SCORPIO: A Scalable Two-Phase Parallel I/O Library With Application To A Large Scale Subsurface Simulator

Sarat Sreepathi*, Vamsi Sripathi†, Richard Mills*, Glenn Hammond‡, G. Kumar Mahinthakumar§
*Oak Ridge National Laboratory
Oak Ridge, TN, USA
sarat@ornl.gov, rtm@eecs.utk.edu
†Intel Corporation
Hillsboro, OR, USA
vamsi.sripathi@intel.com
‡Pacific Northwest National Laboratory
Richland, WA, USA
glenn.hammond@pnnl.gov
§North Carolina State University
Raleigh, NC, USA
gmkumar@ncsu.edu

***Abstract*—Inefficient parallel I/O is known to be a major bottleneck among scientific applications employed on supercomputers as the number of processor cores grows into the thousands. Our prior experience indicated that parallel I/O libraries such as HDF5 that rely on MPI-IO do not scale well beyond 10K processor cores, especially on parallel file systems (like Lustre) with single point of resource contention. Our previous optimization efforts for a massively parallel multi-phase and multi-component subsurface simulator (PFLOTRAN) led to a two-phase I/O approach at the application level where a set of designated processes participate in the I/O process by splitting the I/O operation into a communication phase and a disk I/O phase. The designated I/O processes are created by splitting the MPI global communicator into multiple sub-communicators. The root process in each sub-communicator is responsible for performing the I/O operations for the entire group and then distributing the data to rest of the group. This approach resulted in over 25X speedup in HDF I/O read performance and 3X speedup in write performance for PFLOTRAN at over 100K processor cores on the ORNL Jaguar supercomputer. This research describes the design and development of a general purpose parallel I/O library called Scorpio that incorporates our optimized two-phase I/O approach. The library provides a simplified higher level abstraction to the user, sitting atop existing parallel I/O libraries (such as HDF5) and implements optimized I/O access patterns that can scale on larger number of processors. Performance results with standard benchmark problems and PFLOTRAN indicate that our library is able to maintain the same speedups as before with the added flexibility of being applicable to a wider range of I/O intensive applications.**

***Keywords*-Parallel I/O; High Performance Computing; Subsurface modeling;**

I. Introduction

This paper describes the development of a highly scalable application layer parallel I/O library, SCORPIO (SCalable block-ORiented Parallel I/O) for scientific applications. SCORPIO provides a higher level API (Application Programming Interface) to read and write large scientific datasets in parallel at very large processor counts. The primary design goal is to abstract away the complexity of dealing with lower level parallel I/O libraries (HDF5 [1], netCDF [2], MPI-IO [3] etc.) while ensuring performance scalability. This is accomplished by dividing the traditional I/O operations (read/write) into two phases, a communication phase and an I/O phase.

The rest of the paper is organized as follows. In Section II, we present the motivation for developing a parallel I/O library by illustrating the I/O patterns used and scalability issues that were encountered for a large scale reactive multiphase flow and transport code, PFLOTRAN. We then discuss the two-phase I/O strategy developed by Sripathi et al. [4] [5] for PFLOTRAN that resolved the performance issues identified in Section III. Section IV describes the design of the general purpose parallel I/O library (SCORPIO) incorporating the two-phase I/O strategy into any application. Section V presents performance results for some PFLOTRAN test problems using SCORPIO.

II. PFLOTRAN

The U.S Department of Energy (DOE) is interested in studying the effects of geologic sequestration of CO2 in deep reservoirs and migration of radionuclides and other environmental contaminants in groundwater. Modeling of subsurface flow and reactive transport is necessary to understand these problems. PFLOTRAN [6] [7] is a highly scalable subsurface simulation code that solves multi-phase groundwater flow and multicomponent reactive transport in three-dimensional porous

media. It is written in an object-oriented style in Fortran 90 and employs domain-decomposition parallelism and makes extensive use of PETSc (Portable, Extensible Toolkit for Scientific Computation) [8] numerical solvers and distributed data structures. It uses MPI (Message Passing Interface) [9] through PETSc for inter-process communication and parallel HDF5 (Hierarchical Data Format 5) [1] library to perform simulation I/O.

A. PFLOTRAN I/O

The execution of PFLOTRAN can be divided into 4 different phases: initialization, flow, transport and output. The initialization phase mainly consists of subroutines which read the simulation data from HDF5 input files. The HDF5 input files are internally organized as a collection of HDF5 groups with each group containing multiple 1-dimensional HDF5 datasets. The largest test problem used reads multiple datasets where each dataset contains 1.7 billion elements. The simulation output comprises of multiple 3-dimensional datasets written to a single file. Each process writes to a non-contiguous region of the file and all such writes are interleaved between the processes. HDF5 collective I/O access mode is used to reduce the number of I/O disk accesses by combining the small non-contiguous write requests from different processes into larger contiguous write requests. All processes participate in the parallel I/O operations.

There are two different read access patterns in PFLOTRAN and one write pattern as illustrated in Fig. 1.

- Read Pattern-1: Read the same contiguous region of a dataset.
- Read Pattern-2: Read a chunk of the dataset determined by the process specific offset values.
- Write Pattern-1: Each process writes a 3-d block of data to a specified offset in the global dataset.

Read Pattern-1 is used for reading datasets like boundary connectivity list that are not evenly distributed across processors. Hence all processors read the entire dataset although each processor may only need portions of the list. Read Pattern-2 is used for datasets that map directly to the grid such as porosity and permeability datasets where offsets can be easily calculated.

B. Performance

We briefly present performance analysis of PFLOTRAN using a 270 million DoF (degrees of freedom) test problem on Jaguar, the Cray XT5 supercomputer previously deployed at Oak Ridge National Laboratory. It can be observed from Fig. 2 that there is significant performance degradation in both initialization (read) and write phases as we increase the number of processors. HDF5 collective mode is used for writes in this scenario. In contrast, HDF5 independent mode would incur even higher performance penalty at scale due to an increased number of independent I/O requests. The performance penalty at scale can be traced to large number of I/O requests and inordinately expensive file open calls. Whenever a process needs to open or close a file, it needs to poll the MDS (Meta Data Server) of the Lustre parallel file system [10]. As there is only one MDS for the entire file system, it becomes a performance bottleneck because of resource contention. As the number of processors increases, the file open and close calls become very expensive thereby becoming a scalability impediment. This has been identified as a scalability problem in previous studies [11] [12]. This problem is aggravated when using shared computing resources where the target application is run along with other applications that also stress the I/O system on the machine. For further details, the reader is referred to the performance analysis study by Sripathi [4].

III. Optimized I/O pattern

To alleviate the aforementioned scalability issues, Sripathi [5] identified the following:

- Need to combine multiple small I/O disk requests into one large disk request to decrease the number of I/O disk accesses at higher processor counts.
- Moreover, all processors cannot afford to access the file simultaneously because of current Lustre file system limitations (One Meta Data Server).

Hence the two-phase I/O approach was developed [4] [5] wherein all processes participate in the communication phase but only a limited number of processes participate in the I/O phase. For file read operations, the I/O phase is followed by the communication phase and conversely for write operations. Fig. 3 illustrates the access patterns involved in the two-phase I/O protocol.

The two-phase I/O method is implemented at the application level by splitting the MPI global communicator into multiple sub-communicators. The total MPI processes are divided into groups of sequentially ranked processes and a MPI communicator is created for each such group. The size of the group is a configuration parameter. The final group will have fewer processes than the number specified by group size when the total number of processes is not exactly divisible by group size.

The root process in each sub-communicator is responsible for performing the I/O operations for the entire group and then distributing the data to the rest of the group. By careful selection of the I/O root processes we avoid network congestion during the communication phase. Similarly communicators are created for performing write operations. The size of the sub-communicators determines the number of I/O processes created.

The access pattern within PFLOTRAN is either a contiguous access or an interleaved contiguous access to a dataset. So, the root I/O process consolidates the numerous independent disk requests into a large contiguous disk request and reads the data. Once the root process reads the data, it distributes the data to the rest of the group. The I/O processes need not allocate extra memory when performing read pattern-1 because all processes need to read the same contiguous data. On the other hand, they need to allocate temporary storage for the entire group while performing read pattern-2.

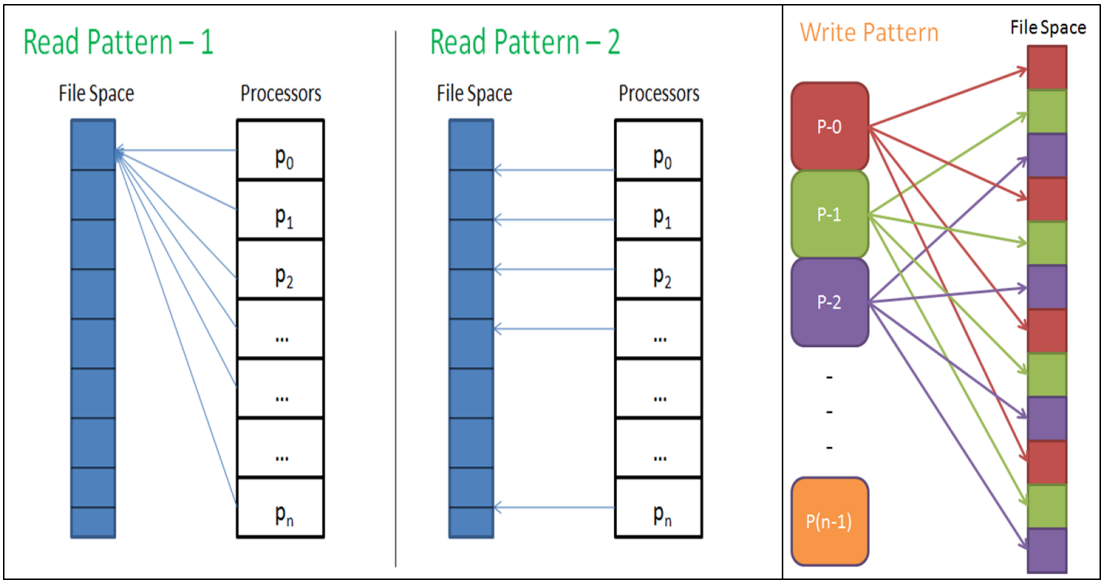

Fig. 1. PFLOTRAN Default I/O access patterns

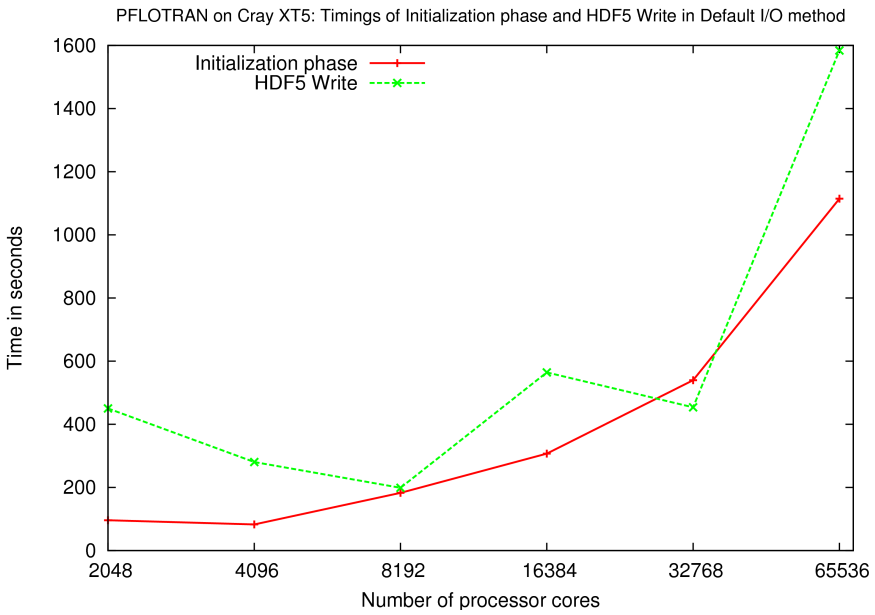

Fig. 2. PFLOTRAN performance on Jaguar using default I/O

Fig. 4 shows the impact of group size on performance of initialization phase on Jaguar. It can be observed that a large group size results in better performance due to fewer readers. Using the two-phase I/O approach, we were able to achieve 25X improvement in the initialization (read) phase of PFLOTRAN at 65,536 cores of Cray XT5. The single process I/O by process 0 in Fig. 4 refers to the case when all the input data is read and broadcast by one process. It must be noted that this scenario is inherently non-scalable for petascale applications because of memory limitations in case of large datasets. We observe from Fig. 4 that read performance is effectively identical for group sizes of 1024 and higher. Based on this empirical evidence, we recommend 1024 as a good choice for read group size at large processor counts for this application. Generally, read group size should be determined based upon the amount of data to be read and total number of parallel processes.

The default write method does not scale well despite using collective I/O mode. This can be traced to the overhead associated with collective I/O at higher processor counts. Hence we reduced the number of processes participating in a collective write call by pooling them into groups and assigning an I/O root process to write for each group. Fig. 5 shows a 3X improvement in write performance by using improved versions

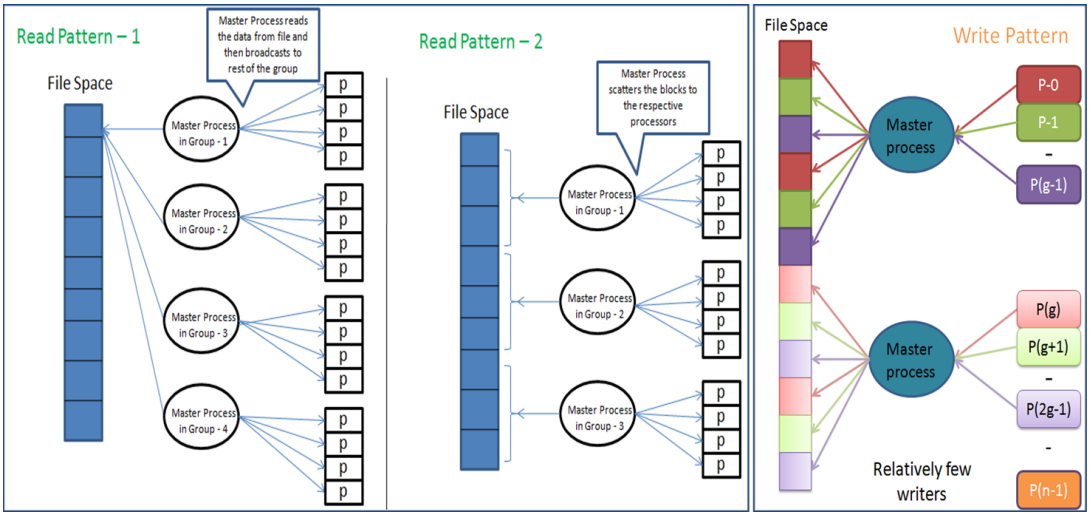

Fig. 3. Optimized two-phase I/O access patterns

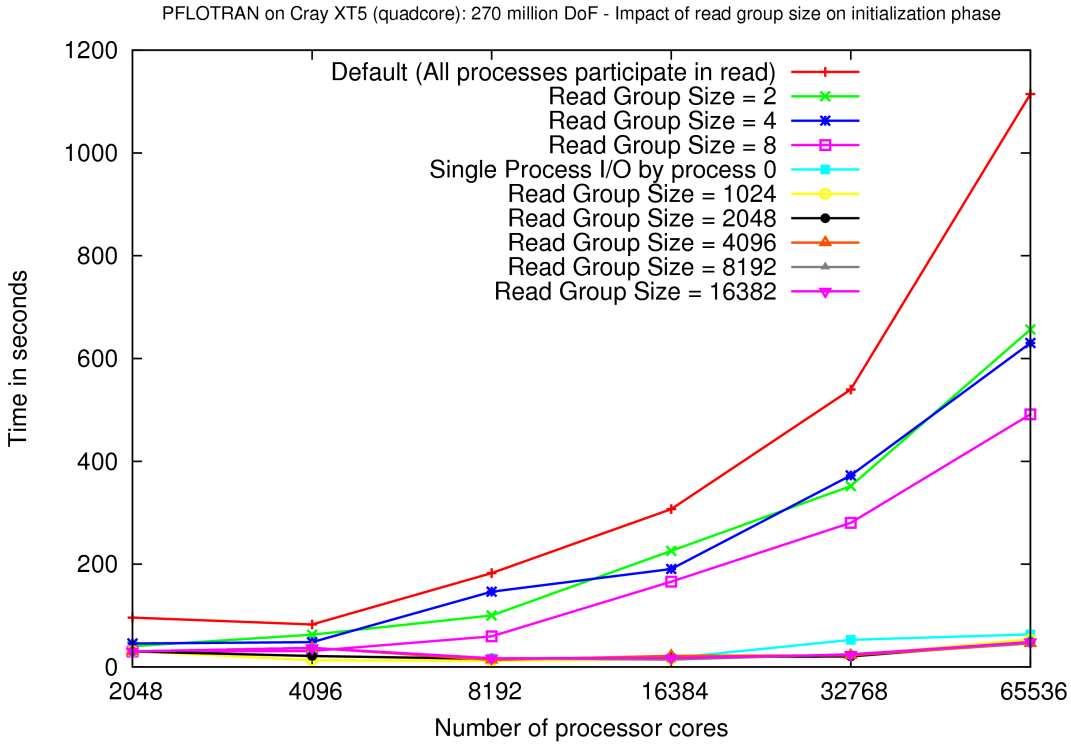

Fig. 4. Performance of optimized PFLOTRAN Initialization phase on Jaguar

of HDF5 write operations with a group size of 16.

The performance improvement in write phase is lower relative to improvement in read phase because of the noncontiguous nature of write pattern which prevents consolidation of multiple write requests into a single contiguous request. Thus each I/O root process has to issue separate write requests for each process in its I/O group. Consequently, a large write group size would degrade performance as each I/O root process would be issuing a large number of write requests in sequence. Hence care must be taken while selecting write group size to balance the number of writers with the number of separate I/O requests by each I/O root process. We recommend starting out with a small group size unless consolidation is not supported for specific access patterns.

The 3X improvement in write performance and 25X improvement in read performance on 64k cores of Jaguar on this test problem resulted in an overall speedup of 5X for the entire application [4].

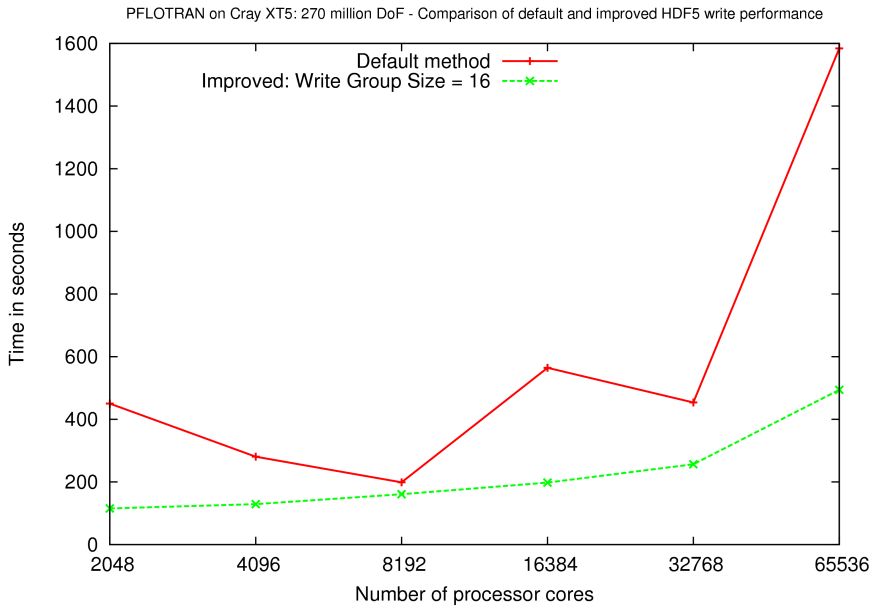

Fig. 5. Performance of optimized PFLOTRAN write phase on Jaguar

IV. SCORPIO LIBRARY

This research has evolved into part of a larger Department of Energy initiative, Advanced Simulation Capability for Environmental Management (ASCEM) [13]. This project is divided into three thrust areas of which the development of parallel I/O algorithms (SCORPIO library) plays a key role in the success of the HPC simulator. This library was envisioned to leverage our earlier I/O optimization experience to build a scalable general purpose parallel I/O capability for any application. Specifically, the goal is to take advantage of existing parallel I/O libraries, such as HDF5 which are being widely used by scientific applications and modify these algorithms to better scale on larger number of processors.

A. Related Work

The idea of decomposing I/O into two stages, namely communication and I/O has previously been explored [16], including in ROMIO [3]. Sripathi et al. [5] recognized the potential for such an approach to mitigate I/O performance issues on current leadership class supercomputers by reducing the number of readers and writers. Compared to some related work that targets I/O layers at a lower level including ROMIO, our approach enables us to tailor the I/O configuration including buffering strategy to better suit the target application requirements.

Recently ADIOS [14] developed some capabilities (`MPI_AMR` transport) for I/O aggregators. To our knowledge, their approach requires use of a custom data format and results in generation of multiple sub-files (one per aggregator) which may necessitate additional processing using ADIOS utility programs. In contrast, our approach works directly with HDF5 files and reads/writes to a single HDF5 file.

Choudhary et al. [17] demonstrated optimizations such as subfiling for parallel netCDF that resulted in substantial improvements in write bandwidth. Liao and Choudhary [19] developed a new file domain partitioning method (`group-cyclic`) as a replacement to the `even` partitioning model used by ROMIO [3] that delivered much better write performance on Lustre filesystems.

B. Architecture

The SCORPIO library implements the two-phase I/O approach wherein the total number of processors will be grouped into a small number of I/O pools and in each I/O pool a single processor will be designated as the I/O processor to perform disk I/O operations. For reads, these I/O processors will first read the entire data associated with its group and then scatter it to each processor within its group using a user specified format. For writes, these I/O processors will first gather the data from each processor within its group and then write to the disk. The gathering and scattering of data is carried out using MPI library calls (`MPI_Scatterv, MPI_Gatherv`).

The basic architecture of the SCORPIO library is depicted in Fig. 6. The user specifies the I/O group size and basic attributes of the I/O operation (file name, read/write, contiguous/non-contiguous, offsets, data dimensions etc.). The library creates MPI sub-communicators equal to the number of I/O groups i.e., the total number of processors divided by I/O group size. Advanced users can create multiple concurrent I/O groups of different sizes customized for different kinds of access patterns.

The first phase of the library has been completed and tested with up to 144K cores on the ORNL Jaguar machine. The library includes support for various types of read and write operations through well-defined Fortran and C bindings

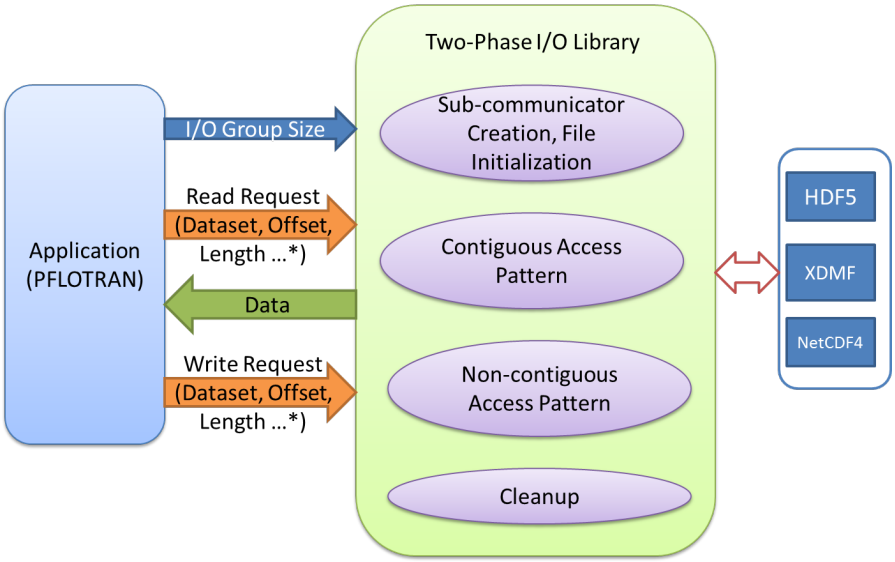

Fig. 6. Overall architecture of the SCORPIO library

(compatible with C++) and hence callable from Fortran, C and C++ applications.

C. Supported I/O Patterns

The SCORPIO library supports various commonly used I/O patterns. Fig. 7 illustrates some supported `uniform` and `nonuniform` contiguous access patterns. In both scenarios, each process accesses a contiguous region in the dataset. Each dataset is partitioned among all processes (four processes in the illustrations shown). In the `uniform` scenario, every process accesses the same extent/length of data whereas in the `nonuniform` scenario, each process can access different extents of the dataset. In case of n-dimensional datasets, the data partitioning is performed along the first dimension. Advanced users have the flexibility to customize their I/O requests and offsets according to their needs using the non-uniform access pattern.

In addition to these patterns, a generic block writing pattern is implemented to support writing an n-dimensional data block to a specified offset in the global dataset. Applications such as PFLOTRAN benefit from this extension as every process writes several non-contiguous 3-dimensional blocks at a specified offset during the output phase.

The library enhances user productivity by implicitly calculating the requisite offsets and lengths wherever possible. In case of `uniform_contiguous` access pattern, an I/O root process can derive the offset at which data has to be written from the overall dimensions of the dataset and the total number of parallel processes. For `nonuniform_contiguous` access pattern, additional coordination is required among the I/O root processes to calculate the offset.

D. Brief description of user callable routines

The SCORPIO library consists of a number of routines that the user can invoke for performing I/O operations as described in Table 1. These routines perform operations such as initialization, opening/closing of a file, read operations, and write operations. In Table 1, it is implied that each library call is prefixed by `scorpio_`.

V. Performance

A micro-benchmark that includes part of PFLOTRANs I/O behavior (Read pattern-2 and 1-d write of same data) was developed to test the efficacy of the SCORPIO library. Performance analysis was carried out on the ORNL Jaguar PF (Cray XT5) machine employing up to 144K cores using two real datasets associated with the PFLOTRAN application. The performance of the two-phase SCORPIO library was compared against that of the default HDF collective I/O approach in Fig. 8. The read and write group sizes used in the two-phase approach for the results presented in this section are 1024 and 16 respectively.

The I/O operation consists of reading two one-dimensional datasets from an input file used in the PFLOTRAN application and then writing them to an output file. The validation was performed by doing an `h5diff` between the original file and the written file. We investigated two test scenarios, (a) reading two $960 \times 960 \times 240$ datasets from a file (1.7 GB) and writing them back to an output file and (b) reading two $1920 \times 1920 \times 480$ datasets from a larger file (14 GB) and writing them out.

The read and write bandwidth obtained for the I/O benchmark using the 14 GB dataset on 144k cores are 301 MB/s and 407 MB/s respectively for two-phase I/O method. The bandwidth calculations include communication as well as HDF5

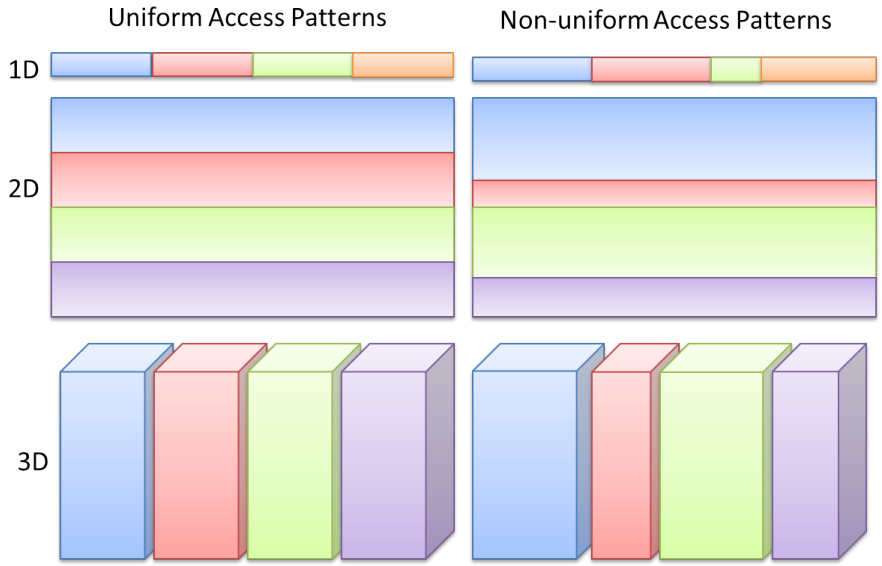

Fig. 7. Supported Access Patterns

TABLE I
SCORPIO USER CALLABLE FUNCTIONS

Function	Description
IOgroup_init	Initializes I/O groups by splitting the user given communicator into a number of sub-communicators equal to the specified number of I/O groups.
open_file	Opens a file for read or write operation. Available file modes are `FILE_CREATE`, `FILE_READONLY` and `FILE_READWRITE`.
read_dataset	Read data from a previously opened file. A number of read patterns are available including `uniform_contiguous`, `nonuniform_contiguous`, and `entire_dataset`. Various datatypes are supported including float, double, integer, long, char and byte. Block reads from multidimensional arrays are also supported.
write_dataset	Write data to a previously opened file. All functional equivalents to available read patterns are available in writes.
write_dataset_block	Writes a n-d block of data at the provided offsets in the global dataset in the file.
read_same_sub_dataset	Every process reads the same portion of the dataset.
get_dataset_size	Query the total size of a dataset from a previously opened file.
get_dataset_ndims	Query the number of dimensions of a specified dataset.
get_dataset_dims	Query the detailed dimensions of a specified dataset.
create_dataset_group	Creates a group within a HDF file that is used for writing datasets.
close_dataset_group	Closes a group in a HDF file that is previously opened using create_dataset_group.
close_file	Closes a previously opened file.

I/O operations. Additionally, shared-file I/O performance degrades at scale due to lock contention from file system consistency control mechanism [18]. While these bandwidth numbers are low compared to expected Lustre performance, these numbers are comparable to the values of < 1 GB/s reported in Choudhary et al. [17] for single shared-file I/O for Flash I/O at lower processor counts on Jaguar.

We observed significant performance variability during these experiments that may be attributed to system noise. During our investigation, we found that default I/O method required considerably higher Cray message portal resources to work with the 14 GB dataset. This is a pertinent benefit of two-phase I/O approach when dealing with large datasets at scale.

Timing is nearly flat for SCORPIO for the large file scenario whereas for default HDF I/O it increases nearly exponentially. SCORPIO is around 12 times faster than default I/O for 1.7 GB file scenario and more than 4 times faster for the 14 GB file.

The timing breakdown in Fig. 9 shows that most of the performance gain comes while opening the file. In the default HDF I/O, all processors participate in opening the file making it very expensive at large processor counts. In SCORPIO, most of the performance loss is in the Init operation where the MPI sub-communicators are created. Further analysis indicated that `MPI_Comm_split` does not scale due to an inefficient sorting scheme among other issues in the MPI library. Communication with the MPI developers revealed that they fixed this in the upcoming release.

We integrated the two-phase I/O library with PFLOTRAN and used it to study performance gains over the default I/O approach for a $960 \times 960 \times 240$ grid test problem (1.7 GB input and 5.8 GB output) as shown in Fig 10. We expect the performance gains to improve further for larger test problems using multiple datasets as PFLOTRAN invokes a file open/read call for every dataset read and file open is an expensive call at scale as demonstrated in Fig. 9. Moreover, write performance

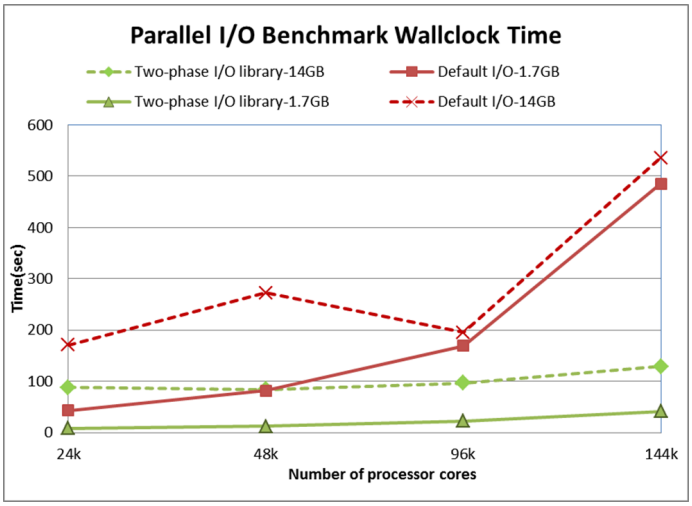

Fig. 8. I/O performance of default I/O (HDF collective) with two-phase SCORPIO library. At 144k cores the wallclock time for default I/O method is 535 secs compared to 129 secs for two-phase library method for the 14 GB file scenario.

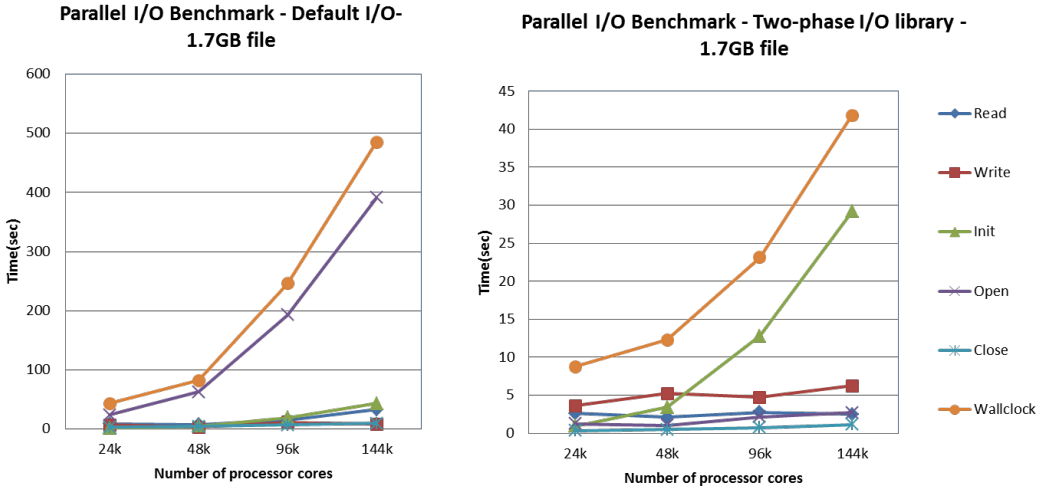

Fig. 9. Detailed performance analysis of Parallel I/O benchmark.

can be further enhanced by consolidating the writes in the two-phase I/O library.

VI. Conclusions and Future Work

This paper presented SCORPIO, an application layer parallel I/O library that implemented a two-phase I/O strategy. Performance results demonstrated that SCORPIO can improve the scalability of large scale scientific applications like PFLOTRAN on leadership class supercomputers. Furthermore, we facilitate multiple I/O group configurations to co-exist during runtime to cater to different stages of the application. Hence an application can choose an appropriate group size for various phases of execution (e.g., initialization, input, output, checkpointing).

In order to derive further performance gains while writing multi-dimensional blocks, consolidation strategies are designed and currently in development to reduce the number of independent I/O requests. The current version of the SCORPIO library assumes that I/O segments associated with each processor do not overlap except for the cases where the entire dataset or same portion of the dataset is being read by all processors. This limitation will be removed in a future release. Another limitation in the current library is the assumption that adequate memory is available in each I/O processor to hold the data from all processors in its group. Future releases will include increasingly sophisticated buffering strategies including the provision to customize the maximum I/O buffer size that can be used by each I/O processor. Additionally, SCORPIO library can incorporate improvements in the underlying I/O libraries such as the ones carried out by Choudhary et al. [17] if they are extended to HDF5. Ongoing work focuses on extending support for multidimensional unstructured data. SCORPIO supports native HDF format and with little additional effort can support netCDF (version 4.0) [2] as it internally uses HDF format for storage. Extensions to support other data formats such as XDMF [15] are planned in future releases.

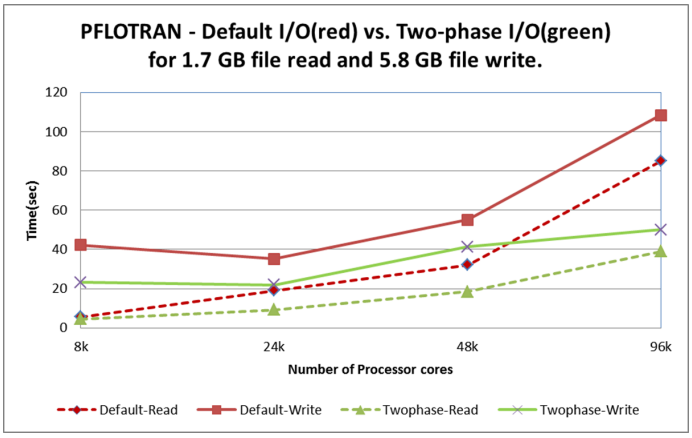

Fig. 10. PFLOTRAN performance with default I/O and two-phase I/O library on Jaguar.

ACKNOWLEDGMENTS

The submitted manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. The initial two-phase I/O development work specific to the PFLOTRAN application has been supported by the Performance Engineering Research Institute (PERI) and is funded by the U.S. Department of Energy Office of Science, as part of its second Scientific Discovery through Advanced Computing (SciDAC-2) research program. The I/O library development work is initially funded by the DOE Office of Environmental Management ASCEM project (Advanced Simulation Capability for Environmental Management).

REFERENCES

[1] HDF5, http://www.hdfgroup.org/HDF5/, date accessed: June 2013.
[2] NetCDF, http://www.unidata.ucar.edu/software/netcdf/, date accessed: June 2013.
[3] R. Thakur, W. Gropp, and E. Lusk, *Data sieving and collective I/O in ROMIO*, Frontiers of Massively Parallel Computation, pages 182-189, February 1999.
[4] Vamsi Sripathi, *Performance analysis and optimization of parallel I/O in a large scale groundwater application on petascale architectures*, M.S. Thesis, North Carolina State University, June 2010. http://www.lib.ncsu.edu/resolver/1840.16/6098
[5] Vamsi Sripathi, Glenn E. Hammond, G. (Kumar) Mahinthakumar, Richard T. Mills, Patrick H. Worley and Peter C. Lichtner (2009), *Performance Analysis and Optimization of Parallel I/O in a large scale groundwater application on the Cray XT5*, Poster Presentation, Supercomputing 2009, Portland, Oregon, Nov 12-16.
[6] David Keyes, Editor, *Towards petascale computing in Geosciences: Application to the Hanford 300 area*, Volume 125, Seattle,Washington, 2008. Journal of Physics: Conference Series, page 012051, IOP Publishing.
[7] Richard Tran Mills, Glenn E Hammond, Peter C Lichtner, Vamsi Sripathi, G (Kumar) Mahinthakumar, and Barry F Smith, *Modeling subsurface reactive flows using leadership class computing*, Journal of Physics: Conference Series, 180(1):012062, 2009.
[8] PETSc Home page, http://www.mcs.anl.gov/petsc, date accessed: June 2013.
[9] MPI standard, http://www.mcs.anl.gov/research/projects/mpi/, date accessed: June 2013.
[10] Lustre, http://wiki.lustre.org/index.php/Main_Page, date accessed: June 2013.
[11] M. Fahey, J. Larkin, and J. Adams, *I/O performance on a massively parallel Cray XT3/XT4*, IEEE International Symposium on Parallel and Distributed Processing, pages 1-12, 14-18 2008.
[12] Weikuan Yu, J.S. Vetter, and H.S. Oral, *Performance characterization and optimization of parallel I/O on the Cray XT*, IEEE International Symposium on Parallel and Distributed Processing, pages 1-11, 14-18 2008.
[13] Advanced Simulation Capability for Environmental Management (ASCEM), http://ascemdoe.org, date accessed: June 2013.
[14] Adaptable I/O System (ADIOS), http://www.olcf.ornl.gov/center-projects/adios/. Date accessed: June 2013.
[15] XDMF (eXtensible Data Model and Format), http://www.xdmf.org, date accessed: June 2013.
[16] del Rosario J, Brodawekar R and Choudhary A, *Improved Parallel I/O via a Two-Phase Run-time Access Strategy*, Workshop on I/O in Parallel Computer Systems at IPPS 93 pp 5670
[17] Alok Choudhary, Wei-keng Liao, Kui Gao, Arifa Nisar, Robert Ross, Rajeev Thakur and Robert Latham, *Scalable I/O and analytics*, SciDAC 2009, Journal of Physics: Conference Series 180 (2009) 012048, doi: 10.1088/1742-6596/180/1/012048.
[18] Ross R, Latham R, Gropp W, Thakur R and Toonen B, *Implementing MPI-IO Atomic Mode Without File System Support*, IEEE/ACM International Symposium on Cluster Computing and the Grid 2005.
[19] Liao W and Choudhary A, *Dynamically Adapting File Domain Partitioning Methods for Collective I/O Based on Underlying Parallel File System Locking Protocols*, International Conference for High Performance Computing, Networking, Storage and Analysis, 2008.