Performance Characterization and Optimization of Parallel I/O on the Cray XT

Weikuan Yu, Jeffrey S. Vetter, H. Sarp Oral
Oak Ridge National Laboratory
Oak Ridge, TN 37831
{wyu,vetter,oralhs}@ornl.gov

Abstract

This paper presents an extensive characterization, tuning, and optimization of parallel I/O on the Cray XT supercomputer, named Jaguar, at Oak Ridge National Laboratory. We have characterized the performance and scalability for different levels of storage hierarchy including a single Lustre object storage target, a single S2A storage couplet, and the entire system. Our analysis covers both data- and metadata-intensive I/O patterns. In particular, for small, non-contiguous data-intensive I/O on Jaguar, we have evaluated several parallel I/O techniques, such as data sieving and two-phase collective I/O, and shed light on their effectiveness. Based on our characterization, we have demonstrated that it is possible, and often prudent, to improve the I/O performance of scientific benchmarks and applications by tuning and optimizing I/O. For example, we demonstrate that the I/O performance of the S3D combustion application can be improved at large scale by tuning the I/O system to avoid a bandwidth degradation of 49% with 8192 processes when compared to 4096 processes. We have also shown that the performance of Flash I/O can be improved by 34% by tuning the collective I/O parameters carefully.

1 Introduction

Systems with unprecedented computational power are continuously pushing the frontier of high performance computing (HPC). Various sites have already launched efforts to build systems that can perform a thousand trillion floating point operations per second (10^{15} flops) [18, 23]. Furthermore, the U.S. Department of Energy has also launched initiatives to prepare for the future era of exa-scale computing [4]. On such large-scale systems, scientific applications, such as those in astrophysics, climate, fusion, combustion, biology, and chemistry, are very data-intensive, requiring adequate I/O capability. In addition, to cope with various forms of system reliability issues, applications need to checkpoint their intermediate results to the storage system, which further increases the need for scalable and efficient I/O. In fact, oftentimes, checkpoint operations are charged to users as allocated CPU hours, so they are desired to be as transparent and fast as possible. Hence, it is important for the scientists to understand the system's I/O software and storage architecture on their target platforms.

Jaguar is a Cray XT supercomputer platform at the Oak Ridge National Laboratory; it is equipped with a significantly large storage system. Jaguar provides computational services for a broad spectrum of applications, such as GTC for fusion [20], Parallel Ocean Program (POP) for ocean modeling, and Chimera for nuclear physics, among others. Not surprisingly, a team of researchers have performed an early performance evaluation of Jaguar [6]; they examined the majority of performance features of the processor, memory, and message passing system. This study, as well as several other studies on the I/O performance of Cray XT platforms [19, 27], has investigated the I/O subsystem using micro-benchmarks. These microbenchmarks have provided relevant information for scientific applications, such as peak system throughput, and the impact of Lustre file striping patterns. However, few insights are provided on how to correlate micro-benchmark results with the organization of the storage system, or their relevance to the applications' I/O performance.

In this paper, we present an extensive characterization of the performance of parallel I/O on Jaguar, and accordingly, the benefits of tuning and optimizing scientific application and benchmarks. First, using regular contiguous I/O patterns, we characterize the I/O performance of individual storage units, such as a single OST and a single storage couplet, S2A 9550 from Data Direct Networks, Inc [13]. Second, we evaluate the scalability trends of the entire system, in terms of both peak I/O bandwidth and the latency of metadata operations like parallel file open and creation. Third, we characterize the strengths and pitfalls of using parallel I/O techniques–data sieving and two-phase collective I/O–for small and non-contiguous data accesses on Jaguar. Finally, based on our characterization, we demonstrate that it is possible to optimize the I/O performance of benchmarks and applications. In one case, we have demonstrated that the I/O performance of S3D combustion application can be improved by using a shared file, avoiding a bandwidth drop of 49% for 8,192 processes. In another case, we have shown that the performance of flash I/O is improved by 34% by carefully tuning collective I/O parameters.

The rest of the paper is organized as follows. In the next section, we provide an overview of Jaguar and the configuration of its I/O subsystem. In Section 3, we provide a detail evaluation, characterization and tuning of Jaguar's I/O performance. In Section 4, we demonstrate the tuning and optimization of I/O patterns for applications and benchmarks over Jaguar. Section 5 concludes the paper.

2 Jaguar and Its I/O Subsystem

In this section, we provide an overview of Jaguar and its I/O subsystem.

2.1 An Overview of Jaguar

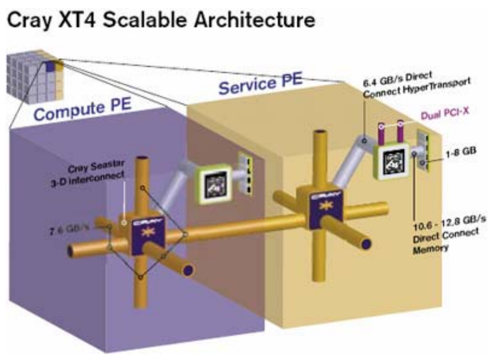

Cray XT4 Scalable Architecture
Compute PE
Service PE
6.4 GB/s Direct Connect HyperTransport
Dual PCI-X
1-8 GB
Cray Seastar 3-D interconnect
7.6 GB/s
10.6 - 12.8 GB/s Direct Connect Memory

Figure 1 Cray XT System Architecture of Jaguar (Courtesy of Cray)

Jaguar is ranked as the second fastest supercomputer as of June 2007 [3]. It is a combination of Cray XT3 and Cray XT4 technologies. Cray XT3 and Cray XT4 represent a line of massively parallel processor (MPP) products from Cray. They have similar architectures, except that XT4 is equipped with higher speed memory (DDR2-667MHz) and its SeaStar2 interconnect has higher sustained bandwidth.

The basic building block of Jaguar is a Processing Element (Compute PE or Service PE), as shown in Figure 1. Each PE has a dual-core AMD processor along with 2GB/core of memory, an integrated memory controller, HyperTransport links, and a dedicated communication chip–SeaStar/SeaStar2. Jaguar inherited its system software from a sequence of systems developed at Sandia National Laboratories and University of New Mexico: ASCI Red [33], the Cplant [24], and Red Storm [8]. Jaguar Compute PEs run a lightweight operating system called Catamount. The Catamount kernel runs only one single-threaded process and does not support demand-paged virtual memory. On the other hand, service PEs (i.e., login, I/O, network, and system PEs) run Linux to provide a user-familiar environment for application development and for hosting system and performance tools. Portals [10] is used for flexible, low-overhead inter-node communication on Cray XT. It delivers data from user space to user space between processes without kernel buffering.

2.2 Configuration of the Jaguar I/O Subsystem

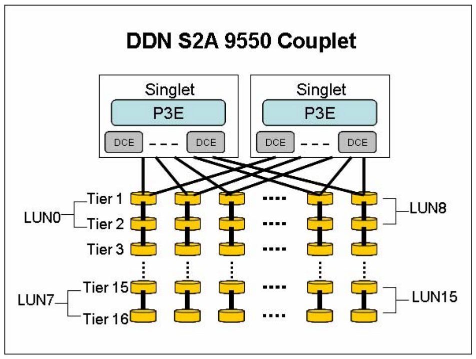


Figure 2: A Diagram of the S2A 9550 Couplet and its LUN configuration on Jaguar

Storage Hardware – The Jaguar I/O subsystem is provided by 18 Silicon Storage Appliance (S2A) 9550 storage targets [12] from Data Direct Network, Inc, often referred to as DDN S2A 9550 couplets. Each couplet has a capacity of 32 TB as shown in Figure 2. Every couplet is composed of two singlets, each containing a Parallel Parity Processing Engine (P3E) and a number of Disk Controller Engines (DCE). Within a couplet, there are 16 tiers of storage disks, offering storage as 16 LUNs (Logical Unit Number). P3E horizontally stripes a serial data stream into parallel segments to an array of DCEs, which in turn provides vertical striping of its data segments. Through vertical striping, every LUN spans two tiers of disks. All sixteen tiers share the same set of data channels to reach DCEs, P3Es and the fibre channel interfaces to the external hosts. On Jaguar, each LUN has a capacity of 2TB and a 4KB block size. The write-back cache is set to 1MB in each DCE.

File Systems -- Jaguar uses Lustre [11] for its IO subsystem. Lustre is an object-based parallel file system composed of four components: Object Storage Targets (OST), Metadata Servers (MDS), Object Storage Servers (OSS), and clients. Further details on Lustre are available in [11]. Jaguar is configured with three Lustre file systems, providing the scratch storage space for experimental data: scr144, scr72a and scr72b. 72 service nodes are configured as OSSes for these 3 file systems. In addition, each file system has its own dedicated MDS node. Every LUN is configured as a single OST for one of three Lustre file systems. Figure 3 shows the mapping of LUNs to the services nodes and Lustre file systems. The biggest file system, scr144, is equipped with 144 OSTs, i.e. 8 LUNs from every S2A 9550 storage devices, with a storage capacity of 288TBs; the other two, scr72a and scr72b, each with 72 OSTs - 8 LUNs each from 9 of S2A 9550 targets. In other words, each OSS node is configured with 4 OSTs. Two of these OSTs are for scr144, the remaining two OSTs for scr72a and scr72b, respectively.

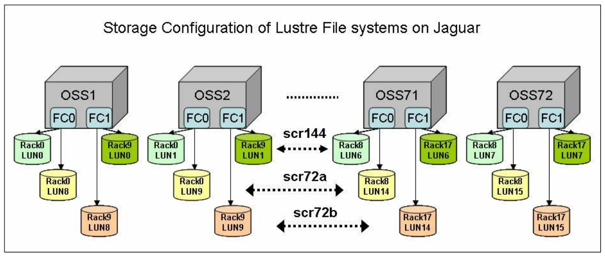


Figure 3: Assignments of OSTs (LUNs) for Lustre File systems on Jaguar

Parallel I/O libraries -- Parallel processes from compute PEs can directly invoke POSIX read/write functions or call through an MPI-IO library for I/O services. Cray provides a proprietary MPI-IO [30] implementation–referred to as AD_Sysio due to its leverage of the SYSIO library. There is also an open-source MPI-IO library called OPAL, which has been deployed as an alternative package on Jaguar. OPAL [39] is designed to provide a Lustre-specific implementation of the ADIO interface inside MPI-IO, enabling a number of good features such as arbitrary striping of MPI files and stripe-aligned domain partitioning. We have used this library for some of our experiments.

3 Parallel I/O Characterization on Jaguar

In this section, we examine the characteristics of individual storage components, the scalability of the entire storage system, the strengths and pitfalls of I/O techniques for small and non-contiguous I/O, and the scalability of metadata operations in terms of parallel file open and creation. The default Cray MPI-IO implementation is used for the majority of the experiments unless otherwise noted.

3.1 Contiguous, Independent I/O

Contiguous, independent I/O is one of the most common I/O patterns for scientific applications running on Jaguar. This scenario includes the pattern in which all processes writes/reads their own datasets to/from either separated files or separated regions of a larger shared file. We measure the performance of Jaguar storage using the IOR benchmark [2] from Lawrence Livermore National Laboratory. In addition, applications also have small input files, generally read by the leader process (rank 0) at the beginning of the execution, and then broadcasted to all other processes. Because these input files are of a small number and have a very low data volume, we omit these scenarios.

3.1.1 Single OST

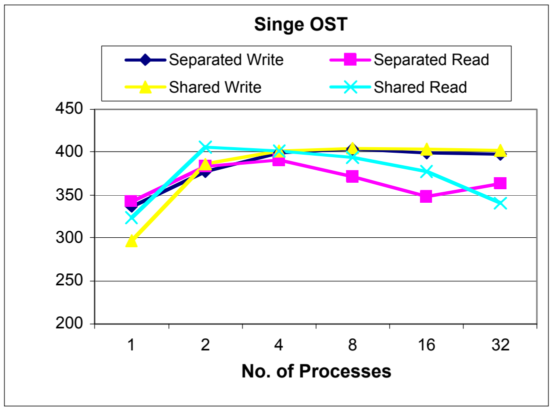
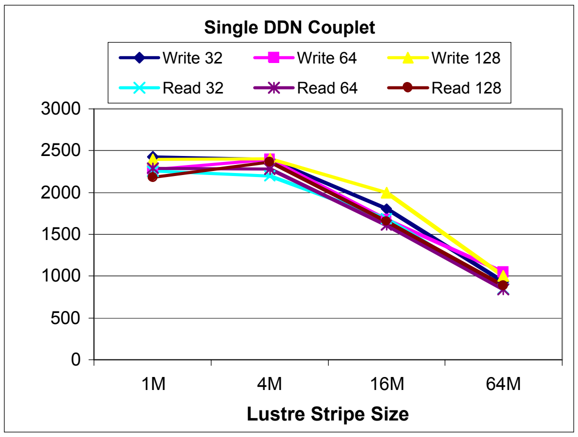


Figure 4: Performance of Single OST

In the single-OST experiments, a varying number of parallel processes are concurrently reading from or writing to a single OST, i.e. a LUN in a DDN couplet. The transfer size per client is set to vary between 4 and 64 MB. The total data volume is 512MB per process. Figure 4 shows the measurement results for both a shared file and separated files. The I/O bandwidth is not sensitive to transfer sizes in this experiment, so only results using a transfer size of 4MB are shown. In the case of a shared file, the maximum read or write bandwidths are measured at 406 MB/s, which come close to the peak bandwidth of a single OST with 4Gbps fibre channel links. Comparing the read and write bandwidths, it is evident that the write bandwidth has a much more graceful scaling trend under the hot-spot pressure from many processes. In addition, we have measured the bandwidth of single-OST using different Lustre stripe sizes. It is observed that the stripe size does not affect the I/O bandwidth within a single OST (data not shown). Also shown in the figure are the bandwidth results with separated files on a single OST. Again, the write bandwidth is more graceful to hot-spot pressure. Both reads and writes can reach close to the peak bandwidth per OST (406MB/sec).

3.1.2 Single DDN Couplet

To measure the performance of a single DDN couplet, all files are striped across 8 LUNs, i.e. all 16 tiers of a couplet. Figure 5(L) shows the bandwidth results of a single couplet with different Lustre file stripe sizes (1MB, 4MB, 16MB, and 64MB). Reads and writes are both measured with 32, 64, and 128 processes. As shown in the figure, the stripe sizes 1MB and 4MB are able to deliver the best performance, while 4MB stripe size is slightly better overall. Figure 5(R) shows the scalability of a single couplet with a varying number of processes. As shown in this figure, parallel reads provides better peak bandwidth compared to writes. The aggregated bandwidth per couplet increases initially with the increasing number of processes. The peak bandwidths for both reads and writes are reached with 16 processes. However, neither can sustain with a strong scaling on the number of processes. As seen earlier with single-OST experiments, the read bandwidth is again more susceptible to the pressure from an increasing number of processes.



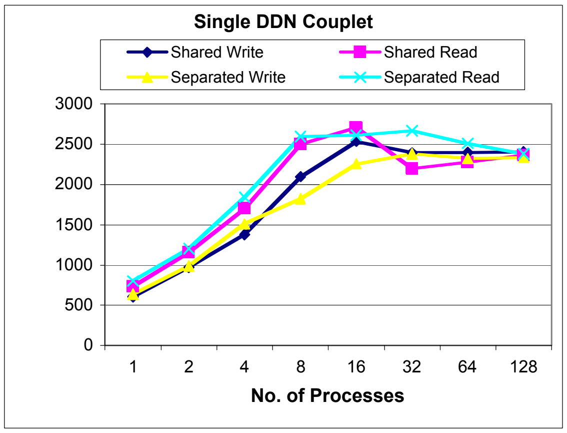


Figure 5: Single DDN Couplet: (L) Impact of Stripe Size (R) Performance Scalability

Figure 5(R) also shows the performance of a single couplet with separated files, using the stripe size of 4MB. In contrast to the single OST results, when the small number of processes is less than 64, the performance of reads is significantly better than that of writes. However, as the number of processes increases, both curves reach the plateau around 2500MB/sec. This resulting bandwidth is lower than the aggregation from 8 distinct LUNs – 3224MB/sec at 406MB/sec each. This means that a DDN couplet does not achieve linear scaling with respect to the number of tiers, due to the contention for the shared channels, engines, and external links when all tiers are utilized. Nonetheless, this number is in line with earlier observations made by the storage vendor [15].

3.1.3 System

We have also measured the scalability trends of the entire storage system, using the largest file system, scr144. The aggregated I/O bandwidths are measured when reading/writing files that are striped across different number OSTs, with 1024 processes. In the separated-file mode, each file is created to stripe across four OSTs. But all processes together access the same number of OSTs as the shared file is striped to. Figure 6 shows the performance comparisons of the two different modes with an increasing number of OSTs. Note that I/O with a shared file performs better than that with separated files when the number of OSTs is greater than 96.

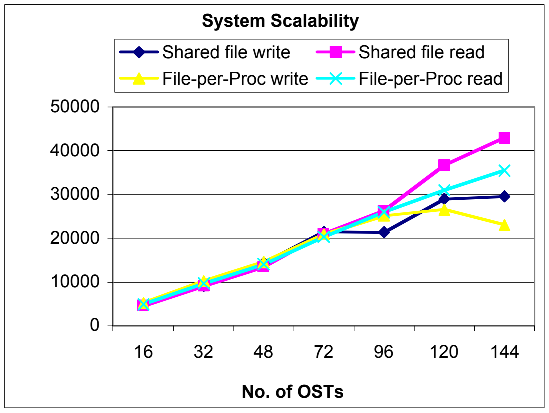


Figure 6: System Performance Scalability

These results suggest that the metadata overhead associated with separated files, though small initially, presents a scalability hurdle to the overall I/O bandwidth. In addition, the bandwidth of reads scales better compared to that of write when the number of OSTs is more than 96. Writes incur additional process costs, such as obtaining locks, allocating storage space, and the creation of objects. Even though a reserved pool of storage space and objects is provided in Lustre, the impact becomes pronounced with many widely striped files. Also note that, when files are located across a large number of OSTs/LUNs, the achieved peak bandwidth per OST is around 300MB/sec for both reads and writes. Our measurements are taken with 1024 processes, based on the fact that too many processes can lead to performance degradation – a phenomenon we showed in earlier figures and also observed by others from other sites with similar Cray XT platforms (c.f. Figure 1 and [19]).

3.2 Small, Non-Contiguous I/O

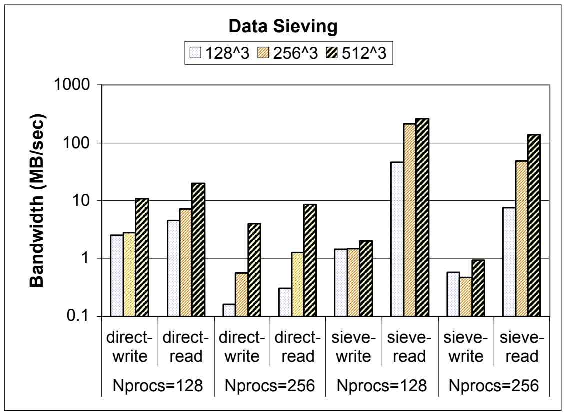
Small and non-contiguous I/O is another common pattern for scientific applications, especially when applications use higher-level, hierarchical data structures, such as multi-dimensional arrays with complex data decompositions among parallel processes. These higher-level abstractions typically translate into fine-grained, hierarchical, often non-contiguous, data accesses [31]. Such patterns can result in I/O bandwidths that are orders of magnitude lower than the peak of the physical storage hardware [28, 29]. Data sieving and collective I/O are strategies developed to improve the performance of small and non-contiguous I/O [17]. We obtained a program from Argonne National Laboratory that can characterize the benefits of these techniques. This program creates a 3-D global array, and decomposes the data along every dimension for I/O. This results in many small and non-contiguous I/O operations. The program allows three options for I/O: *direct, data-sieving, and collective I/O.* In the *direct* mode, every process directly performs many small I/O operations according to the offset/length of data chunks. The program measures the I/O bandwidth results as the average from 10 iterations. The same program is executed multiple times to filter out noticeable outliers due to other concurrent I/O loads.

3.2.1 Data Sieving

Data sieving is developed for small and non-contiguous I/O on a single process. It allocates an intermediate buffer, performs I/O on a contiguous region, and then extracts the needed data chunks. This can avoid many I/O operations on many small data chunks. With holes between the data chunks, the writes are implemented as read-modify-write to avoid overwriting the holes.

Figure 7 (L) shows the performance results with different process counts and different array sizes (128^3, 256^3, and 512^3). Several observations can be made about the strengths and weaknesses of data sieving on Jaguar. First, the I/O performance for small, non-contiguous I/O is very low on Jaguar as shown by the *direct* mode. Second, the performance of writes is much lower than that of reads because all processes are trying to gain exclusive write permission to a small region, therefore contending for exclusive write locks at the Lustre file system layer. Third, the I/O performance increases with an increasing array size for both *direct* and *data-sieving* modes; for the same array size, increasing process counts leads to poorer performance. These are expected because the average data size gets smaller for the latter but bigger for the former. Fourth, data sieving does improve the performance of reads as expected, but in contrast, it leads to performance degradation for writes. For both reads and writes, data sieving triggers the system call *flock()* to lock the I/O segment for maintaining data consistency. However, *flock()* has a costly implementation on the current catamount kernel for Jaguar. Because reads do not involve internal lock contention, the benefits of reading larger data chunks can offset the cost of a single lock system call. However, for the writes, while paying the cost of *flock()*, it does not benefit from larger data chunks because the internal file system lock contention–for exclusive write permission on the overlapped data chunks–remains the same, if not worse. For the writes to gain the same benefits as the reads, a file system optimization is needed to eliminate the internal write locks when a file region is already locked by a previous *flock()* call. The unusually high cost of *flock()* system call is only a side effect of the catamount system used for the Cray XT. It does not happen on regular Linux platforms.

In addition, we have measured the effects of data sieving buffer size for this program on Jaguar, and the results are shown in Figure 7 (R). By increasing the default data-sieve buffer size of 512KB to 8MB, the performance of writes can be improved by 4.5 times, and that of reads can be improved by 54%. Therefore, larger buffer sizes are beneficial for data sieving over Jaguar if this is possible by the application's memory requirement. Though 8MB is good for our test program, an appropriate size is mostly application dependent.



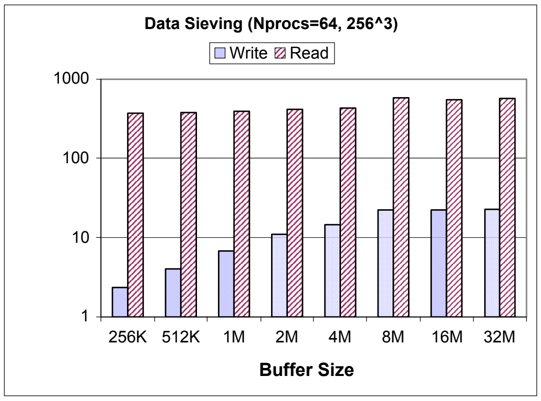


Figure 7: Data Sieving: (L) Comparisons to Direct Mode; (R) Increasing Buffer Size

3.2.2 Collective I/O

Collective I/O is a technique developed for small, non-contiguous I/O across a group of processes. The collective I/O protocol implements interleaved phases of *data exchange* and *file I/O* (reads/writes) on the linearly partitioned file domains. It is designed to aggregate small I/O operations into large I/O requests for better performance. With the same program used in the data sieving experiments, we characterized the benefits of collective I/O as Figure 8 shows. While increasing array sizes leads to better I/O bandwidth, the same array size with more processes–that is finer data chunks–does not lead to degraded I/O performance. These are expected because collective I/O aggregates small I/O requests for better I/O performance, and some variations in small data chunk sizes can also be smoothed out.

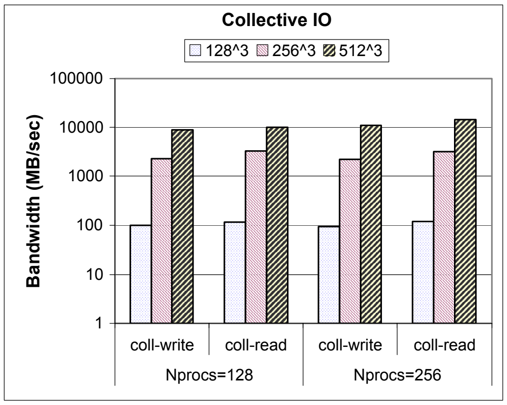


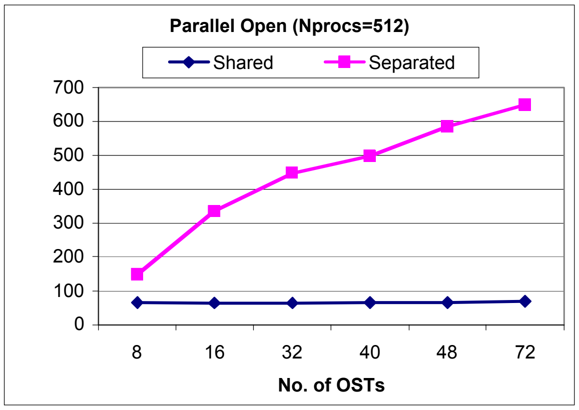
Figure 8: Benefits of Collective I/O

3.3 Parallel File Open

Besides the need for scalable bandwidth, there is also an increasing need on metadata scalability for large-scale applications. In particular, the problem of creating and opening files in parallel across tens of thousands of processes is a significant scalability challenge. For example, a team from DOE laboratories is working on extending the current POSIX IO interface with routines that can facilitate scalable implementation of parallel open [16].

Using the scr72b file system over Jaguar, we have measured the scalability of creating/opening files in parallel across an increasing number of OSTs. Figure 9 shows the time taken for opening a shared file across all processes and the same for opening one file per process. In our tests, all files are created with its first OST corresponds to the first LUN of the first couplet. As shown in the figure, for either mode, the time increases dramatically with the number of processes, as well as with the increasing number of OSTs. The one-file-per-proc mode is about an order of magnitude more costly than the shared-file mode.

As mentioned earlier, the results in Figure 9 were obtained with all files starting from the first OST of the file system. A common execution mode on Lustre is to give a parameter '-1' for the starting OST, which leaves this choice open for the operating system to make a decision dynamically. Instead of leaving the choice for the operating system, we emulated a dynamic, yet balanced distribution of the first OST based on the rank of processes. Using the OPAL library, we have measured the scalability of the dynamic method. Figure 10 shows the scalability with dynamic balanced distribution as compared to the earlier results when the first OST is statically determined. When all the OSTs are within a single DDN couplet, the static mode performs better than the dynamic mode. However, the dynamic mode is much more scalable when a file is striped cross multiple DDN couplets–more than 8 OSTs. Even when the files striped across all 72 OSTs in *scr72b*, the dynamic mode still reduces the parallel open time by 25%. These results suggest that, when files are striped less than 8 OSTs, there is a trade-off between using the static mode to achieve efficient parallel file open and the need of more OSTs for better bandwidth. For files with large strips, the default dynamic mode for the first OST is always beneficial.



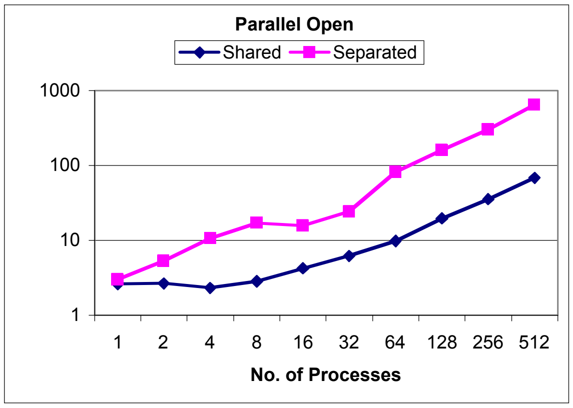


Figure 9: Scalability of Parallel File Open: (L) With increasing OSTs; (R) With increasing processes

4 Improving the Performance of Different Scientific I/O Patterns

In this section, we demonstrate several case studies in tuning and optimizing the parallel I/O performance in scientific applications and benchmarks.

4.1 Independent I/O Optimization: a Case Study with Combustion

Combustion simulation represents a critical domain of the scientific applications because of the great dependence of the world's energy production on combustion. Even with numerous efforts in renewable energy resources such as solar and hydrogen power, the combustion still produces 85% of the total energy for the world. In addition, the aggravating impacts of green house gases call for more efficient and cleaner combustion technologies. S3D, as a leading combustion application developed Sandia National Laboratories, performs the direct numerical simulation of turbulent combustion [26]. It is also an INCITE (Innovative and Novel Computational Impact on Theory and Experiment Program, Department of Energy, U.S.) application on Jaguar at Oak Ridge National Laboratory.

S3D is based on a high-order accurate, non-dissipative numerical scheme and solves the full compressible Navier-Stokes, total energy, species and mass continuity equations coupled with detailed chemistry. S3D is parallelized using a three-dimensional domain decomposition and MPI communication. Each MPI process is in charge of a piece of the three dimensional domain. All MPI processes have the same number of grid points and the same computational load. A complete run of a S3D simulation can take millions of computation hours to finish. To save its intermediate simulation results, the original S3D code writes its output from each process into individual files, periodically at every simulation step, which creates an enormous amount of data for the storage system, nearly a terabyte per hour across 24K processes with a checkpoint interval of every half an hour.

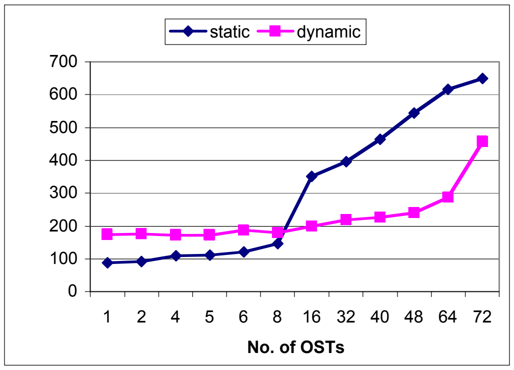
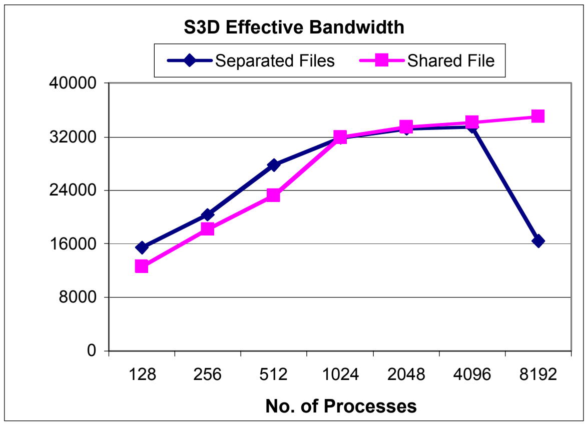


Figure 10: Parallel File Open with Different Selections on the First OST

In view of the scalable file open/creation with a shared file, we have implemented an optimization for S3D to do I/O through a shared file. Figure 11(L) shows the delivered I/O bandwidth to S3D at the application-level. The overall bandwidth is measured as the total amount of application data divided by the time to create the files and write the data. With the default mode of separated files, the delivered application-level bandwidth reaches a plateau at 33GB/sec, but drops down to 15GB/sec for 8192 processes. In contrast to the abrupt performance drop with a large number of processes, the shared file implementation achieves a sustained I/O bandwidth, up to 35GB/sec. We have measured the time to open the file(s), which Figure 11(R) illustrates. We observed that much of the performance drop is due to the increased time in opening/creating the separated files, in the default S3D implementation. Note that there is no global barrier between the file creation and output, so different processes are creating files and writing their output concurrently. Thus the time for file output is not strictly the subtraction of file creation time from the total time.



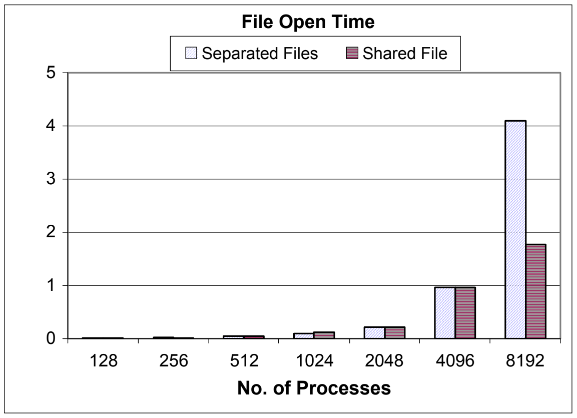


Figure 11: S3D performance: (L) I/O Bandwidth; (R) File Open Time

4.2 Collective I/O Tuning

Besides the benefits of I/O aggregating, the extended two-phase collective I/O protocol also provides a number of tuning parameters, including the buffer size for the I/O aggregation (*collective buffer size*) and the number of processes that are responsible for the aggregation (*the number of I/O aggregators*). We have selected the following programs to demonstrate the benefits of tuning collective I/O.

MPI-Tile-IO – MPI-Tile-IO [37] is an MPI-IO benchmark that tests the performance of tiled data accesses. In this application, data I/O is non-contiguous and issued in a single step using collective IO. It tests the performance of tiled access to a two-dimensional dense dataset, simulating the type of workload that exists in some visualization and scientific applications. In our experiments, each process renders a 1x1 tile with 1024x768 pixels. The size of each element is 32 bytes, leading to a file size of 48*N MB, where N is the number of processes.

BT-IO – NAS BT-IO [1] is an I/O benchmark that tests the output capability of NAS BT (Block-Tridiagonal) parallel benchmark. It was developed at NASA Ames Research Center. Its data set undergoes diagonal multi-partitioning and is distributed among MPI-processes. The data structures are represented as structured MPI datatypes and written to a file periodically. There are several different BT-IO implementations, which vary on how its file IO is carried out among all the processes. In the full mode, BT-IO performs collective I/O for file output; in the simple mode, all processes write out their small I/O requests.

Flash I/O – Flash is an application that simulates astrophysical thermonuclear flashes. It is developed in part at the University of Chicago by the DOE-supported ASC Alliance Center for Astrophysical Thermonuclear Flashes. The Flash I/O [32] benchmark is the I/O portion of the Flash program that measures the performance of its parallel HDF5 [32] output. The MPI-IO interface is used internally by the HDF5 library. Three different output files are produced in Flash I/O: a checkpoint file, a plotfile with centered data, and a plotfile with corner data. These files are written through the HDF5 [7] data format.

4.2.1 Number of IO Aggregators

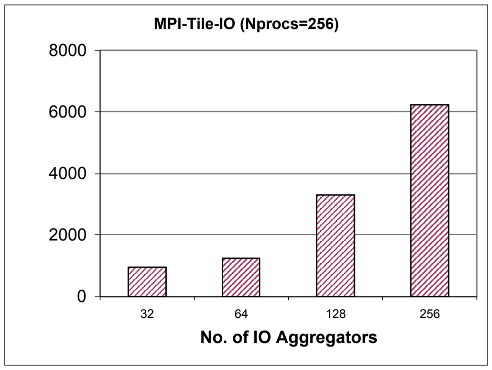


Figure 12: MPI-Tile-IO with Varying Number of I/O aggregators

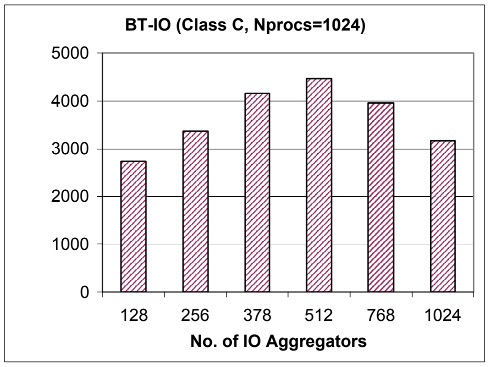


Figure 13: BT-IO with Varying Number of I/O Aggregators

Figures 12, 13 and 14 show the I/O performance of MPI-Tile-IO, BT-IO and Flash I/O, respectively, with different number of I/O aggregators. As shown in Figure 12, MPI-Tile-IO does gain performance from an increasing number of I/O aggregators. MPI-Tile-IO needs more I/O aggregators to store the data of 48MB per process (as introduced earlier). However, for BT-IO, the best I/O bandwidth is achieved with 576 processes, which represents a good balance of tradeoffs between the number of processes, the granularity of file domains, and the associated communication cost for aggregation. Flash I/O was executed with two different block sizes 16- and 32-bytes. For an execution of Flash I/O with 2048 processes, 256 aggregators are optimal when the block size is 16bytes; while 512 or 1024 is sufficient when block size is 32bytes.

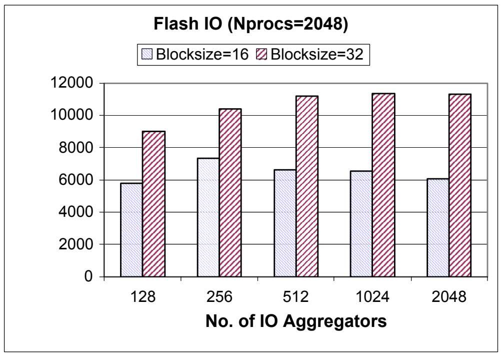


Figure 14: Flash I/O with Varying Number of I/O aggregators

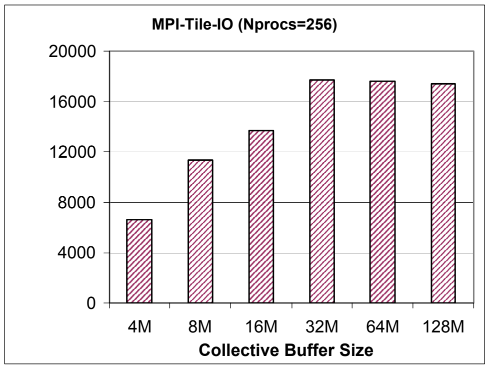


Figure 15: MPI-Tile-IO with Increasing Collective Buffer

4.2.2 Collective Buffer Size

Figures 15, 16, and 17 show the I/O performance of MPI-Tile-IO, BT-IO and Flash I/O, respectively, with varying sizes of collective buffer. For Flash I/O, the performance is measured on the checkpoint file with the data block size to be 32bytes. A comparison to non-collective I/O mode is also shown for Flash I/O in Figure 17. Both MPI-Tile-IO and Flash I/O benefit from increased collective buffer, while 32MB appears to be the optimal for them. However, BT-IO benefits only slightly from increased collective buffer. This is because the I/O of BT-IO is carried out in 40 different iterations. For BT-IO Class C, there is only about 168MB of I/O data in each step. This results in a file domain less than 4MB for each I/O aggregator. Therefore, a very large collective buffer does not benefit BT-IO.

5 Related Work and Conclusions

There is a rich set of literature on the performance characterization of HPC systems. Many studies were carried out to study the inter-process communication, the peak computing power, and the comparisons amongst different interconnect technologies or different large-scale deployments. For example, [9, 21, 22, 25, 35, 36] and [29] have studied the communication characteristics, the performance tradeoffs as well as the comparisons among different interconnect technologies, including Myrinet, Quadrics, InfiniBand, and 1/10Gigabit Ethernet. [5, 6, 34, 35] and [6] have studied the system performance of massive parallel systems including the Cray XT and Blue Gene/L. However, these studies typically do not address the I/O performance of the system. The work most closely related to ours is [14, 19, 38]. Particularly, [14] and [19] report the I/O performance of two popular Cray XT platforms, Red Storm and Jaguar, respectively. Both of them have studied more on the strong scaling of the peak I/O performance. In contrast, we focus more on varying I/O patterns, as well as collective I/O. The precursor of our work has been presented in [38].

In this paper, we have extensively characterized the parallel I/O performance on the Jaguar supercomputer. Our characterization covered the performance and scalability of the individual storage units, as well as the entire system. We have examined the best stripe sizes over Jaguar, and showed that the file distribution pattern across the DDN storage couplets can dramatically impact the aggregated performance. In addition, we have also examined the scalability of metadata- and data-intensive operations. Our results have demonstrated that, for parallel file open, the shared file mode has the best scalability compared to the separated file mode. Moreover, we have investigated the performance impacts of parallel I/O techniques for handling small and non-contiguous I/O, including data sieving and collective I/O. We documented that, with overlapped file segments, data sieving from concurrent processes can lead to performance degradations. Increasing the size of data sieving buffer can improve the performance, but the performance of writes is hindered by the internal lock contentions at the Lustre file system layer.

Finally, we have demonstrated how to leverage the insights from our characterizations to tune and optimize the I/O performance of scientific benchmarks and applications. We have shown that the I/O scalability of S3D combustion application over 8192 processes can be sustained through a simple, yet very beneficial optimization. We also have illustrated how collective I/O tuning parameters can impact the performance of different scientific I/O benchmarks. In particular, we have shown that the performance of Flash I/O can be improved by 34% with careful tuning of the collective I/O parameters.

Acknowledgments

This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725. We also thank Dr. Rajeev Thakur for the 3-D array collective I/O test benchmark.

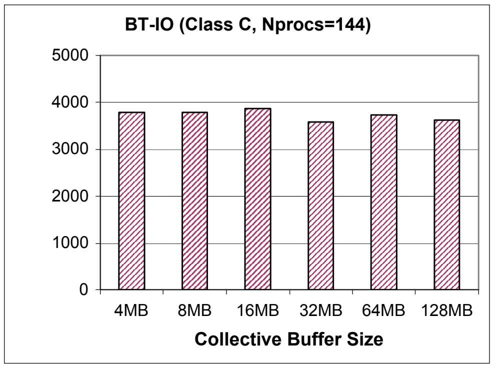


Figure 16: BT-IO with Increasing Collective Buffer

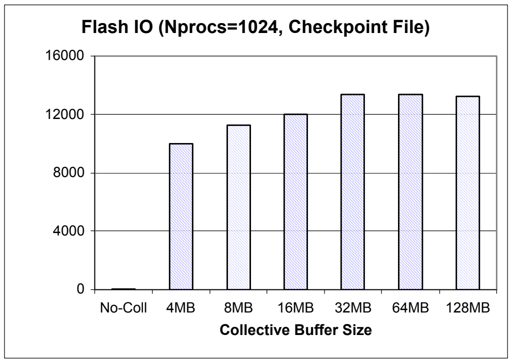


Figure 17: Flash I/O with Increasing Collective Buffer

6 References

[1] FLASH I/O Benchmark Routine -- Parallel HDF 5,
[2] IOR Benchmark, http://www.llnl.gov/asci/purple/benchmarks/limited/ior.
[3] TOP 500 Supercomputers, http://www.top500.org/.
[4] Simulation and Modeling at the Exascale for Energy, Ecological Sustainability and Global Security (E3SGS), Town Hall Meetings, http://hpcrd.lbl.gov/E3SGS/main.html, 2007.
[5] N. R. Adiga, G. Almasi, G. S. Almasi, et al., An overview of the BlueGene/L Supercomputer, in Proceedings of the 2002 ACM/IEEE conference on Supercomputing Baltimore, Maryland: IEEE Computer Society Press, 2002.
[6] S. R. Alam, R. F. Barrett, M. R. Fahey, J. A. Kuehn, J. M. Larkin, R. Sankaran, and P. H. Worley, Cray XT4: An Early Evaluation for PetaScale Scientific Simulation, in *ACM/IEEE conference on High Performance Networking and Computing (SC07)*, Reno, NV, 2007.
[7] N. J. Boden, D. Cohen, and others, Myrinet: A Gigabit-per-Second Local Area Network, *IEEE Micro,* pp. 29-35, Feb 1995.
[8] R. Brightwell, W. Camp, B. Cole, E. DeBenedictis, R. Leland, J. Tomkins, and A. B. MacCabe, Architectural specification for massively parallel computers: an

experience and measurement-based approach: Research Articles, *Concurr. Comput. : Pract. Exper.,* vol. 17, pp. 1271-1316, 2005.

[9] R. Brightwell, D. Dourfler, and K. D. Underwood, A Comparison of 4X InfiniBand and Quadrics Elan-4 Technology, in *Proceedings of Cluster Computing, '04*, San Diego, California, 2004.

[10] R. Brightwell, R. Riesen, B. Lawry, and A. B. Maccabe, Portals 3.0: Protocol Building Blocks for Low Overhead Communication, in *Proceedings of the 2002 Workshop on Communication Architecture for Clusters (CAC)*, 2002.

[11] Cluster File System, Lustre: A Scalable, High Performance File System,

[12] DataDirect Network, S2A Nearline SATA Solution Overview, 2005.

[13] DDN, Products: S2A9550,

[14] M. Fahey, J. Larkin, and J. Adams, I/O Performance on a Massively Parallel Cray XT3/XT4, in *22nd IEEE International Parallel & Distributed Processing Symposium (IPDPS'08)*, Miami, FL, 2008.

[15] D. Filliger, Personal Communication, 2007.

[16] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen, Direct Numerical Simiulation of Turbulent Combustion: Fundamental Insights towards predictive Models, *Journal of Physics: Conference Series,* pp. 65-79, 2005.

[17] High End Computing Extenstions Working Group (HECEWG), Manpage - openg (group open), http://www.opengroup.org/platform/hecewg/uploads/40/10899/openg.pdf.

[18] HPCWire, NSB Approves Funds for Petascale Computing Systems, http://www.hpcwire.com/hpc/1715754.html, 2007.

[19] J. Laros, L. Ward, R. Klundt, S. Kelly, J. Tomkins, and B. Kellogg, Red Storm IO Performance Analysis, in *Cluster*, Austin, TX, 2007.

[20] Z. Lin, S. Ethier, T. S. Hahm, and W. M. Tang, Size Scaling of Turbulent Transport in Magnetically Confined Plasmas, *Phys. Rev. Lett.,* vol. 88, p. 195004, Apr 2002.

[21] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. P. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda, Performance Comparison of MPI implementations over Infiniband, Myrinet and Quadrics, in *Proceedings of Supercomputing '03 (SC '03)*, 2003.

[22] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. P. Kini, P. Wyckoff, and D. K. Panda, Micro-Benchmark Performance Comparison of High-Speed Cluster Interconnects, *IEEE Micro,* vol. 24, pp. 42-51, January-February 2004.

[23] Los Alamos National Laboratory, High-Performance Computing: RoadRunner, http://www.lanl.gov/roadrunner/, 2006.

[24] K. Pedretti, R. Brightwell, and J. Williams, Cplant" Runtime System Support for Multi-Processor and Heterogeneous Compute Nodes, in Proceedings of the IEEE International Conference on Cluster Computing: IEEE Computer Society, 2002.

[25] F. Petrini, W.-c. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, The Quadrics network: High-performance clustering technology, *IEEE Micro,* vol. 22, pp. 46-57, January/February 2002.

[26] R. B. Ross, Parallel I/O Benchmarking Consortium,

[27] H. Shan and J. Shalf, Using IOR to Analyze the I/O Performance of XT3, in *Cray User Group (CUG)*, Seattle, WA, 2007.

[28] R. Thakur and A. Choudhary, An Extended Two-Phase Method for Accessing Sections of Out-of-Core Arrays, *Scientific Programming,* vol. 5, pp. 301-317, Winter 1996.

[29] R. Thakur, W. Gropp, and E. Lusk, Data Sieving and Collective I/O in ROMIO, in *Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, 1999, pp. 182-189.

[30] R. Thakur, W. Gropp, and E. Lusk, On Implementing MPI-IO Portably and with High Performance, in *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, 1999, pp. 23-32.

[31] R. Thakur, W. Gropp, and E. Lusk, Optimizing noncontiguous accesses in MPI– IO, *Parallel Computing,* vol. 28, pp. 83-105, 2002.

[32] The National Center for SuperComputing, HDF5 Home Page,

[33] G. M. Timothy, S. David, and R. W. Stephen, A TeraFLOP Supercomputer in 1996: The ASCI TFLOP System, in Proceedings of the 10th International Parallel Processing Symposium: IEEE Computer Society, 1996.

[34] J. S. Vetter, S. R. Alam, T. H. Dunigan, Jr.,, M. R. Fahey, P. C. Roth, and P. H. Worley, Early Evaluation of the Cray XT3, in *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Rhodes Island, Greece, 2006.

[35] J. S. Vetter and F. Mueller, Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures, in *IPDPS*, 2002.

[36] K. Voruganti and P. Sarkar, An Analysis of Three Gigabit Networking Protocols for Stroage Area Networks, in *Proceedings of International Conference on Performance, Computing, and Communications*, 2001.

[37] P. Wong and R. F. Van der Wijngaart, NAS Parallel Benchmarks I/O Version 2.4, NASA Advanced Supercomputing (NAS) Division NAS-03-002, 2002.

[38] W. Yu, H. S. Oral, J. Vetter, and R. Barrett, Efficiency Evaluation of Cray XT Parallel IO Stack, in *Cray User Group Meeting (CUG 2007*, Seattle, Washington, 2007.

[39] W. Yu, J. S. Vetter, and R. S. Canon, OPAL: An Open-Source MPI-IO Library over Cray XT, in *International Workshop on Storage Network Architecture and Parallel I/O (SNAPI'07)*, San Diego, CA, 2007.