PARALLEL I/O PERFORMANCE STUDY WITH HDF5, A SCIENTIFIC DATA PACKAGE

Christian M. Chilan
MuQun Yang
Albert Cheng
Leon Arber

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

1. Introduction

The amount and complexity of data used in scientific applications demand a portable standard for flexible and efficient access across several computing platforms. Two libraries, Hierarchical Data Format 5 (HDF5) [1] and Network Common Data Form (netCDF) [2], have been able to provide the scientific community with suitable general-purpose data formats and programming interfaces.

HDF5 is a widely used portable file format and library developed at the National Center for Supercomputing Applications (NCSA) for storing, retrieving, analyzing, visualizing and converting data. HDF5 stores multidimensional arrays along with metadata in a portable file. It supports hierarchical file structures providing users with a high degree of flexibility for data management. HDF5 also provides support for parallel data access built on top of MPI-IO, which is specified by the MPI-2 standard [3]. In this way, HDF5 can take advantage of MPI-IO optimizations. Additional features such as variable-size arrays and data compression are made possible by partitioning the storage space into chunks.

NetCDF[2] is another portable file format and programming interface broadly utilized in the scientific community for data access and storage of structured datasets. NetCDF uses a linear data layout which stores data arrays in a contiguous space or interleaved in a regular pattern. This simple and efficient method for data storage minimizes the overhead in I/O operations and allows for arrays to have one dimension (the most significant) with variable size. Parallel netCDF (PnetCDF) [4] is a parallel version of netCDF developed by Argonne National Laboratory and Northwestern University. It provides support for parallel access built on top of MPI-IO which allows possible optimizations.

In this paper, we examine the results of an earlier performance comparison which showed that PnetCDF provides overall higher performance than HDF5 [5] and we explain the reasons for such behavior. We also discuss the effects of independent and collective operations on I/O performance using several test cases. We also intend that HDF5 users reading this paper develop a notion of what type of access pattern is to be preferred (or avoided) when using a particular I/O mode.

2. HDF5 and PnetCDF performance comparison

Published results from an earlier performance study suggest that PnetCDF achieves higher parallel I/O performance than HDF5 [5]. To reproduce the given results and to provide an analysis for performance, we carried out similar testing.

The study was made using two high-performance parallel computing systems. The first system is NCAR Bluesky, an IBM Cluster 1600 system with AIX 5.1 and General Parallel File System (GPFS) [6]. Each node has 32 Power4 processors and 4 ports to the switch (SP Switch2) which provides 1 GB/s of bandwidth. The second system is LLNL uP, an IBM SP system with AIX 5.3 and GPFS. Each node has 8 Power5 processors. A Federation switch provides 122 GB/s of bandwidth. The versions of the parallel libraries are PnetCDF 1.0.1 and HDF5 1.6.5.

The benchmark used is the I/O kernel of FLASH [7], an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. We configured FLASH I/O [8] to generate 3D adaptive mesh refinement (AMR) blocks of size 8×8×8 on Bluesky, and 16×16×16 on uP. Each processor handles approximately 80 blocks, and writes them into three output files. The largest file is a checkpoint file containing all of the data from the blocks. The other files are visualization files that contain center and corner data. In our study, we refer only to the writing performance of the

checkpoint file because its size allows for more stable and relevant measurements. In fact, each processor writes about 8 MB and 60 MB into the checkpoint file when using 8×8×8 and 16×16×16 blocks, respectively. The performance metric provided by FLASH I/O is the parallel execution time in seconds. The aggregate bandwidth is obtained by dividing the file size over the reported time.

FLASH I/O makes use of the PnetCDF libray in collective mode. Collective I/O operations combine the noncontiguous data requests of each process into a single contiguous I/O operation. One important result is that the contribution of access latency, usually the costliest I/O factor, is minimized, improving performance significantly [3]. For HDF5, we performed tests using both collective and independent I/O operations. Also, the HDF5 tests used contiguous storage, which is the default configuration.

The results of our study in Bluesky and uP are shown in Figures 1 and 2, respectively. The figures show the best results from a set of three executions in order to avoid results hindered by concurrent processing loads.

Both figures show that the performance of FLASH I/O using PnetCDF and collective HDF5 scales well with respect to the number of processors. The small decrease in performance present during the transition from 16 to 32 processors in Figure 1 is mainly caused by a larger demand of resources while the number of switch ports remains constant (32-way node).

On the other hand, HDF5 in independent mode does not scale properly. Since the published study used HDF5 in independent mode, the large performance difference we found between PnetCDF and independent HDF5 agrees with the results of that study. Such difference is nominally the same variation in performance between collective and independent I/O operations in FLASH I/O.

A close examination of the access pattern of FLASH I/O reveals that each processor writes a contiguous region of approximately 80 blocks at a time, and that such regions are placed one next to the other. Some MPI-IO implementations may yield only a minor difference between the performance of independent and collective I/O operations for this type of access [7]. However, the MPI-IO implementation on IBM systems, MPI-IO/GPFS [9], provides features that can optimize collective operations even for this access pattern.

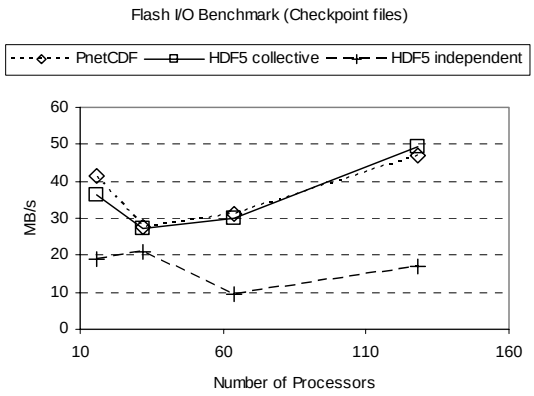

Figure 1: FLASH I/O performance on NCAR Bluesky

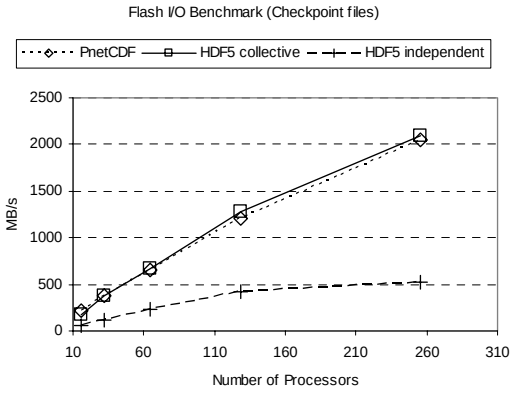

Figure 2: FLASH I/O performance on LLNL uP

MPI-IO/GPFS uses a file system in which the storage of a file is distributed across several GPFS servers. Stripes of consecutive GPFS blocks are bound to an I/O agent on each MPI task in a round-robin fashion. When a task requests data from a block other than the ones it has been assigned, the I/O agent designated to that block performs the access to the file system and then ships the data to the requesting task. This feature is known as data shipping and it is enabled by default. In order to manage large files efficiently, the size of a GPFS block is set to 1 MB, and each stripe contains 16 blocks by default. The stripe size matches the buffer size of each I/O agent. Large file structures help reduce latency and make better use of bandwidth.

Because the size of the contiguous region to be written by each processor at a time in FLASH I/O is ~0.31 MB and 2.5 MB when using 8×8×8 and

16×16×16 blocks, respectively, it is evident that several processors may request access to the same stripe of size 16 MB. In an independent operation, the I/O agent assigned to the stripe performs a separate file access for each request. However, a collective operation will cause the I/O agent to combine the many requests into a single contiguous access reducing latency costs and improving performance [9].

3. Performance effect of collective and independent I/O operations

Data access performance is affected by many factors, including caching, network bandwidth, and latency. Benchmarking with FLASH I/O also shows that the way resources are accessed can affect the execution time. In particular, we want to determine how independent and collective operations affect parallel I/O performance.

We will illustrate the effects of access and layout formats by using four test cases which consist of writing a selection to a rectangular array where the datatype of each element is a double precision float. In our analysis, we consider that the data is stored in row-major order. The tests were executed on NCAR Bluesky using HDF5 version 1.6.5.

We also included in our study the effects of using contiguous and chunked storage. When contiguous layout is used, a dataset is stored as a single contiguous portion of data on the media. In contrast, chunked layout partitions the storage space into several smaller units of contiguous storage that can be located anywhere on the media [1]. Chunked layout allows arrays to have extensible dimensions and enables data compression.

3.1 Collective and independent access using contiguous storage

Consider a rectangular array in which every processor has a noncontiguous selection involving a large number of rows and a small number of columns, as shown in Figure 3. Further note that the access requests of different processors are interleaved.

In order to write the entire selection using independent access, each process has to perform numerous small write operations, paying the high cost of latency many times. This is probably one of the worst cases for independent access in two dimensions.

In collective mode, however, the interleaved access requests of different tasks can be combined by each I/O agent into a single contiguous I/O operation yielding a very high speedup with respect to independent access [9].

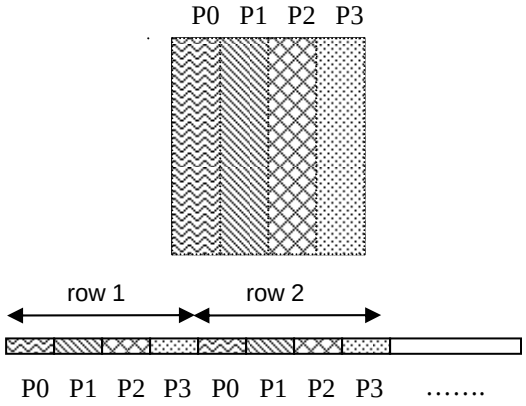

Figure 3: Geometrical layout and processor distribution on a noncontiguous interleaved selection

A write operation with 32 processors where each processor selection has 512K rows and 8 columns took 1,659.48 seconds using independent access and 4.33 seconds using collective access. In this instance, collective write is more than 380 times faster than independent write!

Now consider a rectangular array in which every processor has a contiguous selection as shown in Figure 4. In contrast to our previous case, this is a very good situation for independent access when the selection of each processor is larger than the stripe size, i.e. the buffer size of an I/O agent. It is also expected that collective access would perform well.

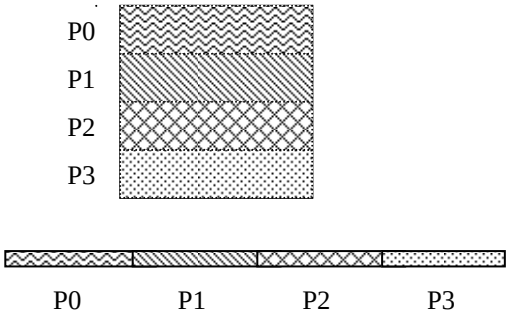

Figure 4: Geometrical layout and processor distribution on a contiguous selection

In this case, there are no interleaved requests from different processors and contiguity is already established in the access pattern. Also, since we assume that the selection of each processor is larger than the stripe size, only a single processor will request access to a specific stripe most of the time (improper alignment may cause two processors to access the same stripe very few times). Therefore, each I/O agent can perform a contiguous access and then ship the data to the requesting task. Collective access may not provide significant improvement in performance.

A write operation with 32 processors where each processor selection has 16K rows and 256 columns took 3.64 seconds using independent access and 3.87 seconds using collective access. Note that the selection size of each processor (32 MB) is larger than the stripe size in Bluesky (16MB). These execution times are nominally the same with a slight difference due to the overhead of collective access.

3.2. Collective and independent access using chunked storage

The application of chunked storage requires the definition of a uniform size for all chunks. A situation in which the chunk size does not divide exactly the dataset size on a particular dimension is not unlikely. This can yield unexpected results in I/O performance.

Consider a rectangular array whose storage has been partitioned into two chunks. Since the chunk size does not divide exactly the array size, one chunk extends beyond the storage space assigned to the array. This difference is represented by the letter Δ in Figure 5.

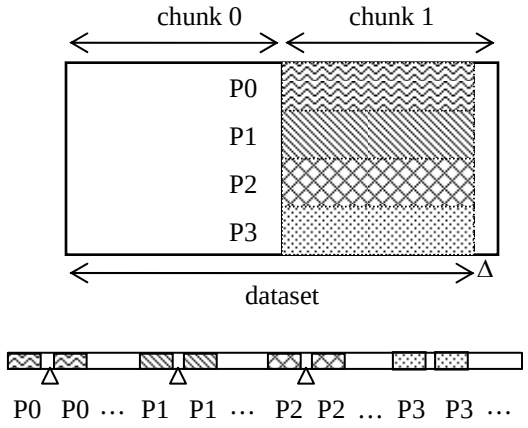

Figure 5: Geometrical layout and processor distribution on a noncontiguous selection within a chunk

A user selecting all the dataset space in chunk 1 may assume that the selection is contiguous. Unlike the case in Figure 4, the selection is noncontiguous because the extra chunk space generates holes in the selection. One way to write the entire selection would be to perform a write operation for each contiguous portion (each row in our example), incurring the high cost of latency several times. However, MPI-IO/GPFS can perform an efficient access by prefetching contiguous byte ranges [9] in an operation similar to data sieving [10]. For this optimization to perform most efficiently it is recommended that the application provide information about the access pattern in the form of derived datatypes.

A write operation with 32 processors where each processor selection has 16K rows and 256 columns took 43.00 seconds using independent access and 22.10 seconds using collective access. Since there are no interleaved requests and the selection of each processor is larger than the stripe size, the mentioned optimization would be expected to cause independent and collective access to achieve similar performance.

The main reason for the observed difference in performance is that the version 1.6.5 of HDF5 only provides datatypes to the MPI-IO layer when using collective operations. Passing derived datatypes for MPI-IO optimization of independent I/O operations will be implemented as an option in version 1.8 of HDF5.

In our final case, we have a rectangular array partitioned exactly into two chunks. Several processes participate within each chunk as shown in Figure 6.

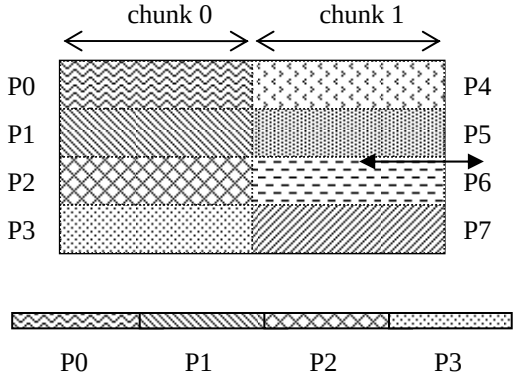

Figure 6: Geometrical layout and processor distribution on a contiguous selection within a chunk

Note that the every processor selection is contained within one chunk and contiguous. Therefore, independent and collective accesses have very similar performance with a small difference due to the overhead of collective operations as in Figure 4.

A write operation with 32 processors where each chunk is selected exactly by 4 processors, and each processor selection has 256K rows and 32 columns took 7.66 seconds using independent access and 8.68 seconds using collective access.

In a more general case, the selection of each processor may span several chunks causing the selection to be noncontiguous. Because HDF5 1.6.5 only provides derived datatypes to the MPI-IO layer when using collective access, it is likely that collective access will achieve higher performance than independent access. As mentioned before, independent operations with improved performance will be available in version 1.8 of HDF5.

4. Conclusions

We have reproduced the results of the earlier study comparing PnetCDF and HDF5 performance. But in contrast, our results indicate that the performance is quite comparable when the two libraries are used in similar manners. If collective access is employed in both, the performance difference is small. When independent mode is used with one library and collective mode is used with the other, performance differs in predictable ways.

It is noteworthy that HDF5 can achieve very high performance while providing superior flexibility for data management by using hierarchical file structures and chunked storage.

Our brief study of independent and collective operations shows that collective access should be preferred in most situations when using HDF5 1.6.5. Even in the cases where collective mode does not improve the performance, the resulting overhead is very small. In either case, it is important to provide information about the access pattern to the MPI-IO layer to allow for possible optimizations. Finally, we showed that the use of chunked storage may affect the contiguity of the data selection, which in turn may have an effect on the parallel I/O performance.

Acknowledgements

The authors would like to thank Frank Baker of the NCSA HDF group for his help in editing the paper.

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, the DOE SciDAC program under grant DEFC02-01ER25508, NCSA core (e.g. CIP): NSF grant SCI 0525308, 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.

References

1. HDF5 Home Page. The National Center for Supercomputing Applications. http://hdf.ncsa.uiuc.edu/HDF5/.
2. Unidata NetCDF Home page. http://www.unidata.ucar.edu/.
3. Gropp, W., Lusk, W., and Thakur, R. *Using MPI-2. Advanced Features of the Message-Passing Interface*, The MIT Press, Cambridge, MA, 1999.
4. Parallel netCDF Home Page. Argonne National Laboratory. http://www-unix.mcs.anl.gov/parallel-netcdf/.
5. Li, J., Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. "Parallel netCDF: A High-Performance Scientific I/O Package," *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, Phoenix, AZ, 2003.
6. *IBM General Parallel File System for AIX: Installation and Administration Guide*, IBM Document SA22-7278-03, July 2000.
7. Fryxell, B., Olson, K., Ricker, P., Timmes, F. X., Zingale, M., Lamb, D. Q., MacNeice, P., Rosner, R., and Tufo, H. "FLASH: An Adaptive Mesh Hydrodynamics Code for Modelling Astrophysical Thermonuclear Flashes," *Astrophysical Journal Suppliment*, pp. 131-273, 2000.
8. FLASH I/O Benchmark. http://flash.uchicago.edu/~jbgallag/io_bench/flash_io_bench.tar.gz.
9. Prost, J.-P., Treumann, R., Blackmore, R., Hartman, C., Hedges, R., Jia, B., Koniges,

A., and White, A. “Towards a High-Performance and Robust Implementation of MPI-IO on top of GPFS,” *Sixth International Euro-Par Conference*, Springer-Verlag, pp. 1253-1262, August-September 2000.

10. Thakur, R., Gropp, W., and Lusk, E. “Data Sieving and Collective I/O in ROMIO,” *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pp 182-189, February 1999.