

HPC Specific Enhancement and I/O Performance Tuning of HDF5

The HDF5 library is an I/O library used by many codes at LLNL. Some codes use HDF5 directly and others use it indirectly through higher-level I/O libraries such as Silo or LEOS. Since its original development, HDF5 has been an important and beneficial part of I/O infrastructure at LLNL. As an example, HDF5's ability to perform application-level check-summing in-situ during I/O operations has served us well as an added guard against reliability problems we have occasionally faced on HPC filesystems.

Nonetheless, as the I/O and filesystem architectures of HPC platforms continue to evolve, it is essential that we invest development effort in HDF5 to ensure adequate I/O performance is consistently achievable. Recent experiences on BG/P (Dawn) and previous experiences on BG/L have demonstrated the serious consequences of failing to make that investment. For example, in the early months of 2010, it became necessary for LLNL to invest significant manpower to address I/O performance issues on BG/P systems. That investment has yielded encouraging results with performance gains of better than 50 times most of which were realized through HDF5. Where it once took 3 hours for a code to read (or write) a restart dump, it can now do so in about 3 minutes!

While these gains represent big savings and improvements in our ability to utilize BG/P, they have also come at substantial cost to LLNL in manpower and delayed programmatic work. Going forward, it will be more cost-effective for LLNL to contract with HDF5 developers (The HDF Group) for HPC specific enhancements and I/O performance tuning. That is the purpose of this contract and statement of work. In particular, this statement of work is aimed at supporting an “agile” development methodology where work on HDF5 is delivered incrementally and routinely and where priorities can be quickly adjusted to address continually evolving HPC I/O requirements over the next several years, a period which includes the delivery of the Sequoia platform. Six broad areas of work are involved. These are...

- Scalable I/O Performance Tracking, Testing and Tuning
- Virtual File Driver Enhancements
- HPC Specific Fast-tracking
- Parallel Interface Enhancements
- Exploratory Design Development
- User Support and Routine Maintenance

Scalable I/O Performance Tracking, Testing and Tuning

The key deliverable here is I/O performance. Most importantly, it is I/O performance at scales typical (> tens of thousands of processors) of LLNL HPC codes used in production computing and includes platforms at LLNL as well as platforms at

other computing centers where LLNL codes are run.

Some important elements of this area of work include development and adoption of a scalable I/O benchmark that is representative of LLNL codes and, in particular, their use of a “Poor Man's Parallel I/O” strategy – that is serial I/O to multiple, separate files, concurrently.

Other elements of this area of work include nightly or at the very least weekly testing of HDF5's main line development trunk at scale on a variety of platforms to ensure inadvertent introduction of performance degrading modifications of HDF5 are caught and corrected before making it into released versions of the product. This testing activity will include variation in compilers, MPI implementations and, where possible, filesystems such as Luster and GPFS and PVFS (Panasas)

HDF5 developers will require access to platforms at LLNL as well as other DOE sites where they can setup and monitor such tests on a routine basis. LLNL stakeholders will use their influence to help ensure such access is granted and maintained both at LLNL as well as other computing centers as necessary.

Another element of this area includes creation of tools and/or instrumentation of HDF5 to aid in the diagnosis of performance degrading behavior. This area also includes investigation and documentation (a performance recipe book) of the impact of various, existing library-wide and object-wide properties that may effect performance.

Virtual File Driver Enhancements

HDF5's low-level I/O operations are handled by something called a Virtual File Driver (VFD). A VFD is like a tiny *plugin* for handling low-level I/O operations to a storage system. This is where the rubber meets the road in I/O performance from the HDF5 library to files on disk.

The HDF5 library comes pre-packaged with a number of VFDs and they fall into two basic classes; parallel and serial. All of the HPC investments that have been made in HDF5 to date have been in the parallel VFDs. However, for a variety of very good reasons, too numerous to go into here, LLNL doesn't use these. In addition, it is not easily practical for LLNL to switch to using the parallel VFDs because of collective calling constraints they impose. LLNL uses the serial VFDs. However, none of these has been designed with HPC platforms and filesystems in mind.

So, the key deliverable of this area of work is a handful of new VFDs designed for HPC platforms such as BG/P and BG/Q to provide a few options for our codes. A prototype for one such VFD, a block-oriented VFD, has already been implemented for Silo and has demonstrated exceptional performance on BG/P.

Several conceptual designs for such VFDs have been outlined already by collaborations between LLNL and HDF5 developers. One of the more important of these for LLNL will be a VFD that *spoofs* a file-per-processor I/O paradigm whilst ultimately writing only a single (or small and controllable number of files shared by

subsets of the processors) shared file via MPI-IO. Another important VFD will be variation of the existing *core* VFD that allows an application to write files to the memory (core) of extra processors in the MPI communicator and, while the application returns to computation, data is drained from the files in core on the extra processors to the filesystem. Some other options include a *split* VFD that generates only a single file on disk with metadata tacked on in one large chunk at the end of the file or sprinkled in fixed size blocks throughout the VFD and a VFD that integrates with/supports with the SCR (Scalable Checkpoint and Restart) library.

In discussing some of the aforementioned VFD options with HDF5 developers, it is apparent that an overhaul to the existing VFD architecture may be essential to make good progress here. For example, the VFD interface could stand some improvement to make VFD development easier. Such work, if it is essential to make progress on the VFD designs outlined here, will be included as an integral part of this statement of work.

The intent is that the VFDs that are developed for this purpose become part of the HDF5 main source code and are maintained by HDF5 developers as a integral part of the library. The performance benefits of the resultant VFDs should continue to be maintained regardless of new features added to the HDF5 library in later years. If for example, some VFDs developed here to support HPC needs cannot be made to easily support these new features, fast-track options (see next section) will allow HDF5 clients to continue to use the VFDs provided they meet the required simplifying assumptions of use.

HPC Specific Fast-Tracking

The HDF5 library is a general purpose I/O library designed to offer a multitude of features related to I/O. However, not all of those features are relevant in typical, HPC I/O scenarios. For example, the ability to delete objects from a file and then reclaim the space in the file is NOT something HPC codes often or ever need. Likewise, the ability to do partial I/O on a given dataset is not often necessary. This is all the more true for codes using Poor Man's Parallel I/O and maintaining their data in domain decomposed pieces.

Most often, when HPC codes engage in I/O for restart or plot purposes, if they write or read any part of a dataset, they do it for the whole dataset in its entirety, always. This has implications for simplifying assumptions possible in the HDF5 library which may be used to by-pass (e.g. fast-track) subroutines as well as possibly inform HPC-specific VFDs regarding simplified I/O operations.

A list of possible HPC specific fast-track options will be developed and prioritized. A means for controlling the most important fast-track options within the HDF5 library will be developed and implemented in the library.

Parallel Interface Enhancements

Note: At time of writing, the HDF5 Group was awarded a “DOE FOA” contract that may tackle all of the parallel interface issues identified here. The degree of overlap between that contract and this needs to be evaluated. Where appropriate, we will leverage results from the DOE FOA contract and focus our efforts on other areas outlined here.

Presently, most LLNL HPC codes do not use HDF5's parallel interface due to the all-collective-all-the-time calling constraints. This is one of the key reasons LLNL has continued to use a Poor Man's Parallel I/O approach to scalable I/O. Silo's serial interface is yet another. It is not practical to re-tool our HPC codes to fit the current HDF5 parallel interface. It is far more practical to improve HDF5's parallel interface so that LLNL codes can more easily take advantage of it. If the HDF5 parallel interface can be improved to relax this all-collective-all-the-time constraint, this would represent a large (and necessary) step towards enabling our codes to conveniently do concurrent writes to a common, single file. There are a few options for doing this. These include...

- Deferred object creation with application driven synchronization of HDF5 metadata.
- A “journaling” approach to managing HDF5 metadata which simplifies file space allocations.
- Setting aside a small number of processors, apart from the application's, to manage HDF5 metadata.
- A multi-threaded implementation of the MPI-messaging necessary to manage HDF5 metadata across processors (this is like item 3, above, but using extra threads on existing processors instead of extra processors).
- The ability to manipulate objects in an HDF5 file that are decomposed over only a subset of processors.
-

Some of these approaches represent more risk than others. All the details of these approaches are not fully understood. So, part of the effort here involves design investigation, interface proposal and evaluation. In addition, it is possible that early investigative work could reveal another, presently unknown option, for improving the parallel interface to HDF5.

Exploratory Design Development

As mentioned in the preceding section, some of the ideas for improving HDF5's parallel interface require more design development to fully understand their impact and cost to develop. Likewise, LLNL users of HDF5 and HDF5 developers alike have a number of promising ideas for ways to improve I/O performance with HDF5 that require further design development to be fully fleshed out.

For example, one such idea is the notion of a “windowed” or “in-core” group

where the entire group's contents can be read or written in a small number of I/O requests that is independent of the number of objects in the group. This is much like the concept of an “in-core” file but controllable on a group-tree basis.

The impact of being able to possibly run a portion of the HDF5 library as *client code* on BG/P and/or BG/Q I/O nodes – a technique that has shown promise at other HPC sites – is another example of the kinds of activity this area of work encompasses.

User Support and Routine Maintenance

An essential component to the success of the work outlined here is that the HDF Group will be available to address issues related to this work on a routine and prioritized basis. Some of the work here will require some collaboration with LLNL experts. Intense development activities such as outlined here will no doubt reveal bugs that need fixing as well as require frequent and routine interfacing between LLNL users and the HDF Group. The option to have member(s) of the HDF Group travel to LLNL for purposes of interacting with LLNL users as well as possibly brief, focused development activities may be necessary. For example, having 1-2 visits per year from HDF Group for user interaction is probably appropriate. All of these activities represent a user support and maintenance burden that must be included and appropriately funded for the overall success of the project.

Tentative Timeline and Deliverables

Task Headline	Expected Quarter of Completion
Develop/Adopt PMPIO Benchmark and baseline it	Q2, 2011
Standup Weekly Testing of PMPIO benchmark	Q3, 2011
Extend Weekly Testing to other fs, compilers, mpis	Q4, 2011
Instrument HDF5 for performance diagnostics	Q3, 2011
Develop strategies/tools for diagnosing/fixing performance	Q1, 2012
Incorporate Silo block-based VFD	Q2, 2011
Improved VFD Interface	Q4, 2011
Single-File, PMPIO VFD	Q1, 2012
Single-File 'split' VFD	Q3, 2012
Remote Core VFD	Q1, 2013
SCR VFD	Q3, 2013
Inventory HPC Fast-track options	Q1, 2011
Implement high priority HPC Fast-track options	Q3, 2011
Implement lower priority HPC fast-track options	Q1, 2012
Deferred object creation and application-driven synch. (note: Possible overlap with FOA contract)	Q1, 2013
Compare Poor Man's and Rich Man's parallel I/O performance	Q2, 2013
Windowed, in-core, Groups	Q3, 2013
Explore HDF5 lib running on BG/P or BG/Q I/O nodes	Q4, 2012
Enhance HDF5 as informed from above investigation	Q4, 2013
User support and routine maintenance	ongoing