

# Using collective IO inside a high performance IO software package – HDF5

MuQun Yang, Quincey Koziol  
National Center for Supercomputing Applications  
University of Illinois, Urbana-Champaign

## I. Introduction

HDF5[1] is a data format and software library for storing scientific data. The software is open-source and free. It is based on a general data model and emphasizes standards and flexible, efficient IO.

HDF5 provides parallel IO support through Message Passing Interface Input and Output (MPI-IO). One way to do parallel IO through MPI-IO is for every process IO to function independently. No communication among processes is required, but significant disk IO latency may be expected when performing non-contiguous subsetting of arrays. MPI-IO provides an alternate option, collective IO, to support efficient IO for non-contiguous subsetting of arrays. Although it involves network communications among processes, the interleaved IO requests from different processes can be combined into a single contiguous operation to yield a significant speedup [2].

HDF5 has supported collective IO since HDF5 Release 1.2 in 1999. In 2003, Argonne National Lab and Northwestern University implemented a parallel version of NetCDF that also supports collective IO[3]. NetCDF is a widely used scientific data format originally developed at Unidata[4].

Recently, NCSA's HDF group has used the FLASH-IO benchmark, which simulates the IO pattern of astrophysical thermonuclear flashes, to compare IO performance among parallel HDF5 in independent mode, parallel HDF5 in collective mode, and parallel NetCDF in collective mode. Fig.1 shows an example of such a study on an IBM Power5 SMP 8-processor-per-node cluster at Lawrence Livermore National Lab. The results show the IO bandwidth of parallel HDF5 in collective mode to be almost equivalent to that of parallel NetCDF. Both greatly outperform the case of parallel HDF5 in independent mode. In this example, both HDF5 and parallel NetCDF store the data in a contiguous data stream on disk; it turns out that this storage option is appropriate for the FLASH-IO benchmark.

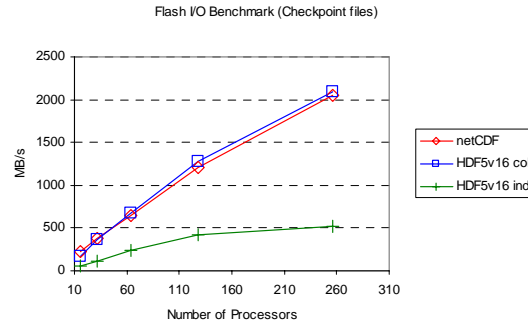


Fig. 1: Comparison of aggregated IO bandwidth of the FLASH-IO benchmark among parallel NetCDF collective mode, parallel HDF5 collective mode, and parallel HDF5 independent mode on IBM POWER5.

However, HDF5 also allows array data to be stored on disk in chunked format. Chunked storage is also called data tiling for the two-dimensional instance. It can improve IO performance when subsetting large arrays[5]. It is required if extensible dimensions are defined for raw data or when a filter, such as deflate compression, is applied to an HDF5 dataset. Extensible dimensions and in-memory data compression are widely used in parallel scientific applications such as WRF[6] and ROMS[7]. Achieving high IO performance with chunked data sometimes requires collective IO.

To achieve high performance of collective IO in chunked storage is challenging. In section II we will explore the difficulties and our approach to this issue.

Parallel HDF5 is widely used by many scientific applications on high performance computing systems such as Teragrid. Since parallel HDF5 needs to use features supported in MPI-IO to do collective IO, intensive tests had to be performed with various MPI-IO packages during development of the HDF5 software. In section III we will share our experiences in handling the difficulties of utilizing MPI-IO packages on several high performance computing systems.

## II. Internal software management to support collective IO with chunked storage

### 1. Description of HDF5 irregular selection with MPI-IO derived datatype

HDF5 can support both regular and irregular selections. A regular selection, sometimes also called a hyperslab, is a selection generated inside an HDF5 program with only one `H5Sselect_hyperslab` routine call [1] for the selected data space. An irregular selection is a selection generated inside an HDF5 program with more than one `H5Sselect_hyperslab` routine call [1].

Figure 2 illustrates regular selections with a two-dimensional data array.

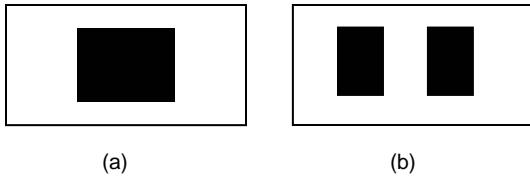


Fig.2: Illustrations of regular selections. Shaded regions represent selections.

Figure 2(a) illustrates a singular selection that covers one logically contiguous block. Figure 2(b) illustrates a non-singular selection that covers more than one logically contiguous block. The shape of each individual block is the same and the stride between adjacent blocks is equal.

There can be many kinds of irregular selections in an HDF5 dataset. Figure 3 presents examples of irregular selections.

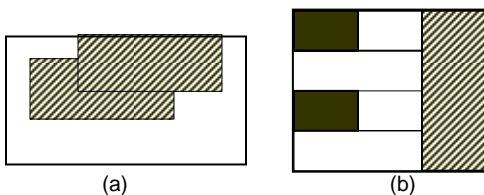


Fig.3: Examples of irregular selections. Shaded regions represent selections.

Figure 3(a) shows a case of one contiguous block that is overlapped by a similar contiguous block. Figure 3(b) shows an irregular selection that includes two regular but dissimilar selections in one dataset.

It is relatively straightforward to support collective regular selections in chunked storage and it is true that most known HDF5 applications use only regular selections. However, the shape of a selection that is regular across an entire dataset may

become irregular within individual chunks in chunked storage. Figure 4 illustrates the change. The selection is regular for the complete array. However, for each chunk the selection becomes irregular. Furthermore, HDF5 allows different shapes of selections between the memory data space and disk data space. The mapping of a selection from disk to memory may change the shape from regular to irregular. Finally, some HDF5 applications may need to use irregular selections in the future. So a more general description of HDF5 selections in terms of MPI-IO is needed.

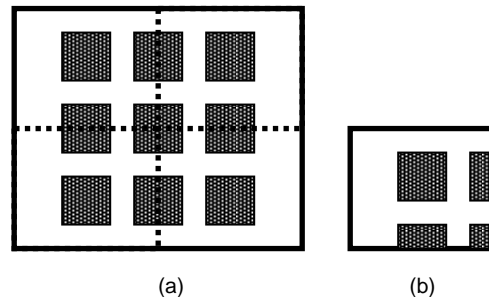


Fig.4: An illustration of the change of type of selections in chunked storage. The shaded area is the selection.

Fig.4(a) shows a regular selection in the whole data space, which includes four chunks. The logical boundaries of the chunks are represented by the dashed line. Fig.4(b) shows that the original regular selection becomes an irregular selection within a chunk.

MPI-IO provides an MPI derived datatype that can not only describe irregular selections but can also be easily connected with the internal HDF5 data structure that describes irregular selections. The MPI derived datatype is a user-defined datatype based on MPI predefined datatypes and other user-defined datatypes. An MPI derived datatype can only be built through MPI derived datatype function calls. Within each chunk, an MPI derived datatype is used to describe the layout of selections in the file and in memory.

The MPI type function calls `MPI_Type_contiguous` and `MPI_Type_vector` describe regular selections. `MPI_Type_indexed` and `MPI_Type_struct` describe irregular selections. The MPI-IO function `MPI_FILE_SET_VIEW` is then used to receive the description of selections with the MPI derived datatype and to pass these descriptions to MPI-IO, which will then perform the collective IO.

## 2. Optimization 1 – One linked-chunk IO

Generally the HDF5 Library only allows an application to do IO on a per-chunk basis. That may cause performance degradation if a selection covers many small chunks. One means of overcoming this problem is to create an MPI derived datatype across all of the relevant chunks. Internally, HDF5 uses a B-tree to store the physical chunk address. Due to the support of extensible dimensions in HDF5, one should be aware that the physical address of the starting point of each chunk may not increase according to the order in which it is created. Sorting the starting physical addresses of all chunks is therefore necessary before creating the MPI derived datatype across all chunks. For each process, the final MPI derived datatype internally links all MPI derived datatypes created in the chunks that the selection covers. Fig. 5 illustrates the collective view of how such MPI derived datatypes are created with two processes across four chunks. This will tremendously improve the performance for parallel applications when multiple selections involve many small chunks to cover selections.

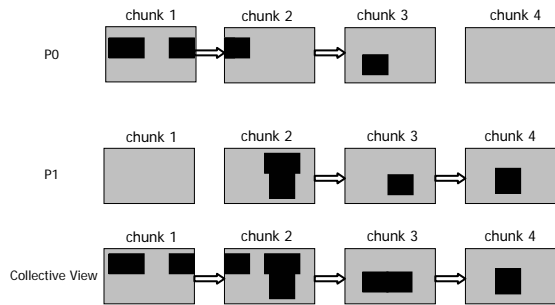


Fig.5: A schematic illustration of an MPI derived datatype that links multiple chunks

## 3. Optimization 2 – Multi-chunk IO

One linked-chunk IO can improve parallel performance. However, not all MPI-IO packages fully support collective IO with complicated MPI derived datatypes (we will further discuss this topic in section III). Another disadvantage of using one linked-chunk IO is that it may require excessive memory and performance may degrade, possibly causing a system memory error. In such a case, we want to retain the original implementation which performs collective IO per chunk. We call this a multi-chunk IO approach.

The trade-off with collective IO is the involvement of additional inter-process communications. Usually the more processes participating in IO and the closer the disk layout of

selections from all processes, the better the chance that collective IO can outperform independent IO. Unfortunately, this situation may not always apply to HDF5 applications that do IO per chunk because each process will not necessarily have selections in every chunk, as illustrated in Fig.5. Collective IO could be done with different chunks from different processes, but physical locations on disk of logically adjacent chunks may be distant. Furthermore, the complex selection resulting from chunked storage may cause the collective IO component of an MPI-IO package to fail to optimize IO accesses and achieve good performance, compared to independent IO.

To overcome this problem, HDF5 internally determines whether it is appropriate to use collective IO for each chunk based on the percentage of processes having selections in the chunk. If the library finds that it is not beneficial to use collective IO for the current chunk, the library will use independent IO for that chunk.

So, contrary to one linked-chunk IO, we now have to do IO for each chunk. This approach provides better control over collective IO on chunked data inside HDF5. The disadvantage is that this approach requires extra MPI gather and broadcast collective calls to finish the job.

## 4. Software management – User-input options

So far, we have presented two options for optimizing collective IO in chunked storage in HDF5. Each option may achieve improved performance for some kinds of applications. We also realize that the HDF5 Library may make the wrong decision about the way to do collective IO with chunked storage in some applications.

Therefore, the HDF5 Library provides several APIs to give applications more control over the way collective IO is done on chunked data. Figure 6 shows the internal procedure the HDF5 Library employs to handle a collective IO request in chunked storage. Through participation in the decision-making process for the collective IO request, the application has more control over the HDF5 Library and may see better performance.

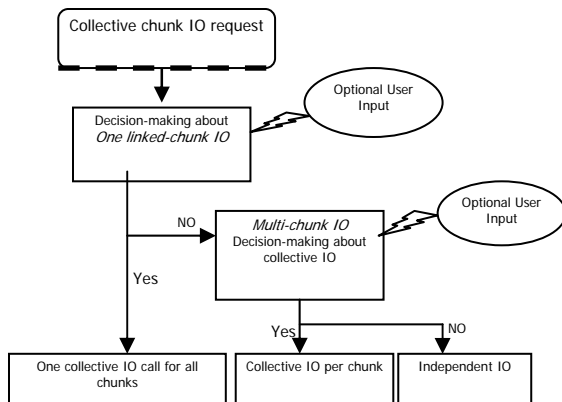


Fig.6: Flow chart of procedure to handle a collective IO request in chunked storage

### III. Experiences with MPI-IO packages

The HDF development group has committed to fully support parallel HDF5 on several key platforms such as Teragrid Linux clusters and IBM AIX SMP clusters. But the collective IO support inside HDF5 relies on the robustness of MPI-IO collective IO calls and makes full use of the MPI derived datatype technique.

In the process of testing HDF5 collective IO features, we frequently find bugs in MPI-IO packages. Most bugs have been fixed in newer versions of these MPI-IO packages. However, many HDF5 applications rely on computing architectures that still use an older version of an MPI-IO package.

On the one hand, we have to provide work-around solutions to assure that HDF5 does not break those applications using older versions of MPI-IO packages. On the other hand, we would like applications to take advantage of our new collective IO support if they are using a newer MPI-IO package. Therefore, we currently disable some collective IO optimizations in situations with known MPI-IO problems, otherwise leaving those optimizations enabled.

We also realize that there are some MPI-IO packages on some platforms that we have not tested and we do not know whether they also have these known bugs. Ideally, we would prefer to figure out their functional status during the HDF5 installation process with simple MPI-IO programs. However, the currently available automatic configuration techniques don't allow us to do so because some parallel programs can only be run in batch mode.

Therefore, we collect MPI-IO programs that can detect specific bugs and include them in the HDF5 parallel test suite. When the parallel HDF5 Library is installed, the MPI-IO test programs will fail if the current MPI-IO package has a known bug. The installer can then follow the guidelines in the HDF5 release notes and manually modify the configure file on that platform to fix the error. We further ask the installer to report this result to the HDF helpdesk.

### IV. Conclusion

Supporting collective IO for chunked storage inside HDF5 is not trivial. Several implementation options and controls are required to assure good performance.

Users can choose to do one collective MPI-IO across all chunks. Alternatively users can do IO per chunk using HDF5 optimizations.

Furthermore, user applications can improve IO performance by participating in the decision-making process.

Because the collective IO support for chunked storage in HDF5 relies on the robustness of MPI-IO packages in handling collective IO, user assistance is valuable in testing and improving the correctness and performance of those packages. Currently, more software maintenance work needs to be done to achieve these goals in both the HDF5 Library and MPI-IO packages.

More information about HDF5 can be found at <http://hdf.ncsa.uiuc.edu/HDF5/>.

### References

1. HDF5: <http://hdf.ncsa.uiuc.edu/HDF5/>
2. William Gropp, Ewing Lusk, Rajeev Thakur. : Using MPI-2. The MIT Press, 51-118, 1999.
3. Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad Gallagher, Michael Zingale. "Parallel netCDF: A High-Performance Scientific I/O Interface," SC'2003 November 15-21, 2003, Phoenix, Arizona, USA.
4. Unidata: <http://www.unidata.ucar.edu/>

5. HDF chunking concept:  
[http://hdf.ncsa.uiuc.edu/UG41r3\\_html/Perform.fm2.html#149138](http://hdf.ncsa.uiuc.edu/UG41r3_html/Perform.fm2.html#149138).
6. WRF: <http://wrf-model.org/>
7. ROMS:  
<http://marine.rutgers.edu/po/index.php?model=roms>

Chilan at the University of Illinois for their work with the FLASH-IO benchmark on the IBM POWER5 SMP cluster at LLNL to create figure 1.

This paper is funded by National Science Foundation Teragrid grants SCI 0504064 and SCI 0451538, the Department of Energy's ASC Program under contract LLNL B527300 and the Cooperative Agreement with NASA under NASA grant NNG05GC60A. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

#### **Acknowledgements:**

The authors would like to thank Frank Baker of the NCSA HDF group for his great help in editing this paper. The authors would also like to thank Leon Arber and Christian