Lawrence Berkeley National Laboratory

Lawrence Berkeley National Laboratory

Title

Technical Highlights from the ExaHDF5 project

Permalink

https://escholarship.org/uc/item/7p85p1j9

Author

Wu, Kesheng

Publication Date

2012-04-10

eScholarship.org Powered by the California Digital Library
University of California

Technical Highlights from the ExaHDF5 project

Prabhat, Kesheng Wu, Jerry Chou, Suren Byna, Mark Howison, E. Wes Bethel
Lawrence Berkeley National Laboratory

Quincey Koziol, Peter Cao, Mohamad Charawi, Christian Chilan, Mike McGreevy
The HDF Group

Karen Schuchardt, Bruce Palmer
Pacific Northwest National Laboratory

DISCLAIMER: This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

Acknowledgments: This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the ASCR Exascale Scientific Data Management award. This work used computational resources provided by NERSC.

Technical Highlights from the ExaHDF5 project

Prabhat, Kesheng Wu, Jerry Chou, Suren Byna, Mark Howison, E. Wes Bethel (LBNL)
Quincey Koziol, Peter Cao, Mohamad Charawi, Christian Chilan, Mike McGreevy (The HDF Group)
Karen Schuchardt, Bruce Palmer (PNNL)

Keywords: Parallel I/O, Filesystems, Productivity Tools and Programming Models, Data Management, Data Models, Index/Query

Executive Summary:

The ExaHDF5 project consists of three thrust areas that address the challenges of data size and complexity on current and future computational platforms:

- We are extending the scalability of I/O middleware to make effective use of current and future computational platforms.
- We are incorporating advanced index/query technology to accelerate operations common to scientific data analysis.
- We are building upon our existing work on data model APIs that simplify simulation and analysis code development by encapsulating the complexity of parallel I/O.

Over the past year, we have made significant progress on all three areas. We have implemented the FastQuery software, which enables FastBit to operate on modern distributed memory, multi-core platforms. We have developed an MPI version of FastQuery to deal with distributed memory, and a multi-core version to utilize computational cores on a single node. We have applied FastQuery to index a massive 56TB dataset, and used 10K cores to resolve queries on the dataset in ~10 seconds.

We have identified a number of scalability bottlenecks in the HDF5 library and are working on improvements. We have examined the problem of removing the collective communication requirement for HDF5 metadata operations, and have settled on using a Metadata server as the most likely solution. We have designed a new HDF5 abstraction Virtual Object Layer, which will enable us to address scalability, synchronization and resilience in the near future. We have identified and implemented optimizations pertaining to metadata synchronization and file truncation. These enhancements will apply to *all users of the HDF5 library*, thus providing a “rising tide” that will “lift all boats”. This aspect of our work has broad applicability to computational scientists at DOE facilities such as NERSC and ORNL/ANL LCFs.

An important goal of the ExaHDF5 project is to work closely with several DOE code and science teams, to enable them to tackle realistic, large scale I/O and analysis problems. We have worked with science teams in the climate, groundwater modeling, accelerator modeling and astrophysics communities, enabling their codes to perform parallel I/O at 10s of thousands of cores, and at 1-2 orders of magnitude better performance. We have also incorporated HDF5 and netCDF-4 based data models into production codes, thereby enabling scientists to use parallel I/O capabilities on DOE’s supercomputing platforms in a productive fashion.

Figure 1 provides a schematic overview of the ExaHDF5 project. The next generation ExaHDF5 substrate forms the platform for the entire project. HDF5 performance enhancements, which will enable efficient, scalable I/O on current petascale and future exascale platforms, are highlighted in Section 2. Simulation codes (such as GCRM, CCSM, SPH, IMPACT, etc) provide the science drivers for the overall project. Specific examples of ongoing collaborations will be expanded upon in Section 4. These codes can utilize easy-to-use data models (Section 3) or advanced index/query techniques to dramatically reduce the time taken to perform analysis (Section 1).

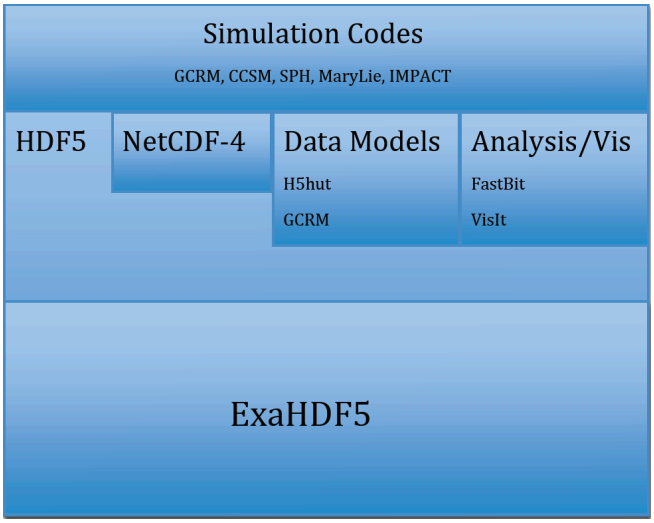

Figure1: Components of the ExaHDF5 project

1. Analysis

Modern scientific datasets present numerous data management and analysis challenges. State-of-the-art index and query technologies are critical for facilitating interactive exploration of large datasets, but we need to overcome a number of challenges to introduce these techniques to scientific data. Conceptually, the database indexing techniques are designed with relational data model as the basis, while most scientific data is based on array data model. This is the first challenge to be addressed. The second challenge is that the system needs to be able to run on distributed multi-core platforms, efficiently utilize underlying I/O infrastructure, and scale to massive datasets. In the past year, we have designed and implemented a preliminary version of FastQuery, a novel software framework that address these challenges [2,3,4]. FastQuery utilizes a state-of-the-art index and query technology (FastBit) and is designed to process massive datasets on modern supercomputing platforms.

1.1 FastQuery software design, development

FastQuery implements a dynamic mapping scheme to mediate between the relational data model used by database indexing methods and the array data model used to store scientific data. This mapping allows a combination of arrays and subarrays to be used together as long as they have conforming dimensions. The FastQuery design also allows the user to specify the groups and the array names separately, which allows one to easily process a group of arrays with the same name in different groups such as across time step or different runs of the same simulation.

Database indexing methods are typically designed for single processor computers and therefore can only be used effectively with a single thread on a parallel machine. To increase the number of parallel tasks, FastQuery introduces three levels of parallelism: file, variable, and subarray. In other word, FastQuery can process different files and different variables separately; it also automatically breaks up large arrays into subarrays to further increase the number of concurrent tasks available.

In terms of the software architecture, FastQuery has the following four main components:

The Query Processor and Index Builder form the programming interface for users of the FastQuery API. The index builder builds the indexes for a whole or a subset of a dataset and stores the indexes in a file. The query processor accepts text-string queries from the user, performs data selection using the stored indexes and returns the selection results to the users.

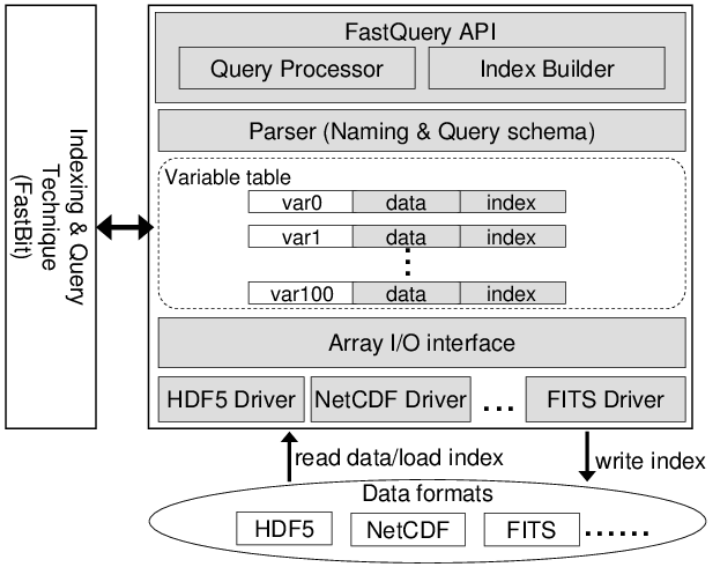

Figure 2: FastQuery software design

The FastQuery Parser is an internal component of FastQuery responsible for parsing user information given by the query processor and index builder. It defines and implements a simple, yet flexible, naming convention and query syntax for users to describe their requested variables and query constraints through the FastQuery API.
The Array I/O Interface defines an I/O API for FastQuery to access data and bitmap indexes stored in an array model file. The primary functions of the interface are to read data, and load and write indexes. We expect the user to implement file format (e.g. HDF5 or netCDF) specific calls to implement the API. Once the interface for the data format has been implemented, FastQuery can provide the data query and indexing functionality for the data format. This extensible design choice greatly increases the applicability of FastQuery to a wide range of present (and future) file formats.
The Variable Table represents the relational data model used by FastQuery and other indexing and query technology. A variable table contains a list of columns, each of which maps to a variable, and each row maps to a record of a variable. FastQuery applies FastBit to build and query indexes by reading the data and bitmaps associated with each record in the table.

1.2 Distributed FastQuery (w/ MPI)

Our first attempt at implementing FastQuery was to develop a MPI based version that runs an MPI process on each core of a parallel machine. This version has relatively straightforward file I/O pattern and data communication pattern. It can also be easily scaled up to take advantage of a large parallel computer. The graph in Figure 3 shows the aggregated I/O rates (in GB/s) of the three main components involved I/O operations: reading user data to be indexed, writing the bitmaps generated for the index, and writing the metadata related to the bitmaps. The set of measurements were gathered using 60 time steps of an accelerator modeling code called IMPACT-Z. The total data size is about 4 TB. The tests were conducted on hopper at NERSC using the Lustre file system. The file system can support a theoretical maximum throughput of 35GB/s. In our measurements, we see that this maximum is achieved during the reading step of index construction. The maximum observed speed for writing the bitmaps is about 20GB/s. Both these operations are perfectly parallelizable in the sense that the I/O operation on one MPI process does not need to coordinate with any other MPI process. The reading operation reads the same number of bytes on each process in the vast majority of the cases; therefore, it is better load-balanced and able to achieve higher I/O rate. The bitmaps from different processes are of different sizes; therefore, the maximum observed I/O rate is a little lower.

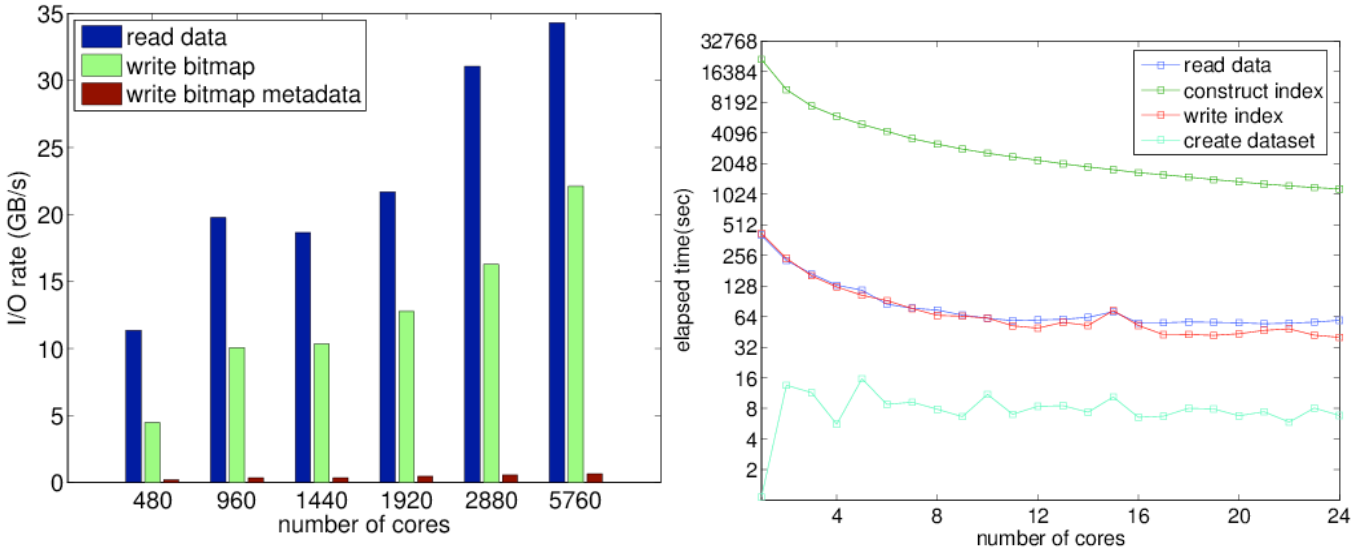

Figure 3: I/O rates obtained for the FastQuery distributed index creation pipeline (left). Scalability of the mullti-threaded FastQuery index creation pipeline (right).

1.3 Multi-threaded FastQuery (on multi-core)

The second effort in FastQuery work is to explore the multi-core parallelism present in modern chipsets. Figure 4 shows timing profiles from runs on a single node of hopper, which has 24 cores. As the number of cores increase from 1 to 24, the CPU time to construct indexes decreases by a factor of about 20, which is nearly perfect. The main reason for not achieving the perfect speedup appears to be the contention at the shared level 3 cache; the level 3 cache miss rate mirrors the changes in computation time quite well.

2. HDF5 Performance Enhancements

A major focus of the ExaHDF5 project is to enhance the performance of the HDF5 library to facilitate I/O at scale on current petascale and future exascale platforms. While we have worked closely with our science collaborators to profile and determine scalability bottlenecks in their present I/O patterns, these enhancements will apply to *all users of the HDF5 library*, thus providing a "rising tide" that will "lift all boats". This aspect of our work has broad applicability to computational scientists at DOE facilities such as NERSC and ORNL/ANL LCFs.

2.1 Optimizing HDF5 Metadata Operations:

Parallel HDF5 allows all processes that opened a file to access dataset elements independently or collectively. However due to synchronization issues, all operations that need to modify the structure of the file, or in other words the file's "metadata", are required to be collective. Breaking free from this requirement will allow applications to have much greater freedom when using the HDF5 API, and will enable performance enhancements that aren't possible with the current operational model. To achieve this goal, the following two goals must be achieved:

- Allow allocation of new space by an individual process, without involving all other processes
- Allow an object's metadata to be locked, updated, unlocked, and made visible to other processes without requiring a collective operation.

2.1.1 Space Allocation

Space allocation in parallel HDF5 applications can result in a race condition if processes do not synchronize with each other, causing multiple processes to believe that they are the sole owner of a range of bytes within the HDF5 file. Setting aside a process that acts as a space allocation server listening to requests from all clients will address this synchronization problem. This "space allocation" server will be incorporated into the "metadata server" solution, described below, and is included in that ongoing work. Naively implemented, the space allocation server could get bogged down with allocations

from many client processes, but when combined the metadata server solution below, we believe there will be no additional overhead to the application processes.

2.1.2 Independent Metadata Operations

We have performed a large amount of background research to come up with designs for breaking the collective requirement for metadata operations. The following design options constitute a subset of the options that were considered as more likely to provide the functionality needed to achieve our goal:

- Distributed Lock Manager. A distributed lock manager (DLM) is a commonly used approach to handle access to shared resources. The DLM is a set-aside process, and does not perform file I/O itself. All operations that read or modify metadata for an object need to acquire a lock on that object from the DLM.
- Metadata Server. A variation of the DLM approach is to have a metadata server (MDS) that actually opens the file and reads/writes metadata to the file. The MDS would perform all metadata operations requested by a client and coordinate changes between clients' requests internally. Different optimization and scalability approaches can be done in this approach, where we can have the notion of private (handled locally) and public (accessed through the MDS) objects, and split the actual file into two files, one for metadata and another for raw data, along with having multiple metadata server processes to spread the load from client processes and provide redundancy. This approach can be generalized to handle raw data, similarly to what ADIOS does with its staging area, but can also integrate with an existing staging infrastructure without duplicating it.

Simulator	1 Op.	2 Op.	100 Op.
ORIG 128	0.002701	0.003401	0.035983
MDS 128	0.003528	0.003715	0.029973
DLM 128	0.03801	0.054422	2.441472
ORIG 256	0.003089	0.004543	0.079608
MDS 256	0.006303	0.00638	0.066449
DLM 256	0.095199	0.151447	4.64511
ORIG 512	0.003713	0.006698	0.175682
MDS 512	0.01136	0.012178	0.147917
DLM 512	0.246199	0.451026	9.407203
ORIG 1024	0.004746	0.012626	0.458339
MDS 1024	0.053389	0.058904	0.372549
DLM 1024	0.694647	0.907409	20.43683
ORIG 5120	0.014127	0.092728	5.431242
MDS 5120	0.287151	0.314072	3.572677
DLM 5120	4.484003	6.557662	145.1383
ORIG 10240	0.110606	0.215091	8.376475
MDS 10240	1.183902	1.206859	6.794207
DLM 10240	14.44829	19.32646	368.3791

Figure 4: Examination of various options for scaling HDF5 metadata operations.

We implemented simple prototypes for the MDS and DLM designs to compare and contrast the communication performance with the original collective approach. Table 3 shows performance results from tests conducted on the NERSC 'hopper' system. The results are shown in seconds for a simulation involving 1, 2, and 100 operations. The MDS approach clearly outperforms DLM mainly because locking is handled locally at the MDS process.

2.1.3 Migrating the HDF5 library to the MDS approach: The Virtual Object Layer

The HDF5 data model is composed of two basic objects, groups and datasets. The main challenge currently in HDF5 is not in the data model itself, but in the methods of accessing a single native HDF5 file on large parallel systems. Storing an application's data in a single file benefits the application developer by keeping all the information for an experiment or timestep in a single location, and helps to reduce

stress on the underlying file system, compared to creating one file per process or node. However, the current method of accessing a single file has performance issues that vary widely over different platforms. Furthermore, in parallel applications, file access by several processes has to be coordinated in order to maintain strict consistency semantics to avoid file corruption. To address these issues, we are splitting the single HDF5 file into two separate files: one for metadata (managed only by the MDS) and one for raw data (accessed by all the other processes).

To implement the MDS approach, we propose to add a new abstraction layer internally to the HDF5 library, immediately beneath the public API. We call this new layer the Virtual Object Layer (VOL). The VOL intercepts all HDF5 API calls that could potentially touch the data in the file and forward those calls to a plugin "object driver". The plugins could actually store the objects in variety of ways, not just in the native HDF5 file format. A plugin could, for example, have objects distributed remotely over different platforms, provide a raw mapping of the model to the file system, or even store the data in other file formats (like the "classic" netCDF format). The user still programs to the HDF5 API and data model as if accessing a single HDF5 "container"; however the plugin object driver translates from what the user sees to how the data is actually stored on disk. This design fits well with the MDS approach described in the previous section, where the MDS is another plugin implemented internally in the HDF5 library. The following figure shows where the VOL is layered and how the data is accessed in the file:

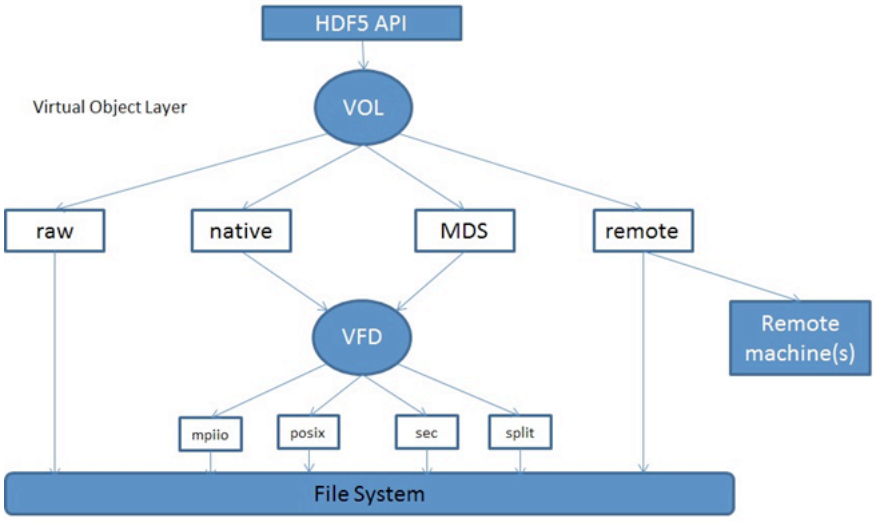

Figure 5: Proposed design of Virtual Object Layer with the HDF5 library

2.3 Testing and Deployment of HDF5 Enhancements

As part of our commitment to supporting HDF5 on large DOE systems, we are working to port the HDF5 daily regression test suite to NERSC's 'hopper' and 'franklin' systems. These test suites include standard HDF5 API features, as well as performance enhancements made under the ExaHDF5 project. All of our performance enhancements will be available to the scientific community via our production release process. Minor production releases of HDF5 are made twice a year, in May and November, and include enhancements that don't modify the file format or public API in non-backwardly compatible ways. Major production releases are made on a slower schedule, every 2-3 years, but are available for application developers to beta test during this development period.

3. Data Models

We believe that domain scientists and code development teams should be able to utilize efficient Parallel I/O capabilities on modern supercomputing platforms, without necessarily knowing all the details of the I/O hardware, and software stack layers. To that end, we continue to develop and enhance HDF5-based data models (H5hut for storing 1D particle and 2D/3D block structured data) and NetCDF-based data models (GIO for storing unstructured grids). These data models present an easy-to-use interface to the

scientists on one end, but on the other hand, exercise highly optimized I/O calls through the relevant API. We have transitioned the IMPACT-Z team to using the H5hut data model within the course of a few days. The SPH/STOMP teams have also been able to utilize the H5hut data model for storing particle data. We continue to develop the GIO layer, and engage with GCRM developers in further refining the climate data model.

4. Science Case Studies

An important goal of the ExaHDF5 project is to work closely with domain scientists and application teams, and enable codes to perform parallel I/O at large concurrencies. In this section, we report on our current collaborations with DOE code teams.

4.1 Accelerator Physics (IMPACT-Z)

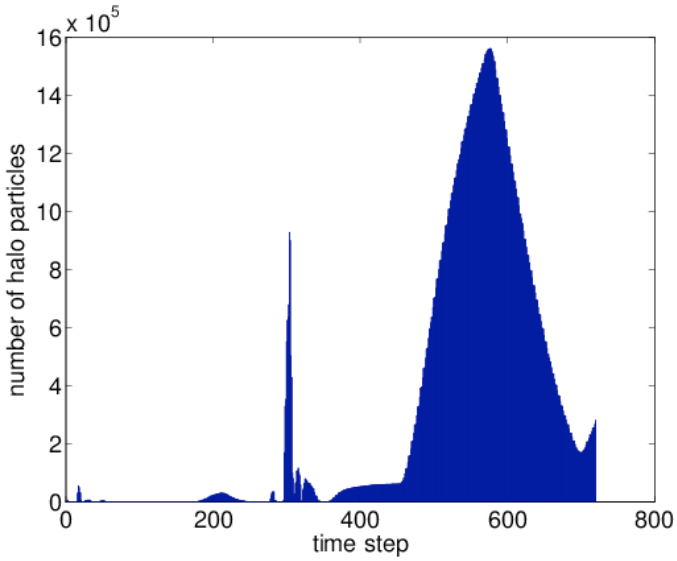

Figure 6: FastQuery analysis results from the 56TB IMPACT-Z run. The broad peak on the right illustrates an unusually high number of halo particles, indicating a sub-optimal beamline design. This scientific insight was fundamentally enabled by large-scale I/O and data analysis capabilities developed under the ExaHDF5 project.

We worked with the IMPACT-Z team based in LBL (Rob Ryne, Ji Qiang) and incorporated H5Part calls in the code within a short period of a few days. This collaboration facilitated the largest scale IMPACT-Z run till date: the team was able to generate 56TB of particle data on 10,000 cores[1]. We were able to achieve a write performance of 5GB/s on franklin (Cray XT4 system). We then used our distributed FastQuery software to index and query this massive dataset on ~10,000 cores. Perhaps the most exciting development from this collaboration is that both Rob Ryne and Ji Qiang were able to explore the entire simulation dataset for the first time, and they were able to confirm a hypothesis they had postulated for a long time, namely that the latter half of the NGLS beamline design was suboptimal. Using FastBit, they were able to compute a histogram of the halo particles in the beamline (shown in Fig 6) and confirm the presence of a broad peak in the latter half. This provided the team with important feedback for future beamline design optimization.

4.2 Climate (GCRM/CESM)

Our goal in this collaboration is to work closely with state-of-the-art climate models (GCRM and CESM) and facilitate large scale I/O for these important codebases. Climate models generate NetCDF files that use HDF as the underlying parallel I/O mechanism. We are taking a closer look at the NetCDF-4/HDF5 interface (as exercised through GCRM) and determining optimal strategies to perform efficient I/O. We use TAU extensively for our profiling needs. Achieving optimal I/O for the NetCDF interface currently requires work at several layers, the application I/O layer, the NetCDF4 HDF wrapper layer, and the HDF5 layer. The main change at the application layer is setting of optimal chunk sizes. Planned improvements

will move this functionality completely into the NetCDF layer. Improvements to date in the NetCDF layer include ensuring fill values (which results in redundant writes) can be completely turned off, reducing overhead in checking for the file magic type, efficiently loading the metadata in the NetCDF4 header, optionally deferring metadata writes until file close, and ensuring that outlier cases such as writing the scalar time value can be done efficiently. At the HDF5 layer, our profiling identified significant bottlenecks in the internal HDF5 metadata handling and file close methods. Section 2 discusses the solutions for these issues. The changes made to date throughout these layers have resulted in overall write performance improvements of 500%.

4.3 Groundwater (SPH/STOMP)

We have integrated the H5Part and H5Block libraries into both the Smoothed Particle Hydrodynamics (SPH) code and the exa-scale STOMP (eSTOMP) continuum code which together form the basis of a multi-scale subsurface model being developed by Scheibe et al under the SciDAC program (https://http://subsurface.pnl.gov). We are tuning I/O for each code separately, using real problems that require tens of thousands of processors for full simulations. For the SPH code, we have completed preliminary IO testing of 8 million particle geometry on both hopper and franklin. We have achieved bandwidths up to 4.5 GB/s on franklin and close to 9 GB/s on hopper.

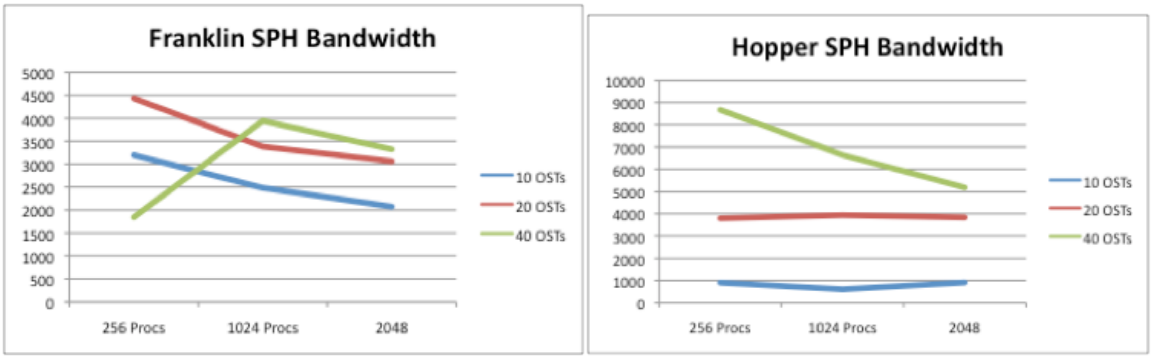

Figure 7: I/O performance for the SPH codes on Franklin and Hopper.

5. Future Work

We have made progress on a number of areas outlined in our project proposal. Over the next year, we continue pushing the research and software development frontier in a number of directions. We are interested in developing a hybrid parallel version of FastQuery and demonstrating scalability on ~100K cores. We are working on demonstrating effective I/O on an unprecedented 1 Trillion particle simulation on 128K hopper cores. In order to further hide the complexity of the I/O stack, we are developing an auto-tuning framework for the HDF5 API that optimizes I/O across HDF5, MPI-IO and Lustre. We will continue to explore promising ideas such as the Virtual Object Layer, and address scalability bottlenecks in the HDF5 and netCDF-4 layers. Finally, we will continue to work closely with various DOE code teams, solicit their feedback and demonstrate scalable I/O performance for real applications on platforms such as hopper and jaguar.

6. References

[1] Jerry Chou, Mark Howison, Brian Austin, Kesheng Wu, Ji Qiang, E. Wes Bethel, Arie Shoshani, Oliver Rübel, Prabhat, Robert D. Ryne: Parallel index and query for large scale data analysis. SC 2011: 30
[2] Jerry Chou, Kesheng Wu, Prabhat: FastQuery: A Parallel Indexing System for Scientific Data. CLUSTER 2011: 455-464
[3] Jerry Chou, Kesheng Wu, Prabhat: FastQuery: A General Indexing and Querying System for Scientific Data. SSDBM 2011: 573-574
[4] Jerry Chuo, John Wu, Prabhat, “Design of FastQuery: how to generalize Indexing and Querying systems for scientific data”. LBL Tech Report (number pending)