Profiling and Improving I/O Performance of a Large-Scale Climate Scientific Application

Zhuo Liu[1] Bin Wang[1] Teng Wang[1] Yuan Tian[4] Cong Xu[1] Yandong Wang[1]
Weikuan Yu[1] Carlos A. Cruz[2] Shujia Zhou[2,5] Tom Clune[2] Scott Klasky[3]

Auburn University[1] Goddard Space Flight Center[2] Oak Ridge National Lab[3]
University of Tennessee Knoxville[4] Northrop Grumman Corporation[5]
{zhuoliu,bwang,tzw0019,congxu,wangyd,wkyu}@auburn.edu
{tiany,klasky}@ornl.gov
{carlos.a.cruz,shujia.zhou,thomas.l.clune}@nasa.gov

***Abstract*—Exascale computing systems are soon to emerge, which will pose great challenges on the huge gap between computing and I/O performance. Many large-scale scientific applications play an important role in our daily life. The huge amounts of data generated by such applications require highly parallel and efficient I/O management policies. In this paper, we adopt a mission-critical scientific application, GEOS-5, as a case to profile and analyze the communication and I/O issues that are preventing applications from fully utilizing the underlying parallel storage systems. Through in-detail architectural and experimental characterization, we observe that current legacy I/O schemes incur significant network communication overheads and are unable to fully parallelize the data access, thus degrading applications' I/O performance and scalability. To address these inefficiencies, we redesign its I/O framework along with a set of parallel I/O techniques to achieve high scalability and performance. Evaluation results on the NASA discover cluster show that our optimization of GEOS-5 with ADIOS has led to significant performance improvements compared to the original GEOS-5 implementation.**

I. Introduction

Scientific applications are playing a critical role in improving our daily life. They are designed to solve pressing scientific challenges, including designing new energy-efficient sources [2] and modeling the earth system [4]. To boost the productivity of scientific applications, High-Performance Computing (HPC) community has built many supercomputers [7] to provide unprecedented computation power over the past decade. Meanwhile, computer scientists are also arduously improving parallel file systems [11, 23] and I/O techniques [19, 20] to bridge the gap between fast processors and slow storage systems. However, despite the rapid evolution of HPC infrastructures, the development of scientific applications dramatically lags behind in leveraging the capabilities of the underlying systems, especially the superior I/O performance.

This paper seeks to examine and characterize the communication and I/O issues that prevent current scientific applications from fully exploiting the I/O bandwidth provided by underlying parallel file systems. Based on our detailed analysis, we propose a new framework for scientific applications to support a rich set of parallel I/O techniques. Among different applications, we select the Goddard Earth Observing System model, Version 5, (GEOS-5) from NASA [4] as a representative case. GEOS-5 is a large-scale scientific application designed for grand missions such as climate modeling, weather broadcasting and air-temperature simulation. Built on top of Earth System Modeling Framework (ESMF) [13] and MAPL library [25], GEOS-5 incorporates a system of models to conduct NASA's earth science research, such as observing Earth systems, and climate and weather prediction.

GEOS-5 contains various communication and I/O patterns observed in many applications for check-pointing and writing output. Data are organized in the form of either 2 or 3 dimensional variables. In many cases, multiple variables are arranged in the same group, called a bundle. A single variable is a composition of a number of 2-D planes, each of which is evenly partitioned among all the processes in the same application. Although the computation can be fully parallelized, our characterization identifies three inefficient communication and I/O patterns in the current design. First, for each plane of data, a process has to be elected as the *plane root* to gather all the data from all processes in the plane, thus causing a single point of contention. Second, only one process, called the *bundle root*, is responsible for collecting data from all the plane roots and writing the entire bundle to the storage system. This behavior essentially forces all the processes to wait until the bundle root finishes I/O, resulting in not only an I/O bottleneck but also a severe global synchronization barrier. Third, GEOS-5, like many legacy scientific applications, is unable to leverage state-of-the-art parallel I/O techniques due to rigid framework constraints, and continue using serial I/O interfaces, such as serial NetCDF (Network Common Data Form) [5].

To address the above inefficiencies, we redesign the communication and I/O framework in this GEOS-5 application, so that the new framework can allow application to exploit the performance advantages provided by a rich set of parallel I/O techniques. However, our experimental evaluation shows that simply using parallel I/O tools such as PnetCDF [16], cannot effectively enable application to scale to a large number of

processes due to metadata synchronization overhead. On the other hand, using another parallel I/O library, called ADIOS (The Adaptable IO System) [19], can improve the I/O performance with the trade-off that it may sacrifice the consistency induced by delayed inter-process written synchronization and complicate the post processing of output files.

To summarize, we have made following three research contributions in this work:

- We conduct a comprehensive analysis of a climate scientific application, GEOS-5, and identify several performance issues with GEOS-5 communication and I/O patterns.
- We design a new parallel framework in GEOS-5 for it to leverage a variety of parallel I/O techniques.
- We have employed PnetCDF and ADIOS for alternative I/O solutions for GEOS-5 and evaluated their performance. Our evaluation demonstrates that our optimization with ADIOS can significantly improve the I/O performance of GEOS-5.

The rest of this paper is organized as follows. Section II presents an overview of related work. Section III characterizes the communication and I/O patterns in existing scientific application, GEOS-5. Section IV introduces our new framework for scientific application to support a variety of parallel I/O techniques. Section V explores the evaluation results in detail. Section VI reviews our contribution and plans for future improvement.

II. Related Work

There is a large body of research literature on improving the I/O performance of scientific applications on large-scale supercomputing systems. Many I/O techniques have been designed to exploit the best I/O performance from underlying file systems. These include NetCDF-4 [30], HDF-5 [26, 20], PnetCDF [16] and ADIOS [1]. Built on top of these techniques, more efforts have been taken to study optimizations such as data buffering [17], file striping [34], subfiling [9, 10], staging [8] and data reorganization for multidimensional data structure [22, 24, 27, 28, 29]. Some of the techniques have been adopted into various I/O middleware libraries.

There is also a rich set of literatures on the performance characterization of HPC systems, spanning a wide variety of aspects such as inter-process communication [31], interconnect technologies, parallel file systems, reading patterns[18] and power management. Many studies are closely related to ours. [12, 14, 32, 33] reported scaling trends of the performance for various I/O patterns on Cray XT platforms. [15] described the challenges to improve the I/O performance and scalability on IBM Blue Gene/P systems. [21] characterized the I/O performance of NASA applications on Pleiades. Our work is different from the aforementioned studies. We focus on a climate application GEOS-5 that has a communication and I/O pattern representative of various climate and earth modeling applications. For this pattern, we introduce an extensible parallel framework that can employ different parallel I/O techniques such as PnetCDF and ADIOS and optimize the I/O performance of climate application.

III. Analysis of GEOS-5 Communication and I/O

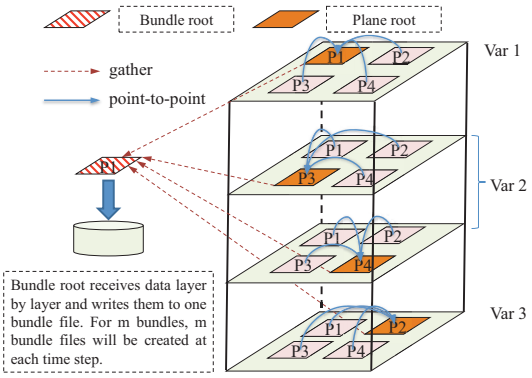


Fig. 1: Overview of GEOS-5 Communication and I/O

In this section, we first characterize the communication and I/O patterns in GEOS-5, and then examine their impacts on the application performance. The profiling results suggest that it is important to explore an alternative design for the data aggregation and storage for GEOS-5.

A. Current Data Aggregation and I/O in GEOS-5

GEOS-5 is developed by NASA to simulate climate changes over various temporal granularities, ranging from hours to multiple centuries. Like many legacy scientific applications, GEOS-5 adopts the serial version of NetCDF-4 I/O library [5] for managing its simulation data.

Data variables that describe climate systems are organized as *bundles*, Each bundle represents a physics model such as moisture and turbulence. It contains a varied mixture of many variables. Each variable has its data organized as a multidimensional dataset, e.g., a 3-D variable transposing internally into latitude, longitude, and elevation. To describe different aspects of the model, multiple 2-D or 3-D variables of state physics are defined, such as cloud condensates and precipitation.

GEOS-5 applies two-dimensional domain decomposition to all variables among parallel processes. 2-D variables have only one level of depth, naturally forming a 2D plane. 3-D variables are organized as multiple 2D planes. The number of 2-D planes is equal to the depth of a 3-D variable. As shown in Fig. 1, the bundle contains two 2-D variables - *var1* and *var3* and one 3-D variable - *var2*, thus forming a four-layer tube. Each 2-D plane of this bundle is equally divided among four processes so that all four processes can perform simulation in parallel.

At the end of a timestamp, these state variables are written to the underlying file system as history data for future analysis (the real production run lasts for tens or hundreds of timestamps). For maintaining the integrity of the model, all state variables that belong to the same model are written into the same file, called a *bundle file*. As mentioned earlier, each bundle file is stored using the *netcdf* format [5] popular for climatologists.

GEOS-5 currently adopts a hierarchical scheme for collecting data variables and writing them into a bundle file. As shown in Fig. 1, at the first step, each plane designates a different process as the *plane root* to gather the plane data from all the other processes. Upon the completion of gathering the planar data, one process called *bundle root* is elected to collect the aggregated data from the plane roots. When there are multiple bundles, several bundle roots will aggregate data in parallel from the 2-D planes that belong to their own bundle.

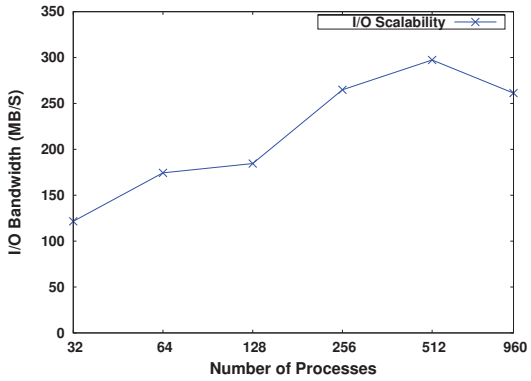


Fig. 2: I/O Scalability of Original GEOS-5

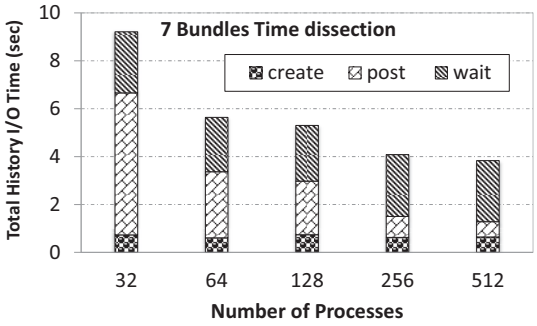


Fig. 3: Time Dissection of Non-blocking MPI Communication

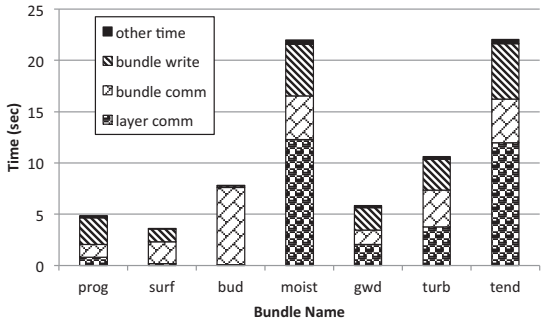


Fig. 4: Time Dissection for Collecting and Storing Bundles

B. Issues with the Existing Approach

While the existing implementation organizes and stores data variables as bundle files in a convenient format for climatologists, the approach described above faces a number of critical issues for scalable performance.

First, it lacks scalability. With the increase of the number of processes and planes, both plane roots and bundle roots can quickly become points of contention, resulting in communication bottleneck. As demonstrated in Fig. 2, the application stops scaling when the number of processes increases from 256 to 512 and 960. Although non-blocking MPI (Message Passing Interface) functions are designed to facilitate the overlapping of communication and computation, in current GEOS-5, larger number of processes leads to longer MPI_Wait time as shown in Fig. 3. Thus simply using non-blocking communication is unable to improve the scalability of the system. Second, increasing the data size of planes can overwhelm the memory capacity of root processes and saturate the network bandwidth of bundle roots, which can be detrimental to the system. Third, such approach leads to a global synchronization barrier among all the processes, since no process can proceed until the bundle root finishes storing all the plane data to the file system. Unfortunately, this point-to-point data transfer between bundle root and I/O server can be very time-consuming, leading to prolonged system running time.

C. Performance Dissection of Communication and I/O

To further quantify the communication and I/O time spent on storing history data, we dissect the entire process of collecting and writing 7 bundles in one timestamp. Fig. 4 shows the results of time dissection. *Moist* and *Tend* are the two largest bundles with 1201 and 1152 planes, respectively. Correspondingly, 55.9% and 54.3% of the time, respectively for Moist and Tend, are spent in collecting the plane data by plane roots. On the other hand, although bundle *Bud* has the fewest number of planes (40), 96.7% of its I/O time is spent on communication between bundle root and plane roots. This is because the plane roots for Bud also play roles as working processes for other bundles. This delays the progress of plane data collection for the bundle Bud. In addition, on average, the I/O time for writing the bundle into the file system only consumes about 27% of total time of writing the history data.

Gathering data consumes a significant portion of I/O time for the history data as shown in Fig. 4. Such implementation limits the scalability and is incapable of supporting large datasets. Many parallel I/O techniques are viable to address this issue; however, the hierarchical I/O scheme depicted in Fig. 1 is unable to leverage these techniques. Therefore, it is critical to overhaul the architecture of scientific application so that it can efficiently run on large-scale cluster with hundreds of thousands of processing cores.

IV. An Extensible Parallel I/O Framework

To overcome the limitations of existing design, especially to improve the performance at large scale, a new framework that can support parallel I/O is imperative. One alternative approach to enable parallel I/O is to have all the participating processes write their own output independently. However, such approach can generate a large number of small files, making it extremely difficult, if not impossible, for post-processing software, such as visualization tool, to analyze the

simulation results. In addition, many parallel file systems, such as GPFS, provide poor performance on managing small files due to high metadata overhead. Therefore, in this work, we take another approach to designing a generalized framework that can leverage a rich set of state-of-the-art parallel I/O techniques to eliminate the data gathering bottleneck in the original system, thereby accelerating the output of simulation data. Fig. 5 illustrates the new framework. Different from original design in which root processes need to gather either plane data or bundle data before writing the output into the file system, our new framework eliminates such limitation by enabling all the participating processes to write their own data into a shared file in the parallel file system.

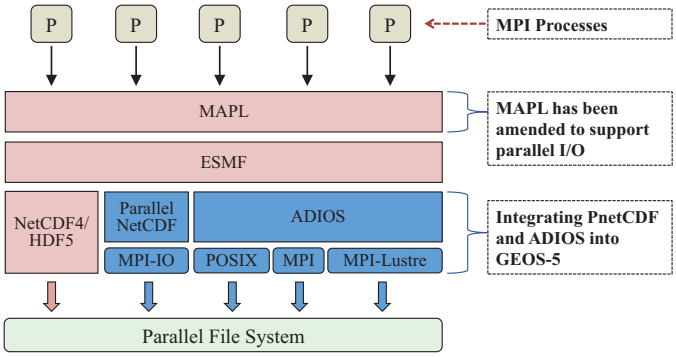


Fig. 5: Parallel I/O Based System Architecture

A. A Generalized and Portable Architecture

Different platforms have been optimized for different file formats. For example, many laboratories have meticulously tuned PVFS [6] to efficiently support NetCDF format. Therefore, to provide high portability across different platforms, we redesign the communication and I/O framework of GEOS-5 as shown in Fig 5 to support three parallel I/O techniques, including parallel NetCDF, NetCDF-4, and ADIOS. Despite the striking differences among these techniques, our new MAPL library and the ESMF framework together can hide such complexities and provide a set of uniform interfaces for upper layer MPI processes, who perform equivalently to conduct basic file operations, such as *open*, *write*, and *close*, *etc*. Within this design, each process can communicate with available I/O servers of file system via I/O middleware independently to transfer data. As a result, when data size is large, it can efficiently utilize the aggregated bandwidth to accelerate data transfers. Most importantly, such design eliminates the scalability bottleneck caused by plane roots and bundle roots described in section III. In the following sections, we provide a detailed description about how different scientific file formats are supported in the new framework.

B. Leveraging Parallel NetCDF

NetCDF is one of the most widely adopted formats used by many scientific applications. Its file structure contains a file header and multidimensional data arrays. File header includes metadata that describes the dimensions, attributes, and variable IDs. Data arrays store the concrete variable values, which can be either fixed-size or varied-size. Built on top of MPI-IO, parallel NetCDF (PnetCDF) [16] overcomes the inefficiency within the serial NetCDF [5] and provides parallel semantics to operate shared files in the file system.

Our new framework strives to leverage the strength of PnetCDF. Each MPI process performs computation on a chunk of plane data (as described in section III). At the end of each timestamp, instead of sending the data to the roots, each process directly writes the data into the shared NetCDF file through specifying the offset to the starting point of the data array. The starting point of the variable array is determined at the variable defining phase, which is conducted during the file open phase. During the data writing, we adopt collective I/O methods provided by PnetCDF in the current design. This means many I/O requests for non-contiguous portions of a file are merged together by either I/O servers or working processes before being processed. Such approach significantly reduces the number of I/O requests that need to be processed by the file system, thus improving the performance. At the file open and close phases, NetCDF metadata, such as dimension definition, variable attributes and lengths, needs to be synchronized among all of the participating processes in the group. To achieve this, one process is elected as the leader of the group to take charge of broadcasting the changes whenever metadata is modified. However, our evaluation results show that maintaining strong metadata consistency can cause high overhead when doing small bundle operations.

C. Integration of ADIOS

In order to leverage the asynchronous I/O of ADIOS to improve GEOS-5's performance, we integrate ADIOS into GEOS-5 as an ESMF component. ADIOS is able to provide the flexibility of switching back-end I/O methods at ease, such as the universal POSIX and MPI-IO methods. Meanwhile, ADIOS provides the easy-to-use plug-in mechanism to add new file system or platform-specific I/O methods for optimal performance, such as MPI-Lustre.

Along with the integration of ADIOS, we use the BP file format [19] for processes to write data to a shared file. BP is a self-describing and metadata rich file format that consists of a number of process groups (each process group maintains data for one individual process) and a footer containing the file's metadata and index information. It can be easily transformed into other scientific file formats, e.g., HDF5 and NetCDF.

In contrast to the current implementation that stores each bundle's data into a separate file at each time step, our ADIOS implementation dumps all bundle data into a single file at each time step, which greatly reduces the number of files created. At each time step, every process writes its portion of bundle data into a single shared file image. Instead of immediately writing a sub-plane at a time, the ADIOS method buffers all the variable data for different bundles in memory and asynchronously writes out the buffered data when buffer is full. Compared to the original NetCDF I/O, ADIOS can save file opening/closing overheads significantly and increase

the possibility of locally aggregating small data portions for reduced number of random disk accesses. In addition, the non-contiguous data layout of ADIOS format also contributes to the performance improvement. As no inter-process communication for aggregating data is needed in ADIOS I/O, the communication and synchronization overheads can be saved thus reducing the overall I/O time significantly.

V. Experimental Evaluation

In this section, we conduct a systematic evaluation of our parallel I/O based framework with PnetCDF and ADIOS, respectively. All experiments are performed five times and the average is presented in the paper. Our experiments are carried out on the Discover cluster [3] at the NASA Goddard Space Flight Center. Discover is one of the major computing platforms in NASA. It is equipped with 128 compute nodes, each of which contains two 6-core 2.8GB Intel Xeon CPUs and 24 GB memory. The compute nodes are connected with a 5PB GPFS parallel storage system via InfiniBand.

A. Performance with PnetCDF

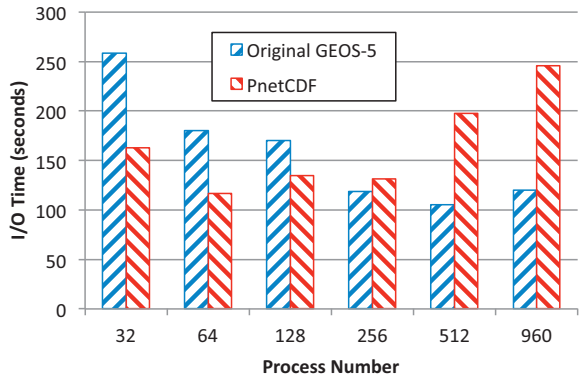
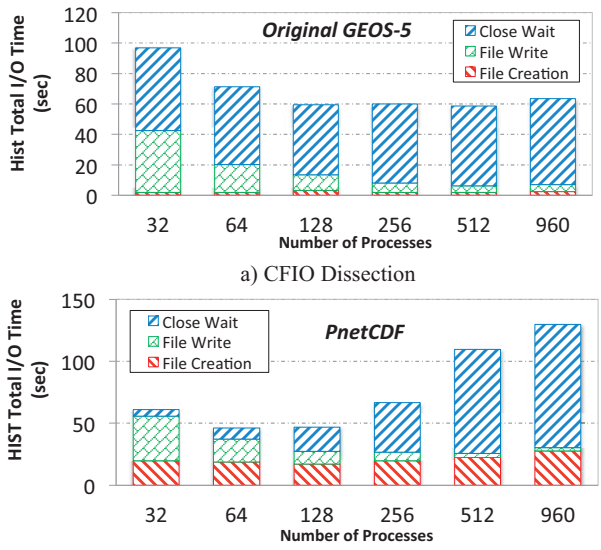


Fig. 6: Comparison between Serial NetCDF and PnetCDF



a) CFIO Dissection

b) PnetCDF Dissection

Fig. 7: Time dissection of CFIO and PnetCDF

We begin our evaluation through assessing the performance when PnetCDF is used. In this experiment, we measure the I/O time via using a single bundle Moist whose size is 1GB. Total number of timestamps is 10 so that the application outputs bundle results for 10 times. Meanwhile, we increase the number of processes from 32 to 960. Fig. 6 shows the evaluation results. However, we observe that compared to the original design called *CFIO*, although using PnetCDF can efficiently reduce the I/O time by up to 36.9% when the number of processes is less than 128, PnetCDF fails to provide improvement when the number of processes increases beyond 256, and unexpectedly degrade the performance by as much as 110% when the number reaches 960.

To investigate the cause, we dissect the I/O time. We measure the time consumed by *file creation*, *file write* and *close wait*, respectively. As shown in Fig. 7, within the given experimental configuration, the original design constantly incurs negligible *file creation* but high *close wait* overheads, while PnetCDF incurs much higher *file creation* and drastically increases *close wait* overheads for maintaining the consistency of metadata and aggregating data variables. In addition, the *file write* time of both the original design and PnetCDF I/O decreases with an increasing number of processes because of higher aggregated bandwidth. Though PnetCDF achieves several times higher write bandwidth than CFIO, its performance degrades as the total process count increases, because the total I/O time has been dominated by the overheads of *file creation* and *close wait* at large-scale for small bundles. For example, when the number of processes reaches 960, the two overheads cost almost 99% of total I/O time when PnetCDF is used. Note that PnetCDF incurs much higher *file create* and *close* overheads due to the metadata synchronization and aggregation of variable data so that the NetCDF format can be strictly consistent all the time. As a summary, these results explain why PnetCDF fails to outperform the original serial CFIO at scale. They also demonstrate that the mitigation of the overhead caused by *file creation* and *close wait* is critical to the successful adoption of any parallel I/O technique into scientific applications.

B. Performance with ADIOS

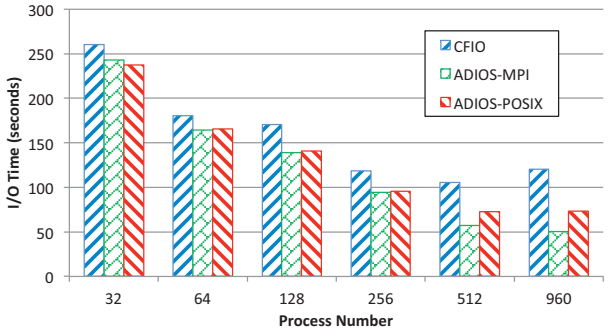


Fig. 8: Comparison between CFIO and ADIOS

To evaluate the performance of GEOS-5 with ADIOS, we select two ADIOS methods, MPI and POSIX, for writing

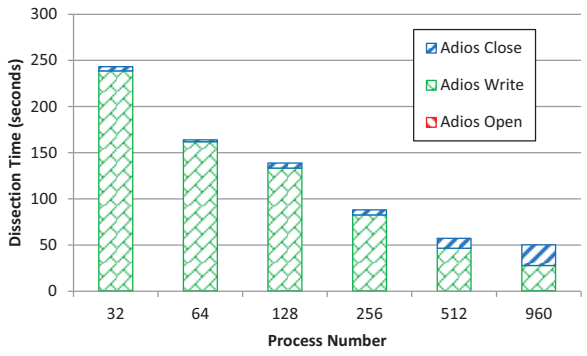


Fig. 9: Time Dissection of ADIOS-MPI

GEOS-5 bundle files. Fig. 8 and 9 show the results of evaluation, in which we use 7 bundles and 30 timestamps and increase the number of processes from 32 to 960. As shown in Fig. 8, compared to the original design, using ADIOS with MPI significantly reduces the I/O time, up to 58.2% when the number of processes is 960. More importantly, ADIOS with MPI shows efficient scalability for a large number of processes (from 256 to 960). On the contrary, although ADIOS with POSIX shows effective improvement over the original design in terms of I/O time reduction, it stop scaling when the number of process increases from 512 to 960 due to consistency semantics rooted in POSIX interfaces.

Fig. 9 shows the time dissection of using ADIOS-MPI. Similar to the approach used in section V-A, we analyze the time spent on *open*, *write*, and *close*, respectively. As shown in the figure, ADIOS incurs almost zero overhead when conducting file open, and negligible overhead for file close. This is because the file open call returns immediately with the use of open-on-write. And when writing out the data, buffered data is frequently flushed to the underlying file system, thus causing little waiting time when file is being closed. In addition, ADIOS with MPI shows scalable file writing with an increasing number of processes, decreasing from 90% of I/O time to 56% of I/O time. This is because of the higher aggregated bandwidth achieved with a large number of processes. Compared with PnetCDF, using ADIOS efficiently mitigates the overhead of maintaining the metadata of shared file, thus achieving better scalability and performance at large-scale.

VI. Conclusions

In this work, we target at identifying and addressing influential factors that can hinder current scientific applications from achieving efficient I/O. By adopting the GEOS-5 climate modeling application as our case study, we analyze the typical communication and I/O patterns in representative scientific applications. And we discover that many legacy scientific applications employ similar hierarchical network aggregation to collect data portions for multi-dimensional variables from all processes and then dump the aggregated data into persistent storage. Through comprehensive measurements on the NASA discover cluster, we quantitatively profile and dissect the network communication and I/O costs by such I/O schemes. In order to address the drawbacks of single point of network contention and under-parallelized I/O patterns, we modify the current I/O framework of GEOS-5, enabling the applications to take advantage of state-of-art parallel I/O techniques like PnetCDF and ADIOS. Experimental results demonstrate that the integrated parallel I/O techniques supported by our I/O framework can improve the application I/O time at various scales. This performance improvement come from the elimination of huge amounts of network aggregation and significantly promoted I/O concurrency.

In the future, we plan to study asynchronous I/O schemes and efficient burst buffering techniques for further I/O performance improvements of scientific applications.

Acknowledgments

This work is funded in part by a NASA grant NNX11AR20G and enabled by the U.S. National Science Foundation award CNS-1059376 to Auburn University.

References

[1] Adaptable I/O System. http://www.nccs.gov/user-support/center-projects/adios .
[2] Energy Sources. http://energy.gov/science-innovation/energy-sources.
[3] NASA Discover. http://www.nccs.nasa.gov/cluster_main.html.
[4] NASA GEOS. http://gmao.gsfc.nasa.gov/systems/geos5.
[5] NETCDF. http://www.unidata.ucar.edu/software/netcdf.
[6] The Parallel Virtual File System, version 2. http://www.pvfs.org/pvfs2.
[7] TOP 500 Supercomputers. http://www.top500.org.
[8] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 39–48, New York, NY, USA, 2009. ACM.
[9] J. Bent, G. A. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate. PLFS: a checkpoint filesystem for parallel applications. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.
[10] A. Choudhary, W. Liao, K. Gao, A. Nisar, R. Ross, R. Thakur, and R. Latham. Scalable I/O and analytics. *Journal of Physics*, 180(1), 2009.
[11] Cluster File System, Inc. Lustre: A Scalable, High Performance File System. http://www.lustre.org/docs.html.
[12] M. R. Fahey, J. M. Larkin, and J. Adams. I/O performance on a massively parallel cray XT3/XT4. In *Proc. 22nd IEEE International Symposium on Parallel and Distributed Processing (22nd IPDPS'08)*, 2008.
[13] C. Hill, C. DeLuca, V. Balaji, M. Suarez, and A. D. Silva. The architecture of the earth system modeling

framework. *Computing in Science and Engg*., 6(1):18–28, January 2004.

[14] J. H. Laros III, L. Ward, R. Klundt, S. Kelly, J. L. Tomkins, and B. R. Kellogg. Red storm IO performance analysis. In *CLUSTER*, 2007.

[15] S. Lang, P. H. Carns, R. Latham, R. B. Ross, K. Harms, and W. E. Allcock. I/O performance challenges at leadership scale. In *SC'09 USB Key*. ACM/IEEE, Portland, OR, USA, November 2009.

[16] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, and R. Latham. Parallel netCDF: A High Performance Scientific I/O Interface. In *Proc. SC03*, 2003.

[17] Z. Liu, B. Wang, P. Carpenter, D. Li, J. S. Vetter, and W. Yu. PCM-based durable write cache for fast disk I/O. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 451–458. IEEE, 2012.

[18] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: Reading patterns for extreme scale science io. In *In Proceedings of High Performance and Distributed Computing*, 2011.

[19] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich IO methods for portable high performance IO. In *IPDPS'09*, 2009.

[20] The HDF Group. Hierarchical data format version 5, 2000–2010. http://www.hdfgroup.org/HDF5.

[21] S. Saini, J. Rappleye, J. Chang, D.P. Barker, R. Biswas, and P. Mehrotra. I/O Performance Characterization of Lustre and NASA Applications on Pleiades. In *HiPC*, 2012.

[22] S. W. Schlosser, J. Schindler, S. Papadomanolakis, M. Shao, A. Ailamaki, C. Faloutsos, and G. R. Ganger. On Multidimensional Data and Modern Disks. In *FAST*, 2005.

[23] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *FAST '02*, pages 231–244. USENIX, January 2002.

[24] T. Shimada, T. Tsuji, and K. Higuchi. A storage scheme for multidimensional data alleviating dimension dependency. In *Digital Information Management, 2008. ICDIM 2008. Third International Conference on*, 2008.

[25] M. Suarez, A. Trayanov, C. Hill, P. Schopf, and Y. Vikhliaev. MAPL: a high-level programming paradigm to support more rapid and robust encoding of hierarchical trees of interacting high-performance components. In *Proceedings of the 2007 symposium on Component and framework technology in high-performance and scientific computing*, 2007.

[26] The National Center for SuperComputing. HDF Home Page. http://hdf.ncsa.uiuc.com/hdf4.html.

[27] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, N. Podhorszki R. Grout, Q. Liu, Y. Wang, and W. Yu. EDO: Improving Read Performance for Scientific Applications Through Elastic Data Organization. In *CLUSTER '11: Proceedings of the 2011 IEEE International Conference on Cluster Computing*, 2011.

[28] Y. Tian, S. Klasky, W. Yu, H. Abbasi, B. Wang, N. Podhorszki, R.W. Grout, and M. Wolf. SMART-IO: SysteM-AwaRe Two-Level Data Organization for Efficient Scientific Analytics. In *MASCOTS*, 2012.

[29] Y. Tian, Z. Liu, S. Klasky, B. Wang, H. Abbasi, S. Zhou, N. Podhorszki, T. Clune, J. Logan, and W. Yu. A lightweight I/O scheme to facilitate spatial and temporal queries of scientific data analytics. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*. IEEE, 2013.

[30] Unidata. http://www.hdfgroup.org/projects/netcdf-4/.

[31] C. Xu, M. G. Venkata, R. L. Graham, Y. Wang, Z. Liu, and W. Yu. SLOAVx: Scalable LOgarithmic AlltoallV algorithm for hierarchical multicore systems. In *Cluster, Cloud and Grid Computing (CCGrid), 13th IEEE/ACM International Symposium on*. IEEE, 2013.

[32] W. Yu, H. S. Oral, J. S. Vetter, and R. Barrett. Efficiency Evaluation of Cray XT Parallel I/O Stack. In *Cray User Group Meeting (CUG 2007)*, 2007.

[33] W. Yu, J. S. Vetter, and S. Oral. Performance characterization and optimization of parallel I/O on the Cray XT. In *IPDPS*, 2008.

[34] W. Yu, J.S. Vetter, R.S. Canon, and S. Jiang. Exploiting lustre file joining for effective collective IO. In *CCGRID*, 2007.