Efficiency Evaluation of Cray XT Parallel IO Stack

Weikuan Yu[1,2]**, Sarp Oral**[2]**, Jeffrey Vetter**[1]**, Richard Barrett**[1,2]

Computer Science and Mathematics Division[1]
National Center for Computational Science Division[2]
Oak Ridge National Laboratory
One Bethel Valley Road, Oak Ridge, TN 37831
{wyu,oralhs,vetter,rbarrett}@ornl.gov

ABSTRACT: PetaScale computing platforms need to be coupled with efficient IO subsystems that can deliver commensurate IO throughput to scientific applications. In order to gain insights into the deliverable IO efficiency on the Cray XT platform at ORNL, this paper presents an in-depth efficiency evaluation of its parallel IO software stack. Our evaluation covers the performance of a variety of parallel IO interfaces, including POSIX IO, MPI-IO, and HDF5. Moreover, we describe several tuning parameters for these interfaces and present their effectiveness in enhancing the IO efficiency.

KEYWORDS: Cray XT, Parallel IO, Efficiency, ROMIO, Lustre

1 Introduction

High Performance Computing (HPC) is soon approaching the era of PetaScale computing. Massive Parallel Processing (MPP) platforms such as Lawrence Livermore National Lab's (LLNL) BlueGene/L [4] and Oak Ridge National Lab's (ORNL) XT4 (Jaguar) [18] have already reached 100s of TFLOPS. Large-scale data-intensive scientific applications on these platforms demand increasingly higher IO throughputs as well as higher computational capacity. Correspondingly, scalable parallel IO needs to be available for these applications to perform well. In fact, aggregated IO throughput ranging from 10-100 GB/s has been reported for Red Storm and Thunderbird at Sandia National Laboratory, and Tera-10 at CEA in Europe [6], among many others. However, such performance marks are the systems' peak IO bandwidth, which are usually measured with best suitable parameters for the IO subsystem, while what matters the most to the end users is the deliverable IO bandwidth to their respective scientific applications.

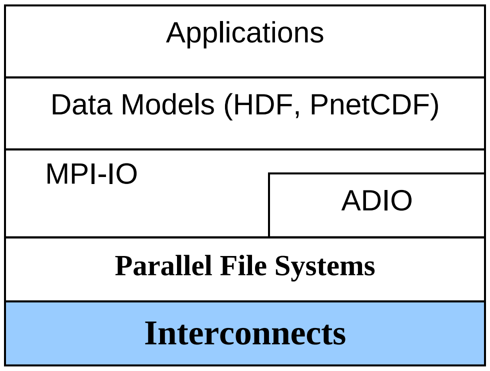

Figure 1 Parallel IO Software Architecutre

For the reasons of portability and convenient data representation, HPC systems are often deployed with a varying depth of software stacks such as the basic POSIX syscall interfaces, MPI-IO and HDF5 as shown in Figure 1. Collectively, these layers provide portable abstractions for IO accesses. At the top end, scientific applications perform IO through middleware libraries such as Parallel netCDF [9], HDF [23, 24] and MPI-IO [22], often cooperatively among their processes. Parallel file systems, towards the bottom of the stack, directly serve IO requests by striping file extents and/or IO blocks across multiple storage devices. Obtaining a good collective-IO performance across many processes on top of these software layers is a complex task. It requires not only awareness of the processes' collective data access patterns, but also a thorough understanding of the entire software stack and in particular, the behavior of underlying file systems.

MPI-IO [15] is the IO interface for the popular message passing parallel programming model. It is also a frequently used IO programming interface in scientific applications. As shown in Figure 1, MPI-IO is layered on top of file systems. ADIO [19] is the abstract IO interface of MPI-IO, which is typically implemented with specializations for specific file systems. Together, these programming stacks offer crucial avenues for efficient storage accesses. Numerous techniques have been investigated and implemented to improve the scalability of MPI-IO data operations, such as two-phase IO [20], data sieving [21], and data shipping [13]. Along with these techniques, there is an assortment of tuning parameters for improving the efficiency of different IO access patterns.

All these different IO possibilities raise really important questions to both the application scientists and system designers: what IO interface is appropriate to choose from the portfolio? How to gain the best IO efficiency for a given IO interface? In this paper, we tackle on these questions and present our recent evaluation of IO efficiency of parallel IO stacks over Jaguar. Our evaluation covers the performance of a set of common parallel IO interfaces, including POSIX IO, MPI-IO, and HDF5. Moreover, we describe a variety of tuning parameters and their effectiveness in enhancing the IO efficiency for these interfaces.

The rest of the paper is organized as follows. In the next section, we provide an overview of Cray XT4 and its IO software stack, as well as the configuration of its IO subsystem. Following that, we analyze the IO efficiency for the POSIX layer. In Section 4, we investigate the MPI-IO layer performance while Section 5 presents examples in tuning the IO kernels of a scientific parallel IO benchmark, Flash IO [1]. Section 6 concludes the paper.

2 Cray XT4 and its IO Subsystem

In this section, we provide an overview of Cray XT4 and the IO subsystem.

2.1 An Overview of Cray XT4

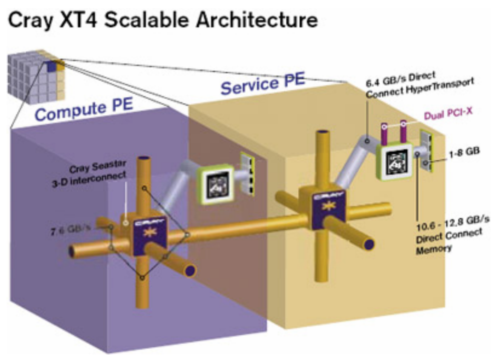
Cray XT4 Scalable Architecture
Compute PE
Service PE
Cray Seastar 3-D interconnect
7.6 GB/s
6.4 GB/s Direct Connect HyperTransport
Dual PCI-X
1-8 GB
10.6 - 12.8 GB/s Direct Connect Memory

Figure 2: Cray XT4 system architecture (courtesy of Cray Inc.)

Cray XT3 and Cray XT4 represent generations of Massive Parallel Processing platforms 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 the Cray XT3/4 systems is a Processing Element (PE), as shown in Figure 1. Each PE is comprised of one AMD processor (single, dual or quad core), along with its own memory, integrated memory controller, HyperTransport links, and a dedicated communication chip, SeaStar or SeaStar2. The HyperTransport interface enables high bandwidth direct connections between the processor, the memory and the SeaStar/SeaStar2 chip. Jaguar has recently combined its Cray XT3 and Cray XT4 platforms into a single 119TF system.

Cray XT4 inherits its system software from a sequence of systems developed at Sandia National Laboratories and University of New Mexico: ASCI Red [17], the CPlant [9, 10], and Red Storm [11]. Two different operating systems are running two different types of processing elements: Catamount on compute PEs and Linux on service PEs. The compute PEs run the Catamount lightweight operating system. The Catamount kernel runs only one single-threaded process and does not support demand-paged virtual memory. Service PEs (i.e., login, IO, network, and system PEs) run Linux to provide a user-familiar environment for application development and for hosting system and performance tools. The Portals [5] data movement layer is used for flexible, low-overhead inter-node communication. It delivers data from a sending process' user space to the receiving process' user space without kernel buffering.

2.2 Cray XT4 Parallel IO Software Stack

The IO subsystem of Cray XT4 is provided through Lustre file system [7]. Lustre offers a very scalable IO subsystem, supporting IO services to tens of thousands of concurrent clients, with an aggregated IO rate of 1000s GB/sec. Lustre is a distributed parallel file system and presents a POSIX interface to its clients with parallel access capabilities to the shared objects. Lustre is composed of four main parts, namely, Metadata Server (MDS) providing metadata services; Object Storage Servers (OSSs) managing and providing access to underlying Object Storage Targets (OSTs); OSTs controlling and providing access to the actual physical block devices and data, and client(s) that access and use the data. The OSS/OST compartmentalization in Lustre provides increased scalability for large-scale computing platforms. On Cray XT systems, IO PEs are configured as Lustre servers that provide IO and metadata services, each attached with high performance storage targets, such as DDN S2A9550 [8].

There are two types of IO clients to the IO services over XT4: regular Linux clients from service PEs and Catamount clients from compute PEs. Clients from the service PEs, including the login PEs, network and system PEs, obtain their IO services through regular Linux Lustre clients. In contrast, on the compute PEs, IO services are only available through an interface called *liblustre*, which transfers Lustre IO requests through Portals communication library to the IO PEs. Figure 3 shows the detail layering of the IO interfaces over Cray XT4. Moreover, parallel processes from compute PEs can perform IO either by directly invoking POSIX read/write, or by calling through a MPI-IO library. Their IO system calls from applications on compute PEs, such as read/write, are converted to liblustre routines through a preloaded SYSIO library [13], either at compile time or run-time. Moreover, Cray provides a proprietary MPI-IO implementation. This package offers parallel IO through a liblustre-specific ADIO [14] implementation of the MPI-IO [15] interface, denoted as AD_Sysio in the figure.

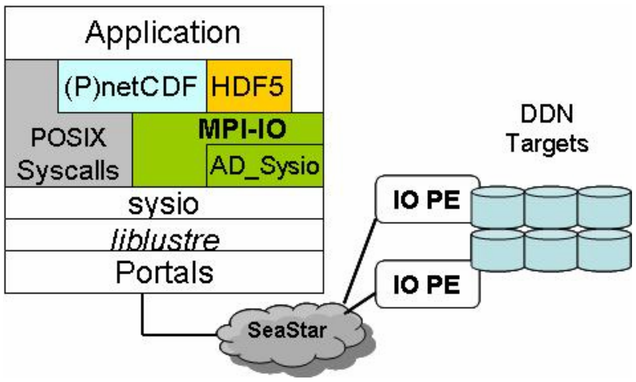

Figure 3 Cray XT4 Parallel IO Stack

2.3 Configuration of the Jaguar IO Subsystem

The ORNL's XT4 (Jaguar) serves 3 Lustre file systems; two in 150 TB and one 300 TB capacity. Each file system has its own dedicated XT4 MDS service node. Jaguar currently has 8 available MDS service nodes. In total 72 XT4 service nodes are assigned as OSSs for these 3 file systems. Each OSS node is configured with 4 OSTs. Two of these OSTs are serving LUNs for the 300 TB file system, while remaining two OSTs serve LUNs for two 150 TB file systems. Details can be seen in Figure 3. Today, ORNL Jaguar provides roughly 45 GB/s block IO bandwidth via the 300 TB Lustre file system.

As mentioned earlier, DDN 9550 is the ORNL's current choice for the backend block devices. There are 18 racks or in other words 18 couplets of DDN 9550s assigned for the block IO. A separate rack is assigned for the current and future metadata needs of Jaguar. Jaguar MDS/OSS nodes and the DDN 9550s are connected direct Fibre Channel (FC) links. Each MDS/OSS XT4 server node is equipped with two 1-port 4 Gbps FC HBAs. DDN 9550 couplets are also provided with 4 Gbps FC ports. DDN 9550s LUNs are configured such that, each LUN spans two tiers of disks as suggested by CFS for the optimal performance as can be seen in Table 1. Each 9550 DDN couplet has a capacity of 36 TB. The write-back cache is set to 1MB on each controller. Below table shows the LUN/Tier configuration for Jaguar attached DDN 9550s. Each LUN has a capacity of 2 TBs and 4 kB block size.

Table 1: ORNL Jaguar's LUN/Tier mapping

LUN	Label	Owner	Tiers	Tier list
0	LUN0	1	2	1 2
1	LUN1	1	2	3 4
2	LUN2	1	2	5 6
3	LUN3	1	2	7 8
4	LUN4	2	2	9 10
5	LUN5	2	2	11 12
6	LUN6	2	2	13 14
7	LUN7	2	2	15 16
8	LUN8	1	2	1 2
9	LUN9	1	2	3 4
10	LUN10	1	2	5 6
11	LUN11	1	2	7 8
12	LUN12	2	2	9 10
13	LUN13	2	2	11 12
14	LUN14	2	2	13 14
15	LUN15	2	2	15 16

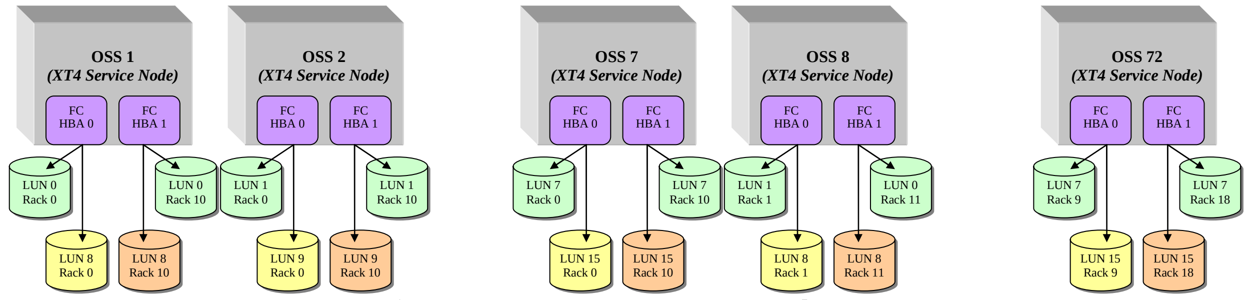

Figure 3: ORNL Jaguar's OSS/LUN layout

3 Parallel IO with the POSIX Read/Write

POSIX IO API, although developed primarily for single node UNIX environments, is still the primary access mechanism for lower layer IO stacks. Scientific computing applications not only benefit from a widely implemented standard IO API that is available on high-performance platforms but also the chance of running legacy code with minor modification if any needed at all. On the other hand, these benefits come with a performance and orchestration penalty over large-scale parallel scientific computational applications. In this section we explore various performance aspects of POSIX IO access for the ORNL's Jaguar system.

3.1 Hotspot tests

As described earlier, Lustre is a parallel distributed file system with multiple OSTs. Each OST holds objects as the smallest storage unit. While accessing objects striped over multiple OSTs theoretically increase the overall IO performance, it is also possible for multiple clients to access a single object located on a single OST. Such access pattern might be beneficial for certain parallel computing applications and data distribution patterns but also has the potential to create a hotspot. In this set of tests we analyze the Cray XT4's IO subsystem response for such an access pattern using the LLNL's IOR block IO benchmark [2].

For this test one of the smaller Lustre systems (150 TB) is used. The file is located on a single OST and a varying number of clients are concurrently accessing it (all read or all write in an alternating fashion). The transfer size per client is set to vary between 4 MB and 64 MB. Figure 4 illustrates the results obtained. The maximum read or write performance is limited to 400 MB/s, which is aligned with the theoretical limit per OST (Fibre Channel 4Gbps adaptors). As can be seen, altering the transfer size does not change overall performance for writes drastically, with increasing number of processes. A similar trend is observed for the reads. However, comparing the read and write performance, it is clear that writes are much more graceful to hotspot pressure compared to the reads.

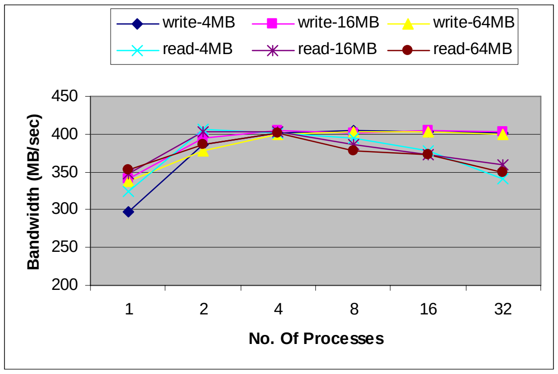

Figure 4: POSIX read/write Hotspot tests

3.2 Read/Write Operation Throughput Tests

Using the same benchmark from LLNL, we also tested the access throughput for a fixed sized object. The transfer size per client is varied between 64 bytes for small message access and 2GB for large messages. 1000 consecutive iterations of read/write operations per client are timed. One client process is assigned to access a single OST. Results can be seen in Figure 5. While we obtained faster reads for very small messages (smaller than 2KB), for transfer sizes beyond 2KB writes outperformed the reads.

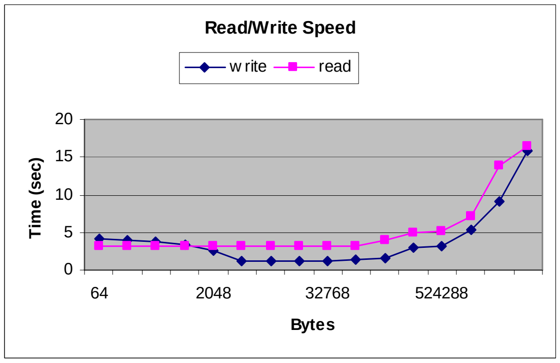

Figure 5: POSIX read/write Speed tests

3.3 Bandwidth Tests

One other interesting aspect to analyze using POSIX IO semantics is the shared and individual file per process access bandwidth performance. Using IOR, we set the file size to 512 MB and the transfer size per client to 64 MB. We also striped the test file(s) over various numbers of stripes (OSTs), namely, 31, 32, 59, and 64. Results can be seen in Figure 6, for shared file and separate file per process, respectively. Overall, shared file performs better than separated file. This can be attributed to increased write-back and read-ahead cache locality as well as more efficient physical disk allocation for the shared file access compared to the separate file per process case. Furthermore, prime number stripe counts help the read performance while hurting the writes. Finally, it can be seen clearly that increasing the number of writer or reader processes might have an adverse affect on the aggregate performance.

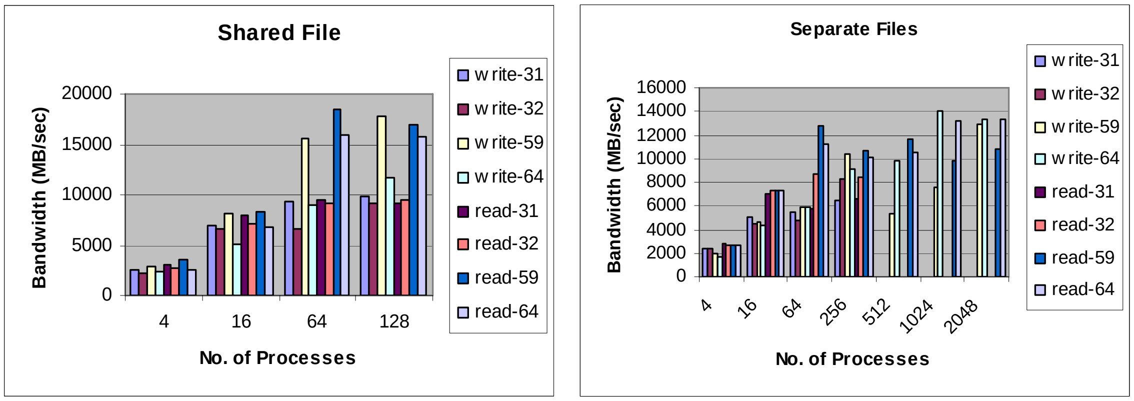

Figure 6: POSIX read/write bandwidth tests: (L) shared file; and (R) separate files

4 MPI-IO

MPI-IO augments the MPI standard by providing a standard parallel IO interface. MPI-IO collective operations allow coalescing for small message requests which means merging small data access operations in a single large one with the potential increase in performance and IO efficiency providing extensive C/C++ and Fortran language bindings. The readers are urged to access the MPI-2 documentation [3] for more details about MPI-IO. As most of the modern MPP systems, Cray XT4 is provided an MPI-IO library.

Due to its ease of use and efficiency, MPI-IO is popular among the parallel application developers. In this section, we investigate the MPI-IO efficiency of ORNL's Jaguar system. The analysis is performed for both individual MPI-IO and collective MPI-IO access.

4.1 Independent MPI-IO access

Using the IOR benchmark described in the previous section, we measured the shared file and file per process IO performance for read and write access. The resultant file size is set to 512 MB and the transfer size per process is 64 MB. Figure 7 shows the performance results. As can be seen, the increasing number of reader/writer process has the potential effect of impeding the overall performance. Furthermore, the prime number stripe counts help for both read and write access for smaller process counts (less than or equal to 512). Writes with large number of processes may also benefit prime number of stripes.

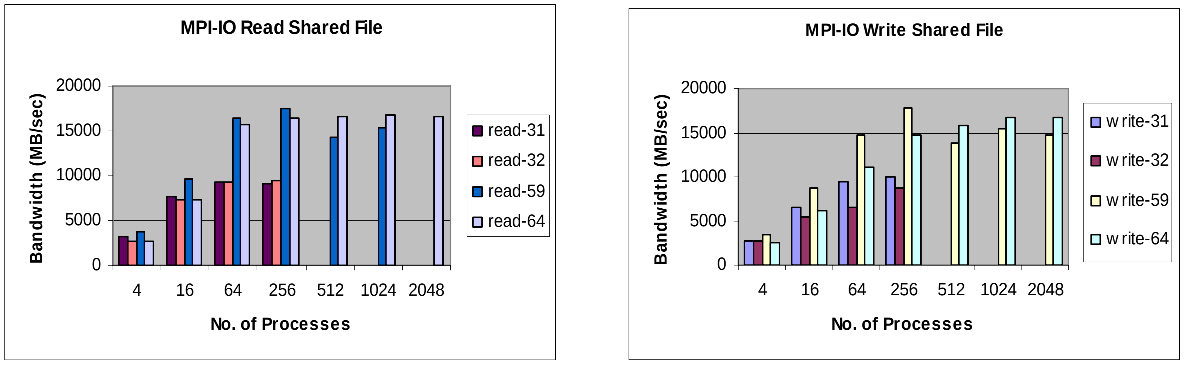

Figure 7: Independent MPI-IO, shared file performance: (L) Read; and (R) Write

For the file per process case, the performance test results are provided in Figure 8. For this access pattern, unlike the results from the shared file access, the prime number stripe counts do not help overall performance. More importantly, the performance with file per process case is worse than the shared file access pattern.

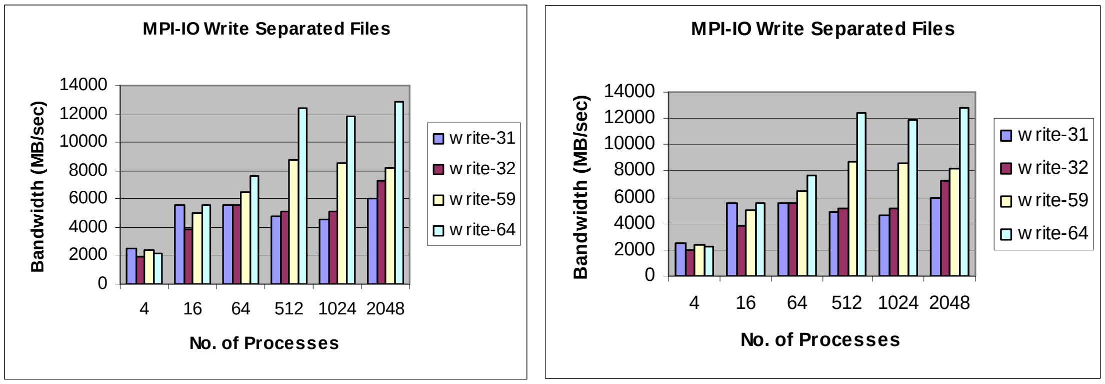

Figure 8: Independent MPI-IO file per process performance: (L) Read; and (R) Write

4.2 Collective MPI-IO access

For evaluating the collective MPI-IO access efficiency we used the MPI-Tile-IO benchmark. MPI-Tile-IO [12] is a tile reading MPI-IO application. It tests the performance of tiled access to a two dimensional dense dataset, simulating the type of workload that exists in some visualization and numerical applications. In our tests, interleaved tiled reads/writes are performed to a file with 64 stripes on one of the smaller 150TB Lustre file systems over Jaguar. Figure 9 shows the results. As can be seen collective write performs better than collective read. Moreover, our analysis clearly shows that collective writes needs a large collective buffer and 32 MB is sufficient for our test environment. This behavior can be seen in Figure 10.

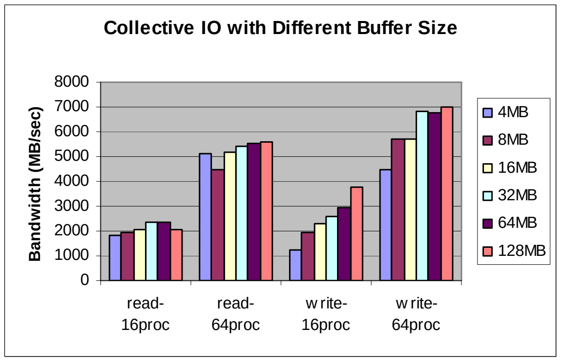

Figure 9: Collective MPI-Tile-IO with varying collective buffer sizes

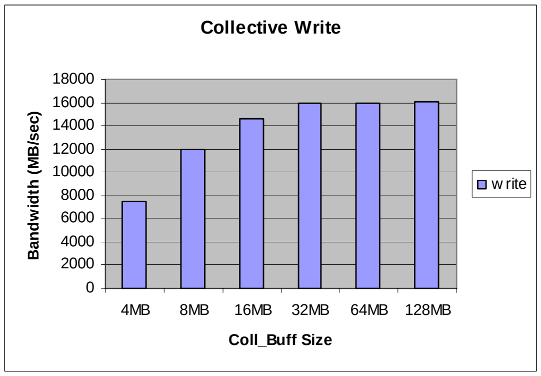

Figure 10: Collective MPI-Tile-IO with varying collective buffer sizes

5 Sample Application Tuning with Flash IO

As stated earlier, benchmarking the IO subsystem gives the system architects an idea how the system will respond under certain conditions. Oftentimes, these are obtained through synthetic benchmarks and reveal the subsystems peak response. However, under normal production workload, a user might experience an entirely different IO subsystem behavior. Although it is not perfectly possible to foresee all possible scenarios and test and prepare for them, it is possible to mimic these workloads using micro kernel codes. These codes are mostly excerpts from actual scientific applications and they are targeted to stimulate and analyze a certain subsystem's behavior under close to realistic workloads or access patterns. Flash IO is examples of such benchmarks. We describe and explain our analysis of ORNL's Jaguar IO subsystem using Flash IO.

The Flash IO benchmark is designed to analyze the IO subsystem performance using the IO kernel of the FLASH application. The Flash IO is implemented on top of HDF5 [16]. It recreates the primary data structures in Flash and produces a checkpoint file, a plotfile with centered data, and a plotfile with corner data. It uses the same IO routines used by FLASH. Our Flash IO performance results with respect to varying buffer sizes are presented in Figure 13. As can be seen, Flash IO benefits from varying collective buffer sizes, thus it is important to enable collective IO buffering. Our analysis reveal that the optimal performance can be achieved using a 32 MB collective buffer size.

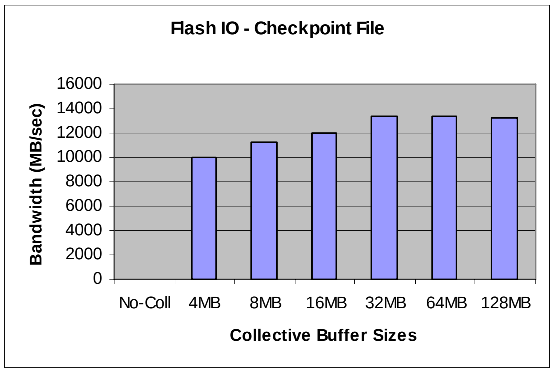

Figure 13: Flash IO results for varying collective buffer sizes.

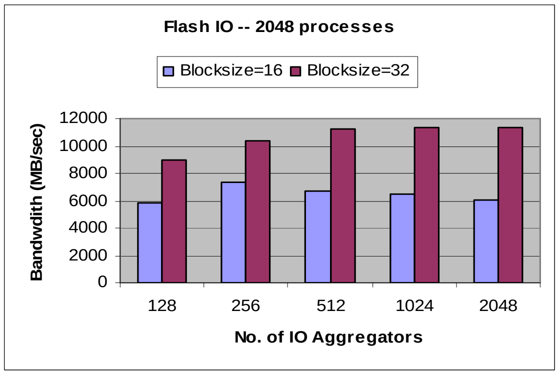

Figure 14: Flash IO results for varying number of IO aggregators.

Figure 14 presents the results for Flash IO test runs with respect to varying number of IO aggregators. This set of tests reveal that, the number of IO aggregators in Flash IO needs to be tuned for optimal performance and for our given test setup, 256 or 512 aggregators are optimal for would be sufficient, particularly for Flash IO runs with small blocksizes.

6 Conclusions

To gain insights into the deliverable IO efficiency on the Cray XT4 platform at ORNL, this paper presents an in-depth evaluation of its parallel IO software stack. Our evaluation covers the performance of a variety of parallel IO interfaces, including POSIX IO, MPI-IO[15], and HDF5 [16]. Through our evaluation, we have shown that it is important to make salient choice on the number of stripes for parallel IO over Cray XT. In addition, users need to be aware of the speed variance of IO accesses with different IO request sizes. Moreover, we have demonstrated that it is beneficial to limit IO accesses to a moderate number of processes, typically less than 512 processes.

Furthermore, we have described a variety of tuning parameters for the parallel IO stack and have demonstrated their effectiveness in enhancing the IO efficiency of a common scientific IO benchmark, Flash IO [1]. Our results suggest that by tuning the collective buffer size and the number of IO aggregators, the achievable bandwidth of Flash IO can be increased by as much as 20%.

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.

References

[1] "FLASH I/O Benchmark Routine -- Parallel HDF 5," http://www.ucolick.org/~zingale/flash_benchmark_io/.
[2] "IOR Benchmark." http://www.llnl.gov/asci/purple/benchmarks/limited/ior.
[3] "MPI-2: Extensions to the Message-Passing Interface," 1997.
[4] N. R. Adiga, G. Almasi, G. S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, A. A. Bright, J. Brunheroto, C. Ca, caval, J. Casta, os, W. Chan, L. Ceze, P. Coteus, S. Chatterjee, D. Chen, G. Chiu, T. M. Cipolla, P. Crumley, K. M. Desai, A. Deutsch, T. Domany, M. B. Dombrowa, W. Donath, M. Eleftheriou, C. Erway, J. Esch, B. Fitch, J. Gagliano, A. Gara, R. Garg, R. Germain, M. E. Giampapa, B. Gopalsamy, J. Gunnels, M. Gupta, F. Gustavson, S. Hall, R. A. Haring, D. Heidel, P. Heidelberger, L. M. Herger, D. Hoenicke, R. D. Jackson, T. Jamal-Eddine, G. V. Kopcsay, E. Krevat, M. P. Kurhekar, A. P. Lanzetta, D. Lieber, L. K. Liu, M. Lu, M. Mendell, A. Misra, Y. Moatti, L. Mok, J. E. Moreira, B. J. Nathanson, M. Newton, M. Ohmacht, A. Oliner, V. Pandit, R. B. Pudota, R. Rand, R. Regan, B. Rubin, A. Ruehli, S. Rus, R. K. Sahoo, A. Sanomiya, E. Schenfeld, M. Sharma, E. Shmueli, S. Singh, P. Song, V. Srinivasan, B. D. Steinmacher-Burow, K. Strauss, C. Surovic, R. Swetz, T. Takken, R. B. Tremaine, M. Tsao, A. R. Umamaheshwaran, P. Verma, P. Vranas, T. J. C. Ward, M. Wazlowski, W. Barrett, C. Engel, B. Drehmel, B. Hilgart, D. Hill, F. Kasemkhani, D. Krolak, 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.
[5] R. Brightwell, R. Riesen, B. Lawry, and A. B. Maccabe, "Portals 3.0: Protocol Building Blocks for Low Overhead Communication," Proceedings of the 2002 Workshop on Communication Architecture for Clusters (CAC), 2002.
[6] Bull Direct, "French Atomic Energy Authority (CEA) takes delivery of Tera-10," 2006.
[7] Cluster File System, "Lustre: A Scalable, High Performance File System."
[8] DDN, "Products: S2A9550," http://www.datadirectnet.com/s2a9550.html, 2007.
[9] P. Kevin, B. Ron, and W. Joshua, "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.
[10] B. Ron, F. Lee Ann, S. G. David, H. Tramm, L. Mike, B. M. Arthur, and R. Rolf, "Massively parallel computing using commodity components," *Parallel Comput.*, vol. 26, pp. 243-266, 2000.
[11] B. Ron, C. William, C. Benjamin, D. Erik, L. Robert, T. James, and B. M. Arthur, "Architectural specification for massively parallel computers: an experience and measurement-based approach: Research Articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, pp. 1271-1316, 2005.
[12] R. B. Ross, "Parallel I/O Benchmarking Consortium."
[13] Sandia National Laboratories, "Scalable IO," http://www.cs.sandia.gov/capabilities/ScalableIO/.
[14] R. Thakur, W. Gropp, and E. Lusk, "An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces," Proceedings of Frontiers '96: The Sixth Symposium on the Frontiers of Massively Parallel Computation, 1996.
[15] R. Thakur, W. Gropp, and E. Lusk, "On Implementing MPI-IO Portably and with High Performance," Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems, 1999.
[16] The National Center for SuperComputing, "HDF5 Home Page."
[17] 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.
[18] 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," IEEE International Parallel & Distributed Processing Symposium (IPDPS), Rhodes Island, Greece, 2006.