full of files instead of just one file.

Parallel I/O methods, allow you to write data into a single file from all of the tasks of a parallel program. The performance is typically lower than that of writing one-file-per-processor, but it makes data management much simpler after the program has finished. No additional recombining steps are required to make the file accessible by vis-tools or for restarting a simulation using a different number of processors.

Parallel HDF5 uses MPI-I/O for its low-level implementation. The mechanics of using MPI-I/O are all hidden from the user by our H5Part file API (the code looks nearly identical to reading/writing the data from a serial program). While the performance is not as good as writing one-file-per-processor, we demonstrate that writing files with Parallel HDF5 is consistently faster than writing the data in raw/native binary using the MPI-I/O library. This efficiency is made possible through sophisticated HDF5 tuning directives that control data alignment and caching within the HDF5 layer. Therefore, we argue that it would be difficult to match HDF5 performance even using a home-grown binary file format.

## H5PART FILE ORGANIZATION AND API

The proposed file storage format uses HDF5 for the low-level file storage and a simple API to provide a high-level interface to that file format. A programmer can either use the H5Part API to access the data files or write directly to the file format using some simple conventions for organizing and naming the objects stored in the file.

The HDF5 format, its benefits, and its file organization is decribed at [**?**]. The file format was also adopted by the DOE ASCI-VIEWS effort, so the library has been tuned and adapted to read and write data efficiently on large-scale parallel computing systems. We adopted HDF5 for our file storage needs because it offers all that is needed as stipulated in the motivation section.

We describe now the H5Part conventions for storing objects in the HDF5 file as well as some examples of the API.

### *H5Part File Organization*

In order to store Particle Data in the HDF5 file format, we have formalized the hierarchical arrangement of the datasets and naming conventions for the groups and associated datasets. The sample H5Part API formally encodes these conventions in order to provide a simple and uniform way to access these files from C, C++, and Fortran codes. The API makes it easier to write very portable data adaptors for visualization tools in order to expand the number of tools available to access the data. Even so, anyone can use the HDF5 $h5ls$ utility to examine the organization of the H5Part files and even write their own HDF5-based interface for reading and writing the file format. The standards offered by the sample API are completely independent of the standard for organizing data within the file.

The file format supports the storage of multiple timesteps of datasets that contain multiple fields. The fields correspond to different properties of the particles at that particular time step – for instance, the 3-dimensional cartesian position of the particles $(X, Y, Z)$ as well as the 3-dimensional phase of each particle $(PX, PY, PZ)$. These two degrees of freedom are organized such that the timesteps are groups (time groups) that are added sequentially to the root group ("/"). The fields are datasets that are nested within the time groups. The convention for naming the time group is $Particles < integer >$ where $< integer >$ is a monotonically increasing counter for the number of timesteps stored in the file.

The fields contained within a given time group are simply named for the property of the particle they represent. For instance, the phase of the particle stored in a simulation variable called 'px' is simply named $``px''$. The field names are user-defined and can be understood automatically by the visualization tools that read the file. The only other convention is that each time group must contain the same set of fields – the contents of the fields will change, but the set of names for these fields must remain the same for all timesteps.

The fields can be either integer or real data types. Initially, the file format supports double precision float and 64-bit integers in order to simplify the requirements for file readers, but HDF5 is capable of automatically down-converting to 32-bit data types upon request. The API will be extended accordingly to support these conversions.

Finally, the file, the individual timesteps, and the individual data arrays can contain *attributes* that provide additional information about the data. For instance, the datasets can be annotated with attributes containing *units* for a given data field, simulation parameters, or code revision information. The *attributes* are key-value pairs where the $key$ is a string that is associated with the file, group, or dataset, and the $value$ is either a string, a real value, or an integer associated with that key.

## GENERAL FORM IN PSEUDOCODE

In Figure **??** we show the very simple API for writing data. The API for reading is almost symmetric. It is also worth to note that there are minimal differences whether one read/write serial or in parallel. The API consists of a small number of C, C++ and Fortran functions and will be described elsewhere. In the parallel case the original domain decomposition can be used or the data can be decomposed according to the new number of processor nodes available. The resulting HDF5 file will contains a simple directory structure that can be navigated using the generic 'h5ls' utility;

```
if(parallel);
  filehandle=OpenFile(filename,mode)
else
  filehandle=OpenFile(filename,mode,mpicomm)
SetNumberOfParticles(filehandle);
loop(step=1,NSteps);
  SetStep(filehandle,step);
  WriteData(filehandle,fieldname1,data1);
              write more data
  WriteData(filehandle,fieldname<n>,data<n>);
CloseFile(filehandle);
```

Figure 2: Usage of H5Part in pseudo-code

## PERFORMANCE

Preliminary performance estimations, looking at global (GD) and local data (LD) rates, suggests that our HDF5 writing has a very good performance even with respect to raw mpi, as shown in Table **??**.

| Mode | GD [MB/s] | LD [MB/s] |
|---|---|---|
| mpi-io (one file) | 241 | 3.7 |
| one file per proc | 1288 | 20 |
| H5Part/pHDFf5 (one file) | 773 | 12 |

Table 1: 64 IBM SP-3 nodes writing $51e6$ particles (6D).

## CONCLUSIONS AND FUTURE WORK

The file format will be extended in the near future to integrate fast bitmap indexing technology [**?**] in order to provide accelerated queries of data stored in the file. With fastbit technolgy, a user can efficiently extract subsets of data using compound query expressions such as $(velocity > 1e6)$ *AND* $(0.4 < phase < 1.0)$.

We are also constantly tuning the performance of the parallel data file format implementation. We will also be porting the H5Part reader to a wider variety of visualization tools.

## REFERENCES

[1] A. Adelmann, R.D Ryne, C. Siegerist, J. Shalf, "From Visualization to Data Mining With Large Datasets," PAC, 2005.

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

[3] Definitions and libraries for SDDS implementation may be found at the link http://www.aps.anl.gov/asd/oag/oagPackages.shtml.

[4] K. Stockinger, J. Shalf, W. Bethel, K. Wu. "DEX: Increasing the Capability of Scientific Data Analysis Pipelines by Using Efficient Bitmap Indices to Accelerate Scientific Visualization." Scientific and Statistical Database Management Conference (SDDBM), 2005.