Adaptable, Metadata Rich IO Methods for Portable High Performance IO

Jay Lofstead
College of Computing
Georgia Institute of Technology
Atlanta, Georgia
lofstead@cc.gatech.edu

Fang Zheng
College of Computing
Georgia Institute of Technology
Atlanta, Georgia
fzheng8@mail.gatech.edu

Scott Klasky
Oak Ridge National Laboratory
Oak Ridge, Tennessee
klasky@ornl.gov

Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, Georgia
schwan@cc.gatech.edu

Abstract*—Since IO performance on HPC machines strongly depends on machine characteristics and configuration, it is important to carefully tune IO libraries and make good use of appropriate library APIs. For instance, on current petascale machines, independent IO tends to outperform collective IO, in part due to bottlenecks at the metadata server. The problem is exacerbated by scaling issues, since each IO library scales differently on each machine, and typically, operates efficiently to different levels of scaling on different machines. With scientific codes being run on a variety of HPC resources, efficient code execution requires us to address three important issues: (1) end users should be able to select the most efficient IO methods for their codes, with minimal effort in terms of code updates or alterations; (2) such performance-driven choices should not prevent data from being stored in the desired file formats, since those are crucial for later data analysis; and (3) it is important to have efficient ways of identifying and selecting certain data for analysis, to help end users cope with the flood of data produced by high end codes. This paper employs ADIOS, the ADaptable IO System, as an IO API to address (1)-(3) above. Concerning (1), ADIOS makes it possible to independently select the IO methods being used by each grouping of data in an application, so that end users can use those IO methods that exhibit best performance based on both IO patterns and the underlying hardware. In this paper, we also use this facility of ADIOS to experimentally evaluate on petascale machines alternative methods for high performance IO. Specific examples studied include methods that use strong file consistency vs. delayed parallel data consistency, as that provided by MPI-IO or POSIX IO. Concerning (2), to avoid linking IO methods to specific file formats and attain high IO performance, ADIOS introduces an efficient intermediate file format, termed BP, which can be converted, at small cost, to the standard file formats used by analysis tools, such as NetCDF and HDF-5. Concerning (3), associated with BP are efficient methods for data characterization, which compute attributes that can be used to identify data sets without having to inspect or analyze the entire data contents of large files.

I. Introduction

File formats like HDF and NetCDF are popular in part due to the rich tool chains available for the scientific data stored using these formats. Both HDF and NetCDF, however, were initially designed for serial access, limiting scalability when used in massively parallel codes. In response, the broader community has developed parallel versions of their APIs [1], [2], with good results demonstrated in the terascale environment compared to their serial counterparts. However, there remain serious scalability issues for petascale machines and beyond, a simple example explained in this paper being the inability of HDF-5 to scale to 8192 processes for benchmarks conducted with the Chimera supernova code. Here, with every performance option enabled for parallel HDF-5 enabled, we measure 1400 seconds to write a 7 GB restart file to the Lustre system on the Jaguar machine at Oak Ridge National Laboratories (ORNL), whereas the use of our ADaptable IO System (ADIOS) and its IO API with underlying POSIX IO reduces that time to 1.4 seconds! ADIOS with MPI-IO and collective MPI-IO yields performance of 10 and 14 seconds, respectively. BP conversion to HDF-5 performed serially on a single login node on Jaguar requires 117 seconds.

IO performance depends on many factors, including the file format used, the implementation and tuning of the associated API, the file system employed, and the architecture of the HPC resource being used for production runs. Our analysis of the parallel HDF-5 implementation on Jaguar, for instance, reveals a large number of MPI_Bcast calls, which are used to guarantee that all processes writing the collective values are writing consistent data to the file and that all IO processes maintain a coordinated march through the data elements for each collective output. We conclude from facts like these that ADIOS should provide end users easy access to multiple IO methods. On Jaguar, for instance, given the severe performance impact of validating the parallel consistency of data output, end users might use HDF-5 and Lustre during testing, but then disable consistency checking during large-scale production runs. Unfortunately, the parallel HDF-5 and parallel NetCDF APIs do not offer this capability, since both the HDF-5 and NetCDF file formats are designed to provide a single view of the entire data in the file. The laudable idea is that file contents should not change with say, 30,000 processes all writing the same HDF-5 file in parallel vs. that file being written serially by a single process. The unfortunate outcome, however, is that the API then forces the user to validate file consistency each time the file is written, no matter if it is serial or parallel output. In response, ADIOS introduces the notion of an intermediate file format that does not require immediate file consistency, termed BP. By using BP, we avoid the runtime costs of consistency checking, but can still obtain the data in a format that integrates with the tools currently used in the science work flow. This is done after the simulation has written the data, by using a converter to validate consistency and create the desired HDF-5 or NetCDF formats.

We term ADIOS an adaptable IO system because using it makes it easy for end users to use the IO methods that offer the levels of support they require. In the Chimera/Jaguar example, for instance, it is appropriate to use parallel HDF-5 during debugging and/or in small runs to validate that the output is being generated correctly. Once the code has been validated, other methods can be used. With ADIOS, such changes do not necessitate changes to science codes but only involve changing a single entry in an external XML configuration file. For the Chimera/Jaguar production runs, the outcome is an up to 3 orders of magnitude improvement in IO performance. Further, with its use of the intermediate BP file format, ADIOS does not prescribe whether or when the output is converted to HDF-5 or NetCDF formats. Such conversions can be done (i) 'in line' as part of the IO operation through an IO Graph [3], to ensure that only one format of the data will ever be stored on disk, (ii) offline using constructs termed metabots [4] that inspect disk-resident data and perform conversions whenever possible and outside the IO fast path, (iii) as part of a larger, more complex workflow using Kepler [5], Pegasus [6], or DAGMan [7], or (iv) via a stand-alone file format converter. In all of these cases, such conversions can be done efficiently, with our initial results reported with a standalone converter running on a single processor on a login node of the ORNL Jaguar machine resulting in a 117 second conversion time for a 7 GB output file (i.e., in contrast to the total 1400 sec. IO time for Chimera quoted above for parallel HDF-5).

ADIOS and its BP intermediate file format not only support the flexible conversion to standard file formats, but they also facilitate the summary inspection of the data to determine if it contains features of interest to end users. One way to provide such functionality is to fully index the data, as done by multiple projects that have developed content indices for HDF-5 files [8], [9], [10], [11]. To achieve a similar goal, but with less overhead in space and time, ADIOS supports the notion of *data characteristics* using which one can collect local, simple statistical and/or analytical data during the output operation (or later) for use in identifying desired data sets. Simple characteristics like local process array minimum and maximum values can be collected nearly 'for free' as part of the IO operation. More complex analytical measures like standard deviations or specialized measures particular to the science being performed may require processing that can be done in a variety of ways, including before or after the data has been written to disk. In all such cases, the BP format offers efficient, compact ways of storing these characteristics. When converting BP files to say, HDF-5 or NetCDF, attributes can be used to maintain them.

The BP file format's implementation uses a footer index, to avoid the known limitation of header-based formats like NetCDF [12] where any change to the length of file data will require moving it. Further, by placing version identifier and an offset to the beginning of the index as the last few bytes of a BP file, it becomes trivial to find the index information and to add new and different data to the file without affecting any data already written. Finally, we incorporate data characteristics into the index, so that we can separate the index for use as a table of contents for the file on a tape storage system like HPSS [13].

To summarize, the ADIOS IO system and its BP file format are designed to help attain scalable, high performance IO while at the same time, maintaining compatibility with the rich tool chains existing for standard file formats like HDF-5 and NetCDF. By using either parallel HDF-5 or parallel NetCDF for initial code development and testing, the internal file consistency and 'correctness' of data output can be ensured. By switching to the BP format and using POSIX, MPI-IO, or collective MPI-IO methods for large-scale production runs and employing a converter, the IO time experienced by petascale codes can be reduced by up to three orders of magnitude, while still obtaining files with identical format and contents as when directly using native parallel APIs. The key contributions of ADIOS are as follows:

1) runtime selection of IO methods to achieve high performance for different platforms and IO patterns;
2) an intermediate file format designed for high-performance parallel IO and for maintaining compatibility for the file formats necessary for analysis workflows; and
3) the use of data characteristics and indexing for rapid data identification and retrieval to enhance scientist productivity.

The remainder of this paper describes related work, the architecture of ADIOS and some details of the BP file format, our evaluation environment and codes and experimental measurements, and ends with conclusions and a discussion of future work.

II. Related Work

There has been substantial work to loosen some of the strict semantics of serial IO systems to enhance performance for parallel IO. For example, NFS relies on a write-back from a local cache before changes may be visible on another machine in the network [14]. The Lightweight File Systems project at Sandia Labs [15] has sought to strip down the file system to only its security and access components at the core and then allows the optional layering of other semantics on an as needed basis, in part to avoid known issues like the fact that the single metadata server in Lustre [16] can cause significant bottlenecks on metadata operations. Other file systems [17],

[18], [19], [20] have successfully addressed this issue, but none have yet addressed the 'internal' file issues raised and solved by ADIOS, such as selective file consistency.

The file formats used by science applications range from the most popular HDF-5 and NetCDF to more niche players like SAF [21], PDS [22], GRIB [23], and HDS [24]. Each of these has been optimized for a particular style of data arrangement and annotation. All of these formats share a requirement to use the same consistency validation during output as what is required by the on-disk file, necessitating the use of global consistency checks as part of the IO process (e.g., by using collectives). The use of an intermediate format like BP in ADIOS addresses this 'internal' file issue, making it possible to avoid and/or delay consistency validation to improve IO performance.

File conversion is commonly used in computer systems. Simulations frequently use converters as part of code coupling operations to 'fix-up' not just the data format on disk, but also to do perform actions like changing units, data filtering, or others. For NetCDF files, NCO [25], the netCDF Operators, provides a way to extract values from one or more files into a new file for more convenient use (e.g., used in the climate community). To provide resilience against file corruption and avoid the performance penalty of resizing the file header when writing each new history output, every output set is written to a new file. NCO is used to construct the view of a single variable over time by pulling the appropriate pieces from the set of NetCDF files into a single, new NetCDF file. The combustion simulation S3D writes each process's output into a separate file for performance and then uses another, independent process to combine all of these outputs into a single new file. This is similar to a workflow related approach using tools like Kepler, Pegasus, or DAGMan. We have been investigating alternative approaches. For example, through the use of an IO Graph, we have demonstrated the ability to write data in a different format and/or with different filtering and/or processing without the intermediate data ever hitting disk [3]. We have also been investigating the use of near-line converters called metabots that monitor the file system for new files and automatically apply their operation to generate the new output. By evaluating the work required to perform the data manipulation, placement of the operation either inline as an IO Graph or near-line as a metabot can be chosen. We have also frequently used components to convert our data output into images or just strip out portions relevant to other downstream processing [26] by using actors in Kepler workflows [27].

Detailed data indexing has been explored in four HDF-5 indexing prototypes. The use of bitmap indices in [8] and [9] provides an efficient way to find values typically within in a given range. The use of projection tables [10] takes the approach of listing values and giving the location(s) at which it appears. PyTables [11] uses Python to provide more traditional database-like indexing of data files. All of these have focused on providing full or a sampling approach data indexing. Our approach is not to give direct access to all data elements but rather to aid in the identification of which data sets are relevant for the desired use. For example, to know which 10 TB file contains the data where the temperature exceeds 10^6, looking at the maximum values for the temperature is sufficient. Simple metrics like these can always be collected and will always be available in ADIOS. Additional information must be computed in analysis workflows.

Previously published ADIOS work includes a workshop paper describing excellent initial performance results using ADIOS [28] including an overview of the programming API and XML format. A 2008 Cray User's Group paper [29] elaborated on the workshop paper. This paper extends this work by re-evaluating the performance after the development of a newer, very different BP file format and discusses the performance numbers in this newer context.

III. ADIOS Use and Software Architecture

ADIOS [28], the ADaptable IO System, provides an API nearly as simple as POSIX IO and an external XML configuration file that is parsed once at startup, the latter providing flexibility in how IO is performed and in what IO occurs. Metadata is either stored in the XML file, to avoid pollution of the source code, or as is the case with data characteristics, it is generated either during or after data storage, and stored directly in the intermediate output files. ADIOS was developed in response to our experiences with the IO performed by full science codes that include GTC (fusion), GTS (fusion), XGC1 (fusion), Chimera (supernova), S3D (combustion), and the Flash IO benchmark (astrophysics). The key features of ADIOS are the following:

1) *external XML file* – for IO description and configuration;
2) *single API for all IO methods* – no matter how IO is actually performed;
3) *runtime selection of potentially different IO methods* – per grouping of data;
4) *BP file format* – designed for minimal required coordination, compact metadata storage, and resilience in cases of failures; and
5) *an initial implementation focus on write performance* .

A. External XML file

The complexity of IO methods is often directly reflected in that of the source code API required for their use. To avoid exposing such complexity to source codes, the ADIOS XML format describes the structure of output data, any attributes attached to items or groups, and the selection of the particular IO method to employ for this run of the science code. Through the use of this XML file, ADIOS controls what data is written and which method is used for each grouping of data in the code. Since this introduces a consistency requirement between the XML file and the source code, we have developed a method for automatically generating nearly all of the code's IO commands based on the XML file. This eliminates the need to maintain a set of calls in the source code and a set of data descriptions in the external XML file. Using this feature reduces the API calls in the science code to just an 'open', 'close', and an include of the generated write or read statements. By properly setting dependencies in the project Makefile, it is easy to automatically generate new include files and properly recompile dependent source files when the XML file changes. This has shown to be effective for nearly all of the IO examples we have encountered in the petascale codes targeted by our work.

B. Single API for all IO Methods

Since all descriptive elements of the output operation have been moved to the XML file, the ADIOS API is nearly as simple as POSIX IO, and in many cases, even simpler. For example, even to have a richly annotated HDF-5 file with many attributes and with hierarchical structure, the user need only have an 'open', '#include', and 'close' statement in the science code. The '#include' will insert the generated ADIOS IO calls based on the XML file for use in the science code. For more complex cases where very fine manipulation of the IO operations is required, it is still possible to have an 'open' followed by a series of 'read' or 'write' statements, and finally, a 'close' statement. The only additional requirement in this case is our 'group_size' call immediately after the 'open' call. This provides a way for the underlying API to decide if it should buffer the output for optimal performance, coordinate with other processes participating in the output to determine the local offset in the global file for output, and to initialize any output parameters not set during the 'open' call.

C. Runtime Selection of Potentially Different IO Methods

Our hope is that ADIOS can be used to attain high IO performance no matter on which platform a code is deployed. To achieve this, it is necessary to be able to select which IO method to employ for a science code. This is particularly important with limited or costly time allocations on petascale machines, where excessive IO times can substantially reduce scientific productivity. The need for selectivity also extends to individual IO groupings within the code, where for instance, since diagnostic output is likely small but frequently written, it makes sense to write it using one approach (e.g., HDF-5), while restarts that are large and infrequent may need to use a different approach (e.g., MPI-IO) to gain the performance benefit of not performing the consistency checks during the large-scale run.

D. The Intermediate BP File Format

ADIOS uses the BP format as a default since its design is central to our ability to achieve high performance, compatibility with standard formats, and efficient data annotation and characterization. The basic layout of BP is shown in figure 1. Briefly, each process writes to its own area independently. Each of these segments of the file are termed a 'process group'. The contents of each of these process groups is indexed and stored at the end of the file for rapid data selection and identification. The format is discussed in detail in [30].

This section briefly describes the key features of BP, based on five goals for a high performance file format, termed RICCI:

1) *Resilient* in the presence of a variety of failures;
2) *Independent*, parallel IO with sufficient annotation to validate and enforce consistency later;
3) *Convertible* to both HDF-5 and NetCDF;
4) *Characterize* data for easier data analysis and selection; and
5) *Index* the metadata and characterize all of the data for fast, direct access.

1) Resilient: To avoid the loss of previously written data during a later IO operation to the same file(s), BP incorporates three key features. First, there is a local copy of all relevant metadata for each process that writes output. Thus, BP does not rely on a centralized header area for access to local data elements. Second, since each process writes its output into the file independently, its failure does not affect other processes and/or the ability to read other file portions. Third, the BP file index contains replicated metadata to deal with data failures. By storing where each process group resides in the file in the index, the BP format affords proper identification of undamaged sections of the file by indicating where to start parsing. This also holds true for variables as we replicate the location of each in the index as well.

2) Independent: As mentioned in the previous section, BP stores each process's output independently. This achieves two advantages. First, by storing the full output of each process independently with full annotation, the BP format can delay consistency checking outside the IO fast path. This can be done as a stand-alone operation or as part of converting to other formats. Second, the lack of coordination among the processes for each piece of the output eliminates any intermediate synchronization points during the IO operation. The BP format only requires one coordination operation at the start to decide on file offsets to write in parallel and once at the end to collect index information to process 0 to append on the end of the file. This general lack of coordination during the writing process affords each process writing in larger blocks with a slow storage node only affecting the output once at the end by delaying the ultimate completion of the output of the index. The total IO time becomes the longest time overall for any individual process rather than the sum of the longest time for each output element.

3) Convertible: Given our own investment in the use of tools requiring HDF-5 and/or NetCDF, BP provides all of the features we have encountered 'in the wild' by users of HDF-5 and NetCDF. There are esoteric features of these standards we have chosen to not address in the first release of ADIOS because we have not encountered a science code that required the feature. As the codes adopting ADIOS increase, the BP format will evolve to meet these additional needs.

4) Characterized: With simulation sizes growing as the machines grow, data is growing commensurately. File sizes today already are in the 10s of GB or larger with multi-TB file sizes becoming more common as use of petascale machines increases. ADIOS characterizes the data as it is written to aid later selection of file(s) for processing. For example, when trying to figure out which set of data output during a materials

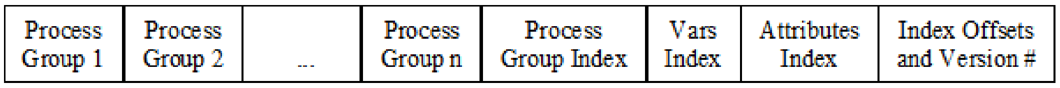

Fig. 1. BP File Layout

simulation run is the one containing the interesting feature being investigated, selecting the file where the number of material pieces grows by more than 10% from the previous time step would be sufficient. We collect these characteristics and store them both with the process output and in the index to preserve resilience and to aid in rapid selection and access. While this information is not directly convertible to HDF-5 or NetCDF except as attributes, we feel automatically collecting this information increases the value of the output and the ease of selecting the proper potentially multi-TB file to process with analysis tools.

5) Indexed: The BP format is a collection of independent process outputs rather than a single organization of data that originated from 1 or more processes. This is not to say that there is not a universal view of the file contents available. Through the index, an HDF-5 or NetCDF-style header output can be constructed without having to parse each of the process outputs. This affords the advantage of highly parallel, independent IO while maintaining a global view of the file contents. The placement of the index was driven by performance and resilience requirements. If the index were at the beginning of the file, if it expands too much, moving data would be required. This performance impact along with the potential for file corruption when moving data dictated that the index be placed at the end of the file. This has the added benefit of making it easier to make the process group outputs into file system stripe sized ‘chunks’ for optimal write performance since it is not necessary to waste potentially 4 MB of space at the front of the file to store the index information just to keep stripe-sized ‘chunks’ per process. Since the BP index is small, on the order of 100 bytes per process output and 100 bytes per unique variable or attribute written and generally at most another 50 bytes per instance in the file, filling a stripe width with the header information is difficult. The index itself stores for each process group the process ID, the ADIOS group name written, the offset from the beginning of the file, and the time-index value for this output, if any. The variable and attributes are each indexed separately, but contain essentially the same information. Each index stores a unique set of variables or attributes across all process groups written with a list of characteristics for each. These characteristics include the offset from the beginning for the file for each place it is written in the file, the array dimensions and minimum and maximum values if it is an array, and the scalar value if it is a simple value. This provides direct information of where each portion of any array is written for any timestep as well as characterizing the data for direct evaluation of whether or not the data is likely what the user is interested in analyzing. These indices can easily be separated from the BP file as a separate file for use on a tape storage system or portable file to identify the full contents of the data file it represents. For rapid retrieval, the last few bytes of the file store a version identifier and the offset at which the index is stored.

E. *Initial focus on write performance*

For the classes of science codes we evaluated in the development of ADIOS, write performance was far more important that read performance. Keeping that in mind, we created ADIOS with the ability to read successfully and with general good performance, but we have not spent any effort in highly tuning the read performance like we have with the write performance. For example, the delayed consistency model relies heavily on the index to achieve good read performance when performing an MxN read. Our main focus in this area has been in the converters to HDF-5 and NetCDF. The next release of ADIOS will focus on providing excellent read performance while addressing a broader array of science codes.

IV. Evaluation

To demonstrate the utility of ADIOS, two different, full scientific codes are used. For the Chimera supernova code, performance differences and scaling factors are shown for various run sizes. For the GTC fusion code, the relative performance of using different IO methods within ADIOS is evaluated. Full science simulations rather than the common IO benchmarks are used to give the best possible representation of how this approach works in a real, production environment.

Evaluations are performed on two different machines. For all of the Chimera-related tests and for the GTC weak scaling tests, the Jaguar Cray XT4 system at Oak Ridge National Laboratory is used. It consists of 7832 compute nodes plus additional login and IO nodes. Each compute node contains a quad core AMD 2.1 GHz Opteron with 8 GB of memory. The login and IO nodes consist of dual core 2.6 GHz Opterons with 8 GB of memory. The system is running Compute Node Linux. Various numbers of compute nodes are used to write to the 600 TB Lustre scratch system. The conversion tests are simply run on a single Jaguar login node against the same Lustre scratch system. The GTC strong scaling performance tests are run on ORNL’s Ewok end-to-end cluster. It is an Infiniband-based Linux cluster consisting of 81 dual core 3.4 GHz Pentium IVs with 6 GB of memory. It is configured with a 20 TB Lustre scratch space used in these tests.

A. *Chimera Evaluation*

The Chimera evaluation has three parts. The first examines the relative performance of the Chimera code with parallel HDF-5 compared to various ADIOS-based IO methods. The second evaluates parallel HDF-5 performance using independent MPI-IO and compares it against ADIOS using independent MPI-IO. The third uses a sample BP file generated by

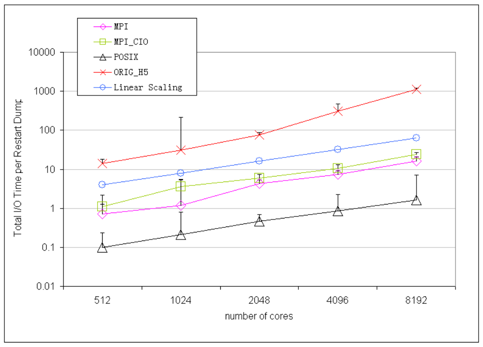

Fig. 2. Chimera Weak Scaling

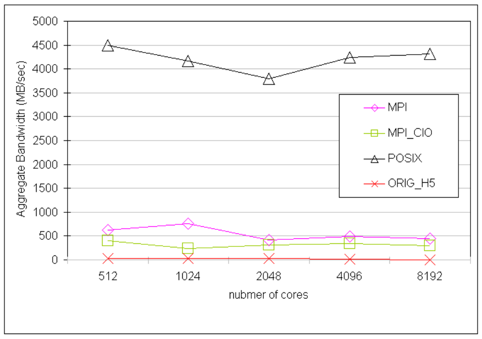

Fig. 3. Chimera Aggregate Bandwidth

TABLE I
PARALLEL HDF-5

Parallel HDF-5		
Function	*# of calls*	*Total Time (sec)*
write	144065	33109.67
MPI_Bcast	314800	12259.30
MPI_File_open	2560	325.17
H5P, H5D, etc.	–	8.71
other	–	60

TABLE II
ADIOS INDEPENDENT MPI-IO

ADIOS Independent MPI-IO		
Function	*# of calls*	*Total Time (sec)*
write	2560	2218.28
MPI_File_open	2560	95.80
MPI_Recv	2555	24.68
other	–	65

Chimera to assess the cost of converting it to an HDF-5 file that is identical to the one previously generated by Chimera.

1) Overall Chimera Evaluation: Chimera output is not particularly large. In a weak scaling model, each process outputs approximately 920KB, with the number of processors varying from 512 to 8192. The simulation is configured to run for 400 iterations with an output being performed every 50 iterations. The simulation is run 5 times at each size, collecting the timing for each output, for a total of 40 measurements per size per IO method. Graphs depict the best performance measured for each size for each method with an error bar to show the range of values seen for that size. Best times are shown in order to minimize the impacts other users of the machine have on measurements.

Important to note about Graph 2 are the facts that the vertical axis is exponential and that the performance of parallel HDF-5 stops scaling linearly at 2048 processors. Graph 3 notes the bandwidths to the storage system achieved with different IO methods. Clearly, these results demonstrate performance issues with HDF-5. We next investigate their principal causes.

2) Principal Causes of Overhead in Parallel HDF-5: Using a test case of 512 cores running the Chimera supernova code, 5 sets of restart dumps are used to analyze performance causes, with results appearing in Table I. Detailed profiling reveals the reasons for inadequate Parallel HDF-5 performance:

1) Expensive MPI_Bcast calls are called frequently. In the test case, MPI_Bcast is called 314,800 times with a total wall clock time cost of 12,259 seconds total across all 512 processes (mean of 23.9 seconds for each process overall, max of 54 seconds, min of 4.6 seconds).
2) Too many small, individual writes are performed. Individual write operations are performed 144,065 times with a total wall clock time of 33,109 seconds across all 512 processes (mean of 64.7 seconds for each process overall, max of 96 seconds per write, min of 46 seconds per write).
3) MPI_File_open calls take longer than necessary because it has not been optimized how the MPI_File_open calls are performed, so that the 2560 calls take a total of 325 seconds across all 512 processes (mean of 0.63 seconds).

3) Performance Analysis of ADIOS with BP format using Independent IO: For a test case of 512 cores running the Chimera supernova code, this evaluation uses 5 sets of restart dumps, with results appearing in Table II. In comparison to the parallel HDF-5 results shown above, this ADIOS run has a straightforward outcome:

1) Buffered writes take the longest time, since ADIOS, by default if memory is available, buffers all writes to the output file locally and then writes the buffered output in a single write operation to disk. MPI_File_write is called 2560 times with a total wall clock time of 2,218 seconds across all 512 processes (mean of 4.3 seconds, max of 11 seconds per write, min of 0.01 seconds (likely cache effects)).
2) By coordinating the MPI_File_open calls, the total time to open the file is reduced. Specifically, rather than have all processes call MPI_File_open at the same time, a coordination token is used (in a round robin fashion) to reduce the load on the metadata server. This reduces the time for the 2560 MPI_File_open calls to 95.80 seconds (mean 0.19 seconds). Including the time for the token passing, the total time for the MPI_File_open and MPI_Recv calls is still only 120.48 seconds across all 512 processes (mean of 0.23 seconds).

4) File Conversion Performance: Since the default implementation of the ADIOS MPI-IO, POSIX, and collective MPI-

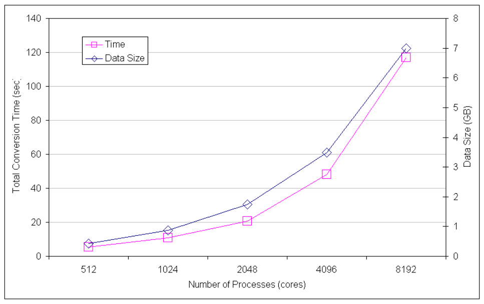

Fig. 4. BP to HDF-5 File Conversion on 1 processor

IO methods use the BP format, the time spent converting this format into the HDF-5 desired by the Chimera scientists is extremely relevant. Tests are performed on a single Jaguar login node reading from the BP file and writing to an HDF-5 file, both stored on the Lustre scratch space.

Five examples are profiled for file conversion performance, described in Figure 4. The left scale is the total conversion time for the file. The right scale is the size of the source file in GB. The bottom scale is the number of processes in the job generating the output. For the 2048 process case, Chimera generates a file of approximately 1.8 GB. This conversion takes about 20 seconds. For a larger example from a run of 8192 cores, the file generated is about 7 GB and is converted to HDF-5 in 117 seconds. Note that the native Parallel HDF-5 calls in Chimera take 1400 seconds to write *each* output while the MPI-IO independent method takes only 10 seconds. Thus, even when combined with the 127 second conversion time, this is a greater than 90% savings in IO time. Also note that even on a single processor, conversion scales linearly with size.

Regarding conversion, it is possible to completely 'hide' its costs by automatically performing it either 'in transit' or once data is stored on disks. The evaluation of such approaches is beyond the scope of this paper.

B. GTC Evaluation

The GTC code has two large output operations that both occur when restarts are written. The first is the normal state output to enable a restart. The second is a set of particles used for analysis. These particles are tracked as they rotate around the simulation toroid and yield some of the scientific data important as the output of the run. We configure GTC to run for 100 iterations with a restart/particles output every 10 iterations. Tests are run 5 times and again, the 'best' results are shown from the 50 outputs. The evaluation is performed for each of POSIX IO, independent MPI-IO, and collective MPI-IO.

For weak scaling tests, we use the Jaguar machine and run with OpenMP to communicate among the cores within a single node (4 cores per node). Strong scaling tests are run on the Ewok machine, without OpenMP.

GTC evaluations are divided into three parts. The first evaluates weak scaling with various ADIOS IO routines. The second evaluate the performance of GTC with strong scaling using various ADIOS methods for comparison. We provide these to demonstrate the applicability of ADIOS to a range of HPC applications beyond Chimera.

C. GTC Weak Scaling

The GTC configuration outputs particles of the size 11.5 MB per MPI process. Since OpenMP is used, this is the

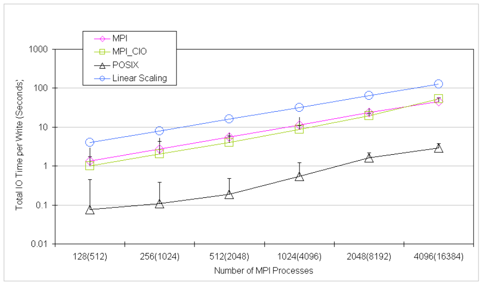

Fig. 5. GTC Particles Weak Scaling Time

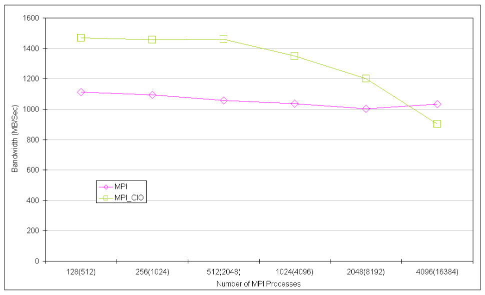

Fig. 6. GTC Particles Weak Scaling Aggregate Bandwidth MPI

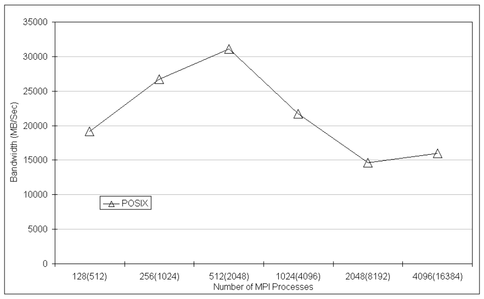

Fig. 7. GTC Particles Weak Scaling Aggregate Bandwidth POSIX

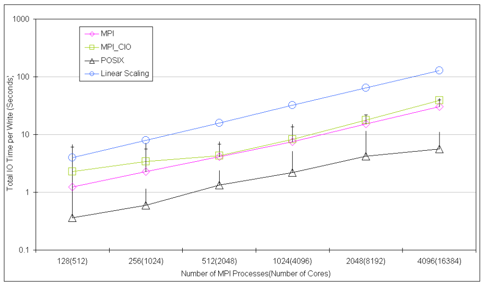

Fig. 8. GTC Restarts Weak Scaling Time

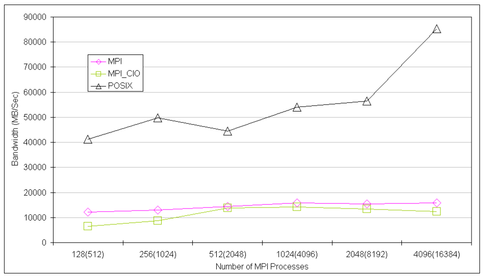

Fig. 9. GTC Restarts Weak Scaling Aggregate Bandwidth

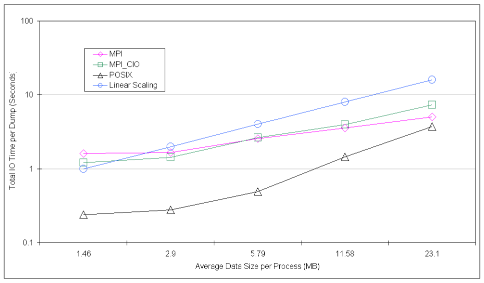

Fig. 10. GTC Particles Strong Scaling Time

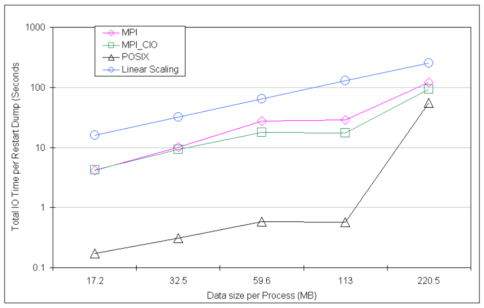

Fig. 11. GTC Restarts Strong Scaling Time

aggregate for the four cores on that particular node. For the restart output, the output of each MPI process is 116.5 MB.

For the particle output, shown in Graphs 5, 6, and 7, the POSIX output of one file per process is still considerably faster than the other approaches, the ‘cost’ of that approach being the large number of resulting files on disk. The more interesting result is in the bandwidth measurements. Collective MPI-IO stays about the same margin better than independent MPI-IO until about 1024 MPI processes. At 2048 processes, the margin is reduced considerably. At 4096 processes, the bandwidth is reduced below that of independent MPI-IO. This further emphasizes the observations in the Chimera runs that the coordination required for collective-style IO does not adequately scale.

For the restart output, in Graphs 8 and 9, the performance of output methods using collective MPI-IO is initially worse than that of the independent MPI-IO output, and it continues to degrade with increasing simulation sizes. While the particle data is relatively small at 11.5 MB per process, the restart data, at an order of magnitude larger, clearly demonstrates differences in IO performance. For such output, it would never be appropriate to use the collective MPI-IO method, instead favoring the independent IO method or POSIX methods, if it is possible to cope with the large number of files they create.

As shown above, both the particle and restart data differ in size and show different performance characteristics depending on the size of the run. This demonstrates the need for differentiating IO methods both by run and by output grouping. ADIOS enables such differentiation.

D. GTC Strong Scaling

Strong scaling results are attained on the Ewok end-to-end cluster at ORNL. Interesting insights are attained despite the cluster’s relatively small size.

Test runs with small data sizes are not meaningful due to caching effects, so discussion is focused on larger data runs on the largest number of processors. The results of these evaluations are shown for particles in Graph 10 and for restarts in Graph 11. The most important characteristic of these results is how relative performance differs between Jaguar and Ewok for the independent and collective MPI-IO calls. Although both of these machines are housed at the same location and maintained and configured by the same staff, their different architectures yield the opposite performance results of what we saw for the particles and restarts. When running on Ewok, or by extension probably other Infiniband-based Linux clusters, using collective MPI-IO for restarts and independent MPI-IO for particles is the better configuration. This further emphasizes the need to have configurable IO as part of scientific simulations in order to achieve best IO performance. ADIOS offers this capability, per IO grouping, by simply changing a single entry in the XML file when the job is submitted.

E. Evaluation Discussion

The performance advantages of the ADIOS approach are shown by measuring the performance of the native parallel HDF-5 output from the Chimera supernova code compared against various methods integrated with ADIOS. The overheads involved in parallel HDF-5 vs. independent MPI-IO are identified and compared with ADIOS performing independent MPI-IO. We show that the coordination and small writes required by HDF-5 for file consistency degrade performance by as much as three orders of magnitude compared to other methods. With a total time of 1400 for the real-time consistency output directly to HDF-5 vs. a net total time of less than 120 seconds to output data in HDF-5 via the BP intermediate file format using a delayed consistency method and conversion, it is hard to justify the wall clock expense during a production simulation run to perform real-time consistency validation.

The GTC fusion code is evaluated with both weak and strong scaling for two different outputs performed at the same time. We show that as the size of the simulation run increases and based on data size, it is appropriate to use different IO methods. By also running tests on the small end-to-end

Ewok cluster, performance differences and resulting changes in recommended IO methods are demonstrated.

Both of these results demonstrate the need for using delayed file consistency, with rapid conversion to a consistent HDF-5 and/or NetCDF file, and with configurable IO as provided by ADIOS. As platforms change or for different sizes of simulation run, the selection of the IO method for each grouping of data within the simulation is critical for minimizing the time spent in IO.

V. Conclusions and Future Work

For petascale machines, the performance penalties of using full internal file consistency during a production run can be too onerous. Through ADIOS, a developer can debug a code using an underlying API with active consistency checks, like parallel HDF-5 or parallel NetCDF, but during a production run, one can switch to whichever IO method gives the best performance, thus yielding the 'most science' with the least IO overhead. In some cases, this will be Parallel HDF-5 or Parallel NetCDF. In other cases, through the use of an intermediate format like BP, it is possible to deliver excellent performance while still maintaining sufficient metadata for easy conversion to the file format compatible with the science workflow already employed. Further, ADIOS' additional feature of data characteristics can aid in data selection on the large output sets.

We have demonstrated that in some, if not many cases, the use of alternative IO methods can yield dramatically better IO performance during production runs while still maintaining format compatibility via relatively cheap methods for file conversion. We conclude therefore, that to attain high IO performance on petascale machines, it is imperative to be able to configure the IO method employed for each IO grouping within a code differently, at runtime and without any source code changes. ADIOS provides such functionality.

The next release of ADIOS is focused on providing high levels of performance for read operations. We acknowledge that MxN-style read operations would require additional efforts on our part, including potentially custom IO methods to organize data as it moves to disk. For example, for climate modeling, data is widely distributed and potentially distributed differently on each run. Efficiently storing and retrieving such data may require methods that use a data staging area to organize the data, both as it is written to disk and read for distribution to the compute nodes. We have yet to develop these methods.

References

[1] HDF-5, "http://hdf.ncsa.uiuc.edu/products/hdf5/index.html."

[2] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, and R. Latham, "Parallel netcdf: A scientific high-performance i/o interface," 2003.

[3] P. M. Widener, M. Wolf, H. Abbasi, M. Barrick, J. Lofstead, J. Pullikottil, G. Eisenhauer, A. Gavrilovska, S. Klasky, R. Oldfield, P. G. Bridges, A. B. Maccabe, and K. Schwan, "Structured streams: Data services for petascale science environments," University of New Mexico, Albuquerque, NM, Tech. Rep. TR-CS-2007-17, November 2007. [Online]. Available: http://www.cs.unm.edu/ treport/tr/07-11/PetaScale.pdf

[4] P. M. Widener, M. Barrick, J. Pullikottil, P. G. Bridges, and A. B. Maccabe, "Metabots: A framework for out-of-band processing in large-scale data flows," in *Proc. 2007 International Conference on Grid Computing (Grid 2007)*, Austin, Texas, September 2007.

[5] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, no. 3, pp. 219–237, 2005.

[7] G. Malewicz, I. Foster, A. Rosenberg, and M. Wilde, "A tool for prioritizing dagman jobs and its evaluation," *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pp. 156–168, 0-0 2006.

[8] R. R. Sinha and M. Winslett, "Multi-resolution bitmap indexes for scientific data," *ACM Trans. Database Syst.*, vol. 32, no. 3, p. 16, 2007.

[9] L. Gosink, J. Shalf, K. Stockinger, K. Wu, and W. Bethel, "Hdf5-fastquery: Accelerating complex queries on hdf datasets using fast bitmap indices," in *In SSDBM*, 2006, pp. 149–158.

[10] H.-. P. Indices, "http://www.hdfgroup.uiuc.edu/rfc/hdf5/hdf5indexing/pis.html."

[11] H.-. P. Tables, "http://www.carabos.com/docs/opsi-indexes.pdf."

[12] NetCDF, "http://epp.eps.nagoya-u.ac.jp/num-analysis/guidec/netcdf-c-14.html."

[13] R. W. Watson and R. A. Coyne, "The parallel i/o architecture of the high-performance storage system (hpss)," in *MSS '95: Proceedings of the 14th IEEE Symposium on Mass Storage Systems*. Washington, DC, USA: IEEE Computer Society, 1995, p. 27.

[14] NFS, "http://www.ietf.org/rfc/rfc3010.txt."

[15] R. Oldfield, L. Ward, R. Riesen, A. Maccabe, P. Widener, and T. Kordenbrock, "Lightweight i/o for scientific applications," *Cluster Computing, 2006 IEEE International Conference on*, pp. 1–11, 25-28 Sept. 2006.

[16] P. J. Braam, "Lustre: a scalable high-performance file system," Nov. 2002. [Online]. Available: http://www.lustre.org/docs/whitepaper.pdf

[17] Panasas, "http://www.panasas.com/."

[18] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *In Proceedings of the 2002 Conference on File and Storage Technologies (FAST*, 2002, pp. 231–244.

[19] P. NFS, "http://tools.ietf.org/wg/nfsv4/."

[20] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, "Serverless network file systems," *ACM Trans. Comput. Syst.*, vol. 14, no. 1, pp. 41–79, 1996.

[21] M. C. Miller, J. F. Reus, R. P. Matzke, W. J. Arrighi, L. A. Schoof, R. T. Hitt, and P. K. Espen, "Enabling interoperation of high performance, scientific computing applications: Modeling scientific data with the sets & fields (saf) modeling system," in *International Conference on Computational Science (2)*, 2001, pp. 158–170.

[22] P. D. System, "http://pds.jpl.nasa.gov/documents/sr/index.html."

[23] G. in Binary, "http://www.sesp.cse.clrc.ac.uk/publications/data-management/report/node12.html."

[24] H. D. System, "http://www.sesp.cse.clrc.ac.uk/publications/data-management/report/node14.html."

[25] N. Operators, "http://nco.sourceforge.net/."

[26] N. Podhorszki, B. Ludaescher, and S. A. Klasky, "Workflow automation for processing plasma fusion simulation data," in *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science*. New York, NY, USA: ACM, 2007, pp. 35–44.

[27] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[28] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible io and integration for scientific codes through the adaptable io system (adios)," in *CLADE 2008 at HPDC*. Boston, Massachusetts: ACM, June 2008.

[29] C. Jin, S. Klasky, S. Hodson, W. Yu, J. Lofstead, H. Abbasi, K. Schwan, M. Wolf, W. Liao, A. Choudhary, M. Parashar, C. Docan, and R. Oldfield, "Adaptive io system (adios)." Cray User's Group, 2008.

[30] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Input/output apis and data organization for high performance scientific computing," in *In Pro-*

ceedings of Petascale Data Storage Workshop 2008 at Supercomputing 2008, 2008.

Acknowledgment

This work was funded in part by Sandia National Laboratories under contract DE-AC04-94AL85000, a grant from NSF as part of the HECURA program, a grant from the Department of Defense, a grant from the Office of Science through the SciDAC program, and the SDM center in the OSCR office.