DEVELOPING HDF5 FOR THE SYNCHROTRON COMMUNITY

N. Rees, Diamond Light Source, Oxfordshire, UK
H. Billich, PSI, Villigen, Switzerland
A. Götz, ESRF, Grenoble, France
Q. Koziol & E. Pourmal, The HDF Group, Champaign, IL, USA
M. Rissi, Dectris AG, Baden, Switzerland
E. Wintersberger DESY, Hamburg, Germany

Abstract

HDF5[1] and NeXus[2] (which normally uses HDF5 as its underlying format) have been widely touted as a standard for storing Photon and Neutron data. They offer many advantages to other common formats and are widely used at many facilities. However, it has been found that the existing implementations of these standards have limited the performance of some recent detector systems. This paper describes how the synchrotron light source community has worked closely with The HDF Group to drive changes to the HDF5 software to make it more suitable for their environment. This includes developments managed by a detector manufacturer (Dectris - for direct chunk writes) as well as synchrotrons (DESY, ESRF and Diamond - for pluggable filters, Single Writer/Multiple Reader and Virtual Data Sets).

INTRODUCTION

The original initiative for this work came from a workshop held at Paul Scherrer Institute (PSI) in late May 2012. This was jointly organised by PSI and Dectris and brought together representatives from the synchrotron community and employees of The HDF Group. A number of issues were identified at that workshop, including:

1. The single threaded nature of the HDF5 library made it difficult to benefit from the multicore architecture of modern CPU's.
2. Compression was an important mechanism to improve data throughput in many synchrotron use cases, but including any form of compression limited the throughput of the library and also could not be used when writing in parallel with the Parallel HDF5 library (pHDF5).
3. Use of a hierarchical format like HDF5 led to the need to store many detector frames in one file. However, this increases the latency of any processing because HDF5 files could not be open for read and write simultaneously.

Since this meeting the authors have collaborated to fund a number HDF5 developments that have eliminated many of these problems for practical synchrotron use cases.

DIRECT CHUNK WRITES

This development was sponsored by Dectris and PSI and introduced a new function `H5DOwrite_chunk` in the high-level HDF5 library which bypasses the data gathering and scattering, data conversion, filter pipeline, and chunk cache and writes the data chunk to the file directly (see Fig. 1). This allows the user program to compress the data outside the library – potentially using parallel algorithms or hardware accelerators. It also avoids a number of data copies, which limits any dataflow through the filter pipeline to ~500 MB/sec on typical processors.

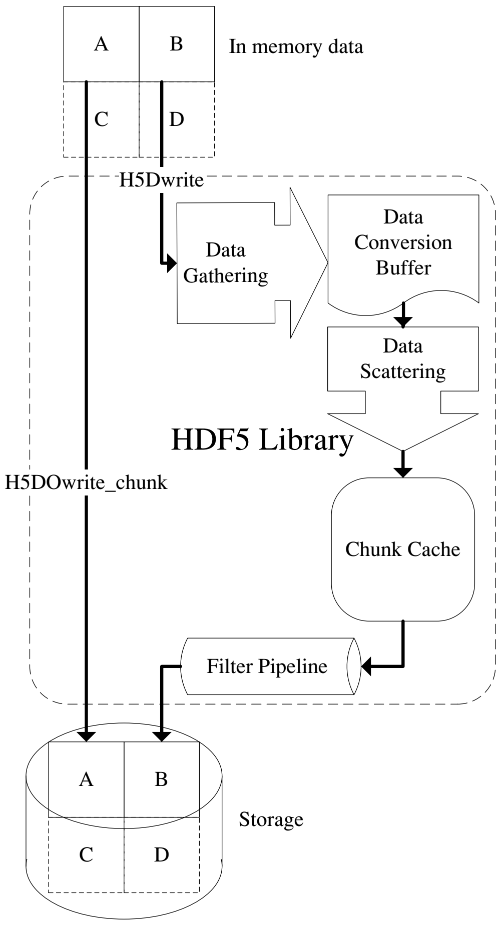

Figure 1: HDF5 data writing, showing the elements that `H5DOwrite_chunk` bypasses.

The feature was released in HDF5 version 1.8.11 in May 2013. The implementation and testing required about 5 months of work and, while the function did not present any technical difficulties, fitting it into the HDF5 library required some rethinking of the HDF5 source code architecture. The latest versions of the library are set for

ISBN 978-3-95450-148-9

expansion with other optimized functions if the need arises.

DYNAMICALLY LOADED FILTERS

This development was sponsored by DESY. As mentioned above, HDF5 implements data compression by means of a filter pipeline. The library comes with several predefined filters including GZIP and SZIP compression algorithms [3] but SZIP, for example, has license terms that may restrict data distribution. Those algorithms are also very general and so may not achieve optimal performance in terms of speed and/or compression ratio for a specific experimental application. Also prior to this development custom compression methods had to be incorporated into the main HDF5 library at link time and this further limited the way filters could be used.

With the new feature, the filters can be dynamically loaded at run-time which allows application domains to develop specialist filters suited to their requirements without the need for integrating them within the HDF5 code base or passing support to The HDF Group. When a filter DLL or a shared library is available on the system, HDF5 tools and user applications can use an off-the-shelf HDF5 library to read and modify data transparently to a user's application. Each filter has a unique identifier and filter developers are encouraged to register their filters with The HDF Group to avoid identifier clashes and allow a list of available filters to be published (see https://www.hdfgroup.org/services/contributions.html).

Writing, testing and distributing custom HDF5 filters is now a community effort with The HDF Group just providing guidance. The HDF Group, DESY and Dectris are working on a repository of the dynamically loaded filters (https://svn.hdfgroup.uiuc.edu/hdf5_plugins/), so users can download code and binaries for filters they need.

SINGLE WRITER/MULTIPLE READER

With the transition from less structured data to a structured data format that is related to the goals of an experiment comes the requirement to store all data for an experimental scan (or even experiment) in a single file. This groups like data together and makes data storage more efficient at high frame rates – where in the past each frame and its metadata had been stored in separate files. However, since HDF5 did not allow simultaneous reading and writing by separate threads or processes, this either meant that any processing of the file had to wait until the scan completed, which in extreme cases can be several hours, or that data servers had to be written to handle all the data reading and writing, creating a data bottleneck.

Single Writer/Multiple Reader (SWMR) allows a single process to update datasets in an HDF5 file whilst multiple other processes are reading data from the same datasets. To solve the difficulties involved with allowing concurrent access to write and read processes, HDF5 developers used an approach that is lock-free and that does not require any communication between processes. All communications are done through the HDF5 file itself.

In previous versions of HDF5 the file organisation on disk was only guaranteed to be consistent if it wasn't open for writing because data storage metadata (that describes how the data is laid out on disk) was kept in the writers cache until the file was closed. The approach taken for the SWMR development was to modify the writer's metadata cache and file data structure code to ensure that metadata that is referred to by another metadata object is always flushed from the cache first. This ensures that a reader will never attempt to resolve an invalid offset and has the secondary benefit that the state of the file on disk is always consistent and cannot be corrupted by an application crash. The parent-child relationship between the metadata objects is called a flush dependency.

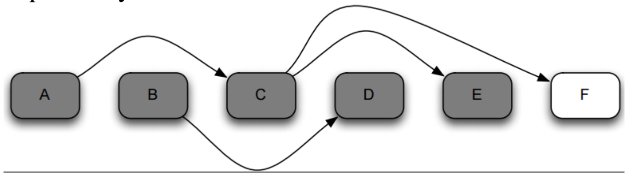

Figure 2: Representative flush dependencies between metadata cache objects.

An example of the flush dependency algorithm is shown in Fig. 2 where entry A is the parent of C, B is the parent of D, and C is the parent of both E and F. Dirty entries are shaded (A-E), and clean entries are not shaded (F). With this configuration, A could not be written to the file until C is clean, and C could not be written to the file until both E and F are clean. Likewise, B could not be written to the file until D was clean. So, with this configuration, when the metadata cache is attempting to flush dirty entries to the file (perhaps when flushing all the metadata for the file), either entries D and E could be written to the file first (making them clean), then entries B and C, and finally entry A. If C was written before B, A would also be able to be flushed before B.

This algorithm is based on the assumption that the HDF5 file resides on a file system compliant with POSIX I/O semantics. There are two major features of POSIX I/O semantics that are critical for SWMR use:

- POSIX I/O writes must be performed in a sequentially consistent manner.
- Writes to the file must appear as atomic operations to any readers that access the file during the write operation.

These semantics apply to any processes that access the file from any location. The caveat is that some file-systems (notably NFSv3) do not abide by these semantics and they cannot be used since reading processes do not see changes to the data in the same order as it is written.

The feature presented significant technical difficulties and required a lot of rework of the HDF5 library architecture. It also triggered changes to the HDF5 File Format introducing new chunk indexing structures with enabled checksums on the metadata items.

The feature leveraged the work done by The HDF Group developers for another commercial customer who funded the effort for several years.

The completion of the SWMR development for "append-data" only writers [4] was sponsored by Diamond Light Source, ESRF and Dectris. The development was coordinated, but each organisation contributed to a different work package of the development and so funding was kept separate.

The development comprised about twelve months of the effort and was delivered as a specialized version of the HDF5 library to DLS, ESRF and Dectris. The HDF Group has been working on merging the feature with the main stream of HDF5 and delivering it in the HDF5 release version 1.10.0.

VIRTUAL DATA SETS

The virtual dataset concept has the most interesting history in that it was developed last and has evolved the most from the original ideas in the PSI workshop in 2012. The original driver was storage of data from high-speed detectors. Two approaches were considered to maximise the storage throughput – one was compression (for example, the Pilatus detector used the imgCIF packed compression) and the other was writing data in parallel. The problem was that pHDF5 was not able to write compressed data, and so a proposal called "Parallel Compressed Writer" was generated. As the name implied, one of the initial design ideas was to allow compression in pHDF5, but the complexities of this was a problem and it was also discovered that the performance per process of pHDF5 was not as good as single stream HDF5 because of the MPI overhead on the nodes used for writing.

After some discussion another approach was proposed which extended the concept of the HDF5 dataset soft-link to allow for a dataset to be comprised of a union of datasets from a number of other files. These files could then be written independently and in parallel and since the writers wouldn't use pHDF5 they could also be compressed. When The HDF Group saw this proposal they connected it with use cases from other domains far beyond just high-speed detectors. Consequently, the idea was generalised so that the parent dataset could be composed of an arbitrary mapping of child datasets, and the name of "Virtual Dataset" was coined [5].

Figure 3 shows a simple mapping of source datasets to a virtual dataset, but much more complex mappings are possible. The mappings can be any n-dimensional rectangle and gaps can be filled with user specified fill values. These gaps can be both gaps between the maps or, in a data acquisition application, gaps representing data that hasn't yet been written to the underlying datasets. Hence virtual datasets can be used for sparse data, or for combining data from different sources to be used by an application that wasn't considered when the original files were written. Virtual dataset capabilities are particularly useful in an environment where multiple data sources need to update an HDF5 dataset simultaneously. The child files can all be written by different SWMR processes and readers can still open and read the parent dataset without conflict, and periodically poll the dataset metadata to determine if additional data has appeared.

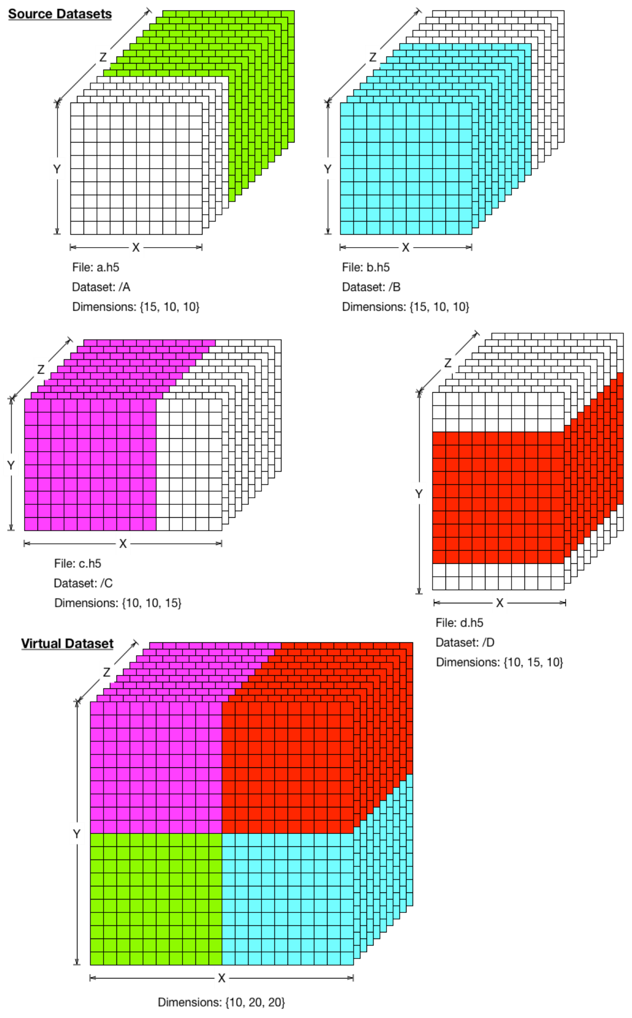

Figure 3: Simple mapping of source datasets to a VDS.

The feature implementation required adding a new storage layout to HDF5 and triggered improvements to the HDF5 library code. The total implementation effort took seven months of the developers' time. Another two months will be spent on merging VDS with the HDF5 mainstream library and preparing the feature for the HDF5 1.10.0 release.

The Virtual Data Set project was jointly funded by DESY, Diamond and the Percival detector project. Since the project could not be broken down into appropriate sized work packages, Diamond Light Source coordinated the work and all three parties contributed equal amounts.

CONCLUSIONS

The main point of this paper is to demonstrate how some of the major synchrotron light sources have collaborated to develop our common software for the benefit not just of the stakeholders, but for the entire community. These developments were different to many of the collaborative software developments we have undertaken in the past (e.g. EPICS and TANGO) in that all the development was done commercially rather than in-house. This has created challenges because money rather than manpower had to be committed and this has required a strong motivation and champion for the development. However, it has also proved to be productive because the developments have been delivered largely to time and to the agreed budget.

The largest time delay has come for the latter two projects because they have involved a change in the underlying HDF5 data file format. This has meant that they cannot be released as an HDF5 point release, but have had to be delayed until the next major HDF5 release (HDF5 1.10). This has slowed the uptake of the developments and we look forward to the release of HDF5 1.10 late in the first quarter of 2016.

REFERENCES

[1] The HDF Group. Hierarchical Data Format, version 5, 1997-2015. http://www.hdfgroup.org/HDF5/

[2] NeXus. A coomon data format for neutrons, x-ray and muon sources. http://www.nexusformat.org

[3] The HDF Group. “Using compression in HDF5”, https://www.hdfgroup.org/HDF5/faq/compression.html

[4] The HDF Group. “HDF5 New Features”, https://www.hdfgroup.org/HDF5/docNewFeatures/

[5] The HDF Group. “HDF5 Virtual Dataset”, https://confluence.hdfgroup.org/display/UF/HDF5+Virtual+Dataset