Accelerating I/O Forwarding in IBM Blue Gene/P Systems

Venkatram Vishwanath, Mark Hereld, Kamil Iskra, Dries Kimpe,
Vitali Morozov, Michael E. Papka, Robert Ross, and Kazutomo Yoshii
Argonne National Laboratory
9700 South Cass Avenue,
Argonne, IL 60439
{venkatv, hereld, iskra, dkimpe, morozov, papka, rross, kazutomo}@mcs.anl.gov

***Abstract*—Current leadership-class machines suffer from a significant imbalance between their computational power and their I/O bandwidth. I/O forwarding is a paradigm that attempts to bridge the increasing performance and scalability gap between the compute and I/O components of leadership-class machines to meet the requirements of data-intensive applications by shipping I/O calls from compute nodes to dedicated I/O nodes. I/O forwarding is a critical component of the I/O subsystem of the IBM Blue Gene/P supercomputer currently deployed at several leadership computing facilities. In this paper, we evaluate the performance of the existing I/O forwarding mechanisms for BG/P and identify the performance bottlenecks in the current design. We augment the I/O forwarding with two approaches: I/O scheduling using a work-queue model and asynchronous data staging. We evaluate the efficacy of our approaches using microbenchmarks and application-level benchmarks on leadership-class systems.**

I. Introduction

Leadership-class systems are providing unprecedented opportunities to advance science in numerous fields, such as climate sciences, biosciences, astrophysics, computational chemistry, materials sciences, high-energy physics, and nuclear physics [17]. Current leadership-class machines such as the IBM Blue Gene/P (BG/P) supercomputer at the Argonne National Laboratory and the Cray XT system at the Oak Ridge National Laboratory consist of a few hundred thousand processing elements. BG/P is the second generation of supercomputers in the Blue Gene series and has demonstrated ultrascale performance together with a novel energy-efficient design. As of November 2009, five of the top 20 systems in the Top 500 list [20] and thirteen of the top 20 most power-efficient systems were based on the Blue Gene solution [8].

While the computational power of supercomputers keeps increasing with every generation, the I/O systems have not kept pace, resulting in a significant performance bottleneck. In order to achieve higher performance, many HPC systems run a stripped-down operating system kernel on the compute nodes to reduce the operating system "noise." The IBM Blue Gene series of supercomputers takes this a step further, restricting I/O operations from the compute nodes. In order to enable applications to perform I/O, the compute node kernel ships all I/O operations to a dedicated I/O node, which performs I/O on behalf of the compute nodes. This process is known as I/O forwarding [3].

Lang et al. [12] performed an initial study of the I/O forwarding mechanism on BG/P and found the overall efficiency to be 60%. This low efficiency affects the performance of reading from and writing to storage systems and is a key in making compute-intensive applications I/O bound. Additionally, there is an increasing need to perform data analysis and visualization while a simulation is running [11]. In order to do so, data must travel down a similar path when streamed off the system, such as when performing visual analysis concurrently with the simulation. Thus, simulation-time analytics is also affected by the performance of I/O forwarding. By improving the performance of the I/O forwarding, researchers would be able to accelerate the time to solution or apply more complex models during the same time frame.

We present here a detailed analysis of the I/O forwarding mechanism on IBM BG/P systems. Novel contributions of our paper include the following:

- Identifying the performance bottlenecks in the I/O forwarding mechanism on BG/P due to the resource contention in the I/O forwarding mechanism.
- Designing an I/O scheduling mechanism for I/O forwarding on BG/P to mitigate resource contention.
- Augmenting the BG/P I/O forwarding mechanism with asynchronous data staging to overlap I/O operations with computation.
- Demonstrating our approaches using synthetic and application benchmarks to yield up to a 50% improvement over the current I/O forwarding mechanism.

The remainder of the paper is organized as follows. We present a brief overview of the BG/P I/O infrastructure in Section II and perform a detailed performance evaluation in Section III. We discuss our approach to improve the I/O performance in Section IV. Detailed experimental results and the corresponding analysis are presented in Section V. Literature related to our work is presented in Section VI. We draw conclusions in Section VII.

II. Background

We present an overview of the Argonne Leadership Computing Facility (ALCF) resources, specifically the IBM Blue Gene/P architecture, and describe the I/O subsystem of the

SC10 November 2010, New Orleans, Louisiana, USA 978-1- 4244-7558-2/10/$26.00

BG/P. The ALCF resources described are used for the system evaluation.

A. The Argonne Leadership Computing Facility (ALCF)

ALCF is a U.S. Department of Energy facility that provides leadership-class computing infrastructure to the scientific community. Figure 1 depicts the architecture of the primary ALCF resources consisting of the compute resource (Intrepid), the data analysis cluster (Eureka), and the file server nodes interconnected by a large Myrinet switch complex, which are the components of the ALCF that are of focus in this paper.

The BG/P system is the second in a series of supercomputers designed by IBM to provide extreme-scale performance together with high reliability and low power consumption. Intrepid is a 160K core BG/P system with a peak performance of 557 TF, 80 TB local memory, and 640 I/O nodes each with 4 cores, connected to the switching interconnect with aggregate 6.4 Tbps. BG/P systems comprise individual racks that can be connected together; each rack contains 1,024 four-core nodes, for a total of 4,096 cores per rack. Blue Gene systems have a hierarchical structure; 64 nodes are grouped into a pset, and 8 psets together form a midplane that contains 512 nodes. Each rack contains two such midplanes. Large Blue Gene systems are constructed in multiple rows of racks.

Each node on the BG/P uses a quad-core, 32-bit, 850 MHz IBM Power PC 450 with 2GB of memory. Each node is connected to multiple networks. The I/O and interprocess communication of BG/P travel on separate internal networks. A three-dimensional torus network is used for point-to-point communicating among compute nodes (CNs), while a tree network allows CNs to perform I/O to designated I/O forwarding nodes (this network can also be used for optimized MPI collective operations). In the BG/P system, these I/O forwarding nodes are referred to simply as I/O nodes (IONs). For each pset a dedicated ION receives I/O requests from the CNs in that group and forwards those requests over its 10 gigabit Ethernet port to the external I/O network.

The IONs are connected to an external 10 gigabit Ethernet switch, which provides I/O connectivity to file servers nodes (FSNs) of a clusterwide file system as well as connectivity to the data analysis (DA) cluster nodes. Eureka, the data analysis cluster, contains 100 servers with 800 Xeon cores, 3.2 TB memory, and 200 nVidia Quadro FX 5600 GPUs. Eureka is connected to the switch with 100 links at 10 Gbps each. There are 128 file server nodes, each node of which is a dual-core dual-processor AMD Opteron with 8 GB RAM per core. Each FSN is connected to the Myrinet switch complex over 10 Gbps. The FSNs are connected via InfiniBand 4X DDR to 16 Data Direct Network 9900 storage devices.

B. I/O Forwarding in BG/P

The BG/P runs a lightweight Linux-like operating system on the compute nodes called the Compute Node Kernel (CNK). CNK is a minimalistic kernel that mitigates operating system interference by disabling support for multiprocessing and POSIX I/O system calls. To enable applications to perform I/O, CNK forwards all I/O operations to a dedicated I/O node, which executes I/O on behalf of the compute nodes. There are two options for I/O forwarding on BG/P: CIOD from IBM and ZOID from the ZeptoOS project [22].

1) Control and I/O Daemon (CIOD): CIOD [15]) is the BG/P I/O-forwarding infrastructure provided by IBM. Figure 2a depicts the architecture of CIOD. It consists of a user-level daemon running on the ION. For each CN process in the partition, a dedicated I/O proxy process handles its associated I/O operations. CIOD receives I/O requests forwarded from the compute nodes over the collective network and copies them to a shared-memory region. The CIOD daemon then communicates with I/O proxy process using this shared memory. The I/O proxy executes the I/O function on behalf of the CN and returns the result of the function calls to the compute node.

2) ZeptoOS I/O Daemon (ZOID): ZeptoOS [22] is a research project investigating open-source operating systems for petascale architectures with 10,000 to 1 million CPUs. Its current focus is the IBM Blue Gene/P, where it provides a complete, flexible, high-performance software stack, including a Linux-based compute node kernel, HPC communication libraries, I/O forwarding, and performance analysis. ZOID [10] is the I/O forwarding component of the ZeptoOS project responsible for handling the I/O operations of the CNs.

Figure 2b depicts the multithreaded ZOID architecture. ZOID creates a pool of threads large enough to handle simultaneous I/O operations from all CNs on separate threads. As shown in the figure, the write operation performed by the application on the CN is forwarded to the ION. This is handled by the ZOID thread responsible for the CN and first copied into a buffer managed by ZOID. The ZOID thread executes the write on behalf of the compute node. It sends back the result of the function calls to the compute node and deletes the buffer. A distinguishing feature of ZOID with respect to CIOD is the multithreaded, rather than multiprocess, architecture. In addition, ZOID can be easily extended with new functionality via plug-ins [10].

III. Performance Evaluation of the BG/P I/O System

The BG/P I/O system consists of two stages: the collective network between the CNs and the ION and the external I/O network connecting the ION to the file server and analysis nodes. In this section, we evaluate the performance of each of these stages and then measure the combined end-to-end performance of the two stages. Figure 3 depicts the stages involved in the BG/P I/O system.

A. Collective Network Throughput

The achievable collective network throughput gives us a measure of how fast data can be moved from the CNs in a pset to their designated ION. The theoretical peak bandwidth of the collective network is 850 MBps ($\approx$ 810 MiBps).[1] The peak throughput—taking into account 16 bytes of header

[1] 1 MiB = 1024 * 1024 bytes. In our evaluations MB refers to MiB.

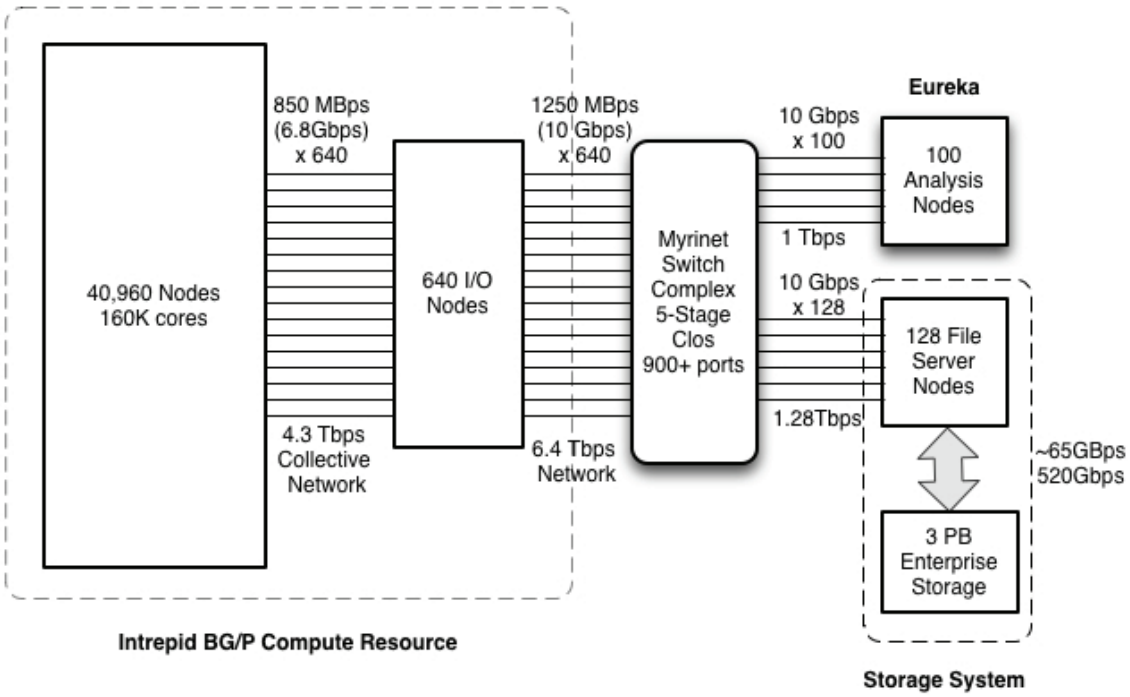
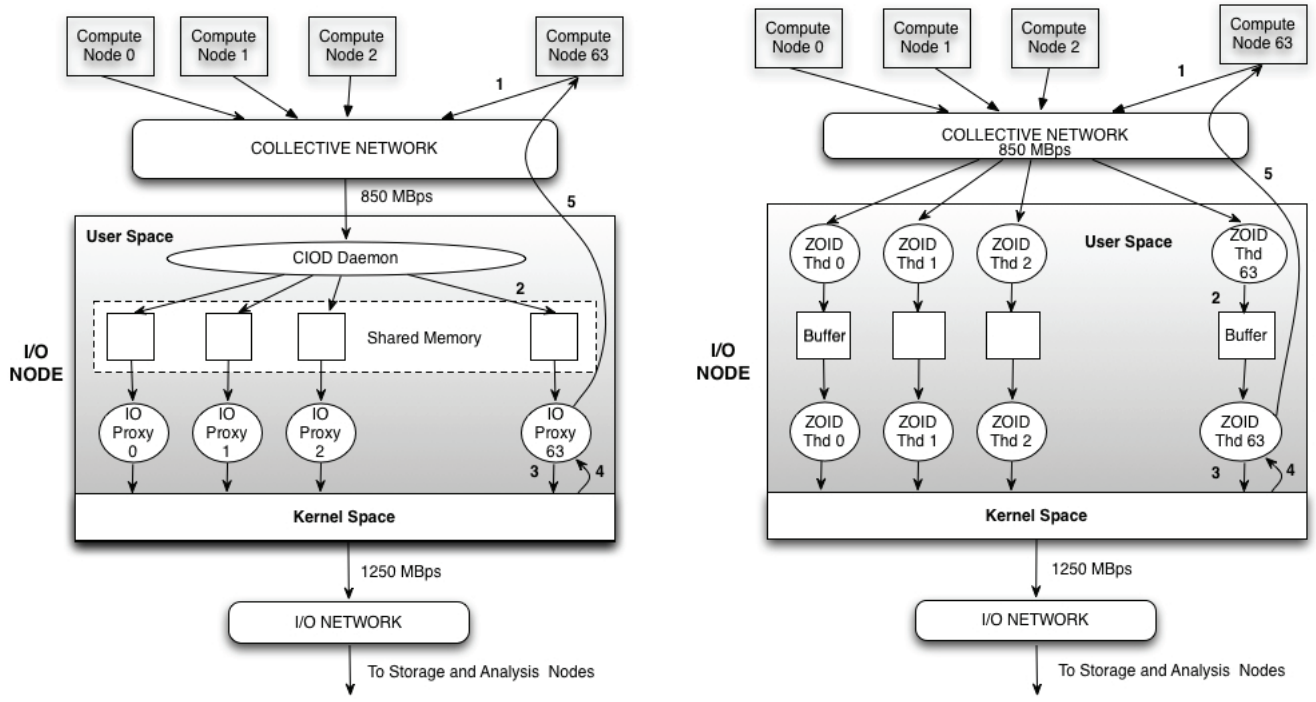


Fig. 1. The Argonne Leadership Computing Facility maintains a 160K core BG/P compute cluster (Intrepid), data analysis cluster (Eureka), and the file server nodes all interconnected by a 5-stage Myrinet switch complex, as well as other compute infrastructure including several test systems and a large computing cloud resource



(a) CIOD (b) ZOID

Fig. 2. The two I/O forwarding mechanisms available on the BG/P at the ALCF

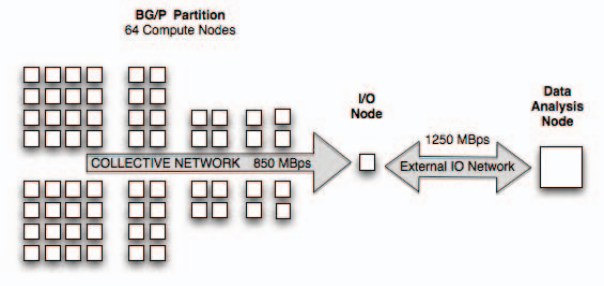


Fig. 3. Architecture of the BG/P I/O system

information for the I/O forwarding mechanism in both CIOD and ZOID for every 256-byte payload, as well as 10 bytes of hardware headers related to operation control and link reliability—is ≈ 731 MiBps. To measure the collective network performance, we wrote a parallel benchmark to read and write data to /dev/null on the compute nodes. As the I/O operations on the CN are forwarded and executed on the ION, this benchmark effectively measures the achievable throughput of the collective network. We tested this benchmark using CIOD and ZOID for moving data from CN to the ION, varying the buffer sizes as well as the number of CNs in a pset involved concurrently in the transfer.

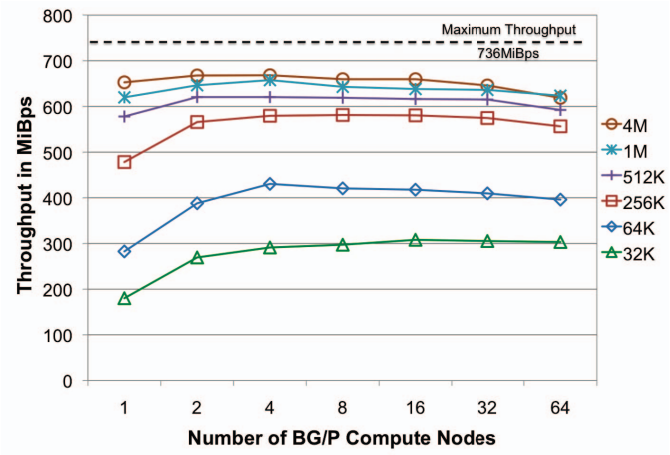


Fig. 4. Performance of collective network streaming from compute nodes to I/O node

Figure 4 depicts the performance of CIOD to forward I/O from CNs to the ION. As we increase the message size, the ratio of the time spent on the actual data transfer to the time spent on sending the control information increases, thus leading to increased network utilization. Additionally, the performance peaks between 4 to 8 nodes and reduces as we increase beyond 32 nodes. The ION is a 850 MHz quad-core processor. There is an increased contention for resources on the ION as we increase the number of CNs, leading to performance degradation.

ZOID demonstrates performance very similar to that of CIOD. We do notice a 2% performance improvement over CIOD with respect to I/O forwarding, however. This is primarily due to the multithreaded design of ZOID and the lower overhead associated with thread context switches in ZOID compared to the process context switches in CIOD. Thus, for a given partition and a message size of 1 MiB, the collective network is able to sustain up to 680 MiBps, or 93% efficiency.

B. External I/O Network Throughput

The second stage of the I/O pipeline is the external I/O network connecting the IONs to the DAs (and FSNs). The ION is connected by a 10 Gbps NIC to the analysis and file system nodes. Thus, the theoretical peak bandwidth achievable is ≈ 1190 MiBps. To measure the achievable network throughput between the ION and DA, we used nuttcp version 7.1.2 [16], a commonly used performance benchmarking tool in high=performance networking. Nuttcp supports multiple protocols; we used TCP for the tests. Since the external I/O network is shared, we report the maximum achievable throughput for five 2-minute runs, where the message size was 1 MiB.

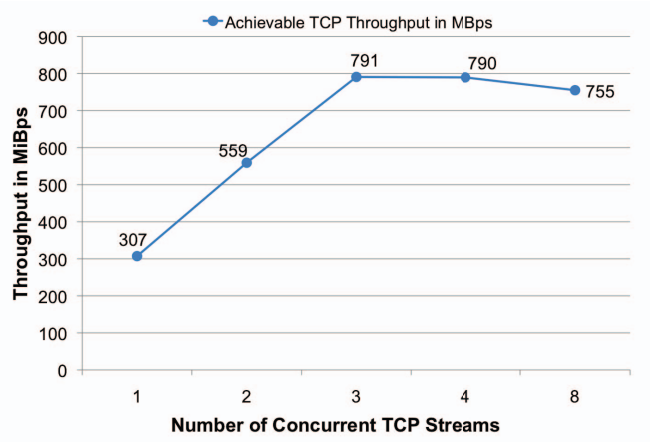


Fig. 5. Performance of data streaming from a BG/P I/O node to an analysis node

Figure 5 depicts the achievable throughput of moving data (memory-to-memory) from the ION to a DA over the external I/O network. In the figure, we notice that a single thread is able to sustain only 307 MiBps. The reason is primarily the lower compute capability of a single BG/P ION core, which is unable to sustain any additional network traffic.

Increasing the number of sending threads to 2 leads to an improvement in the sustained aggregate throughput as we use more computational resources to drive the network. Using 4 threads on the ION, we are able to sustain a maximum of 791 MiBps. As we increase the number of threads to 8, however, the performance decreases, mainly because of the increased contention of resources on the ION. Using nuttcp, we were able to sustain 1110 MiBps between two DA nodes with a single thread. The reason is the higher compute capability of the processors on the DA nodes. The DA nodes are dual-processor quad-core 2 GHz Intel Xeon processors, compared to the 850 MHz PowerPC quad-core processor of the ION.

C. End-to-End Performance of the BG/P I/O System

Given the effective collective network throughput and the external I/O network throughput, one would hope that the achievable end-to-end performance of the BG/P I/O system would be close to the minimum of the maximum sustained throughput of the collective network and the external I/O network (i.e., $\approx$ 650 MiBps). To measure the I/O forwarding performance, we wrote a parallel memory-to-memory data transfer benchmark which uses sockets to communicate between CNs and the analysis node using both the collective and external I/O network. A memory-to-memory data transfer benchmark eliminates any effects of the storage system and gives us a true measure of the I/O forwarding capabilities. Additionally, this memory-to-memory transfer is critical for data movement between the CNs and the FSNs for parallel filesystems, and data movement between the CNs and DAs is of increasing importance for real-time analysis.

Figure 6 compares the end-to-end sustained throughput of CIOD and ZOID as we vary the number of CNs in a given pset. This experiment involved 1,000 iterations of transferring 1 MiB from the CN memory to the DA node memory. The maximum throughput plotted in the figure is the minimum of the maximum sustained throughputs from Figure 4 and Figure 5. As seen from the figure, the maximum throughput sustained by CIOD and ZOID is $\approx$ 420 MiBps, which is only 66% of the maximum achievable throughput. This performance is consistent with the result obtained by Lang et al. [12] for transferring data from the CNs memory to the FSN memory for a given partition. Additionally, as we increase the number of nodes, we notice a drop in performance due to the increased resource contention between the various threads.

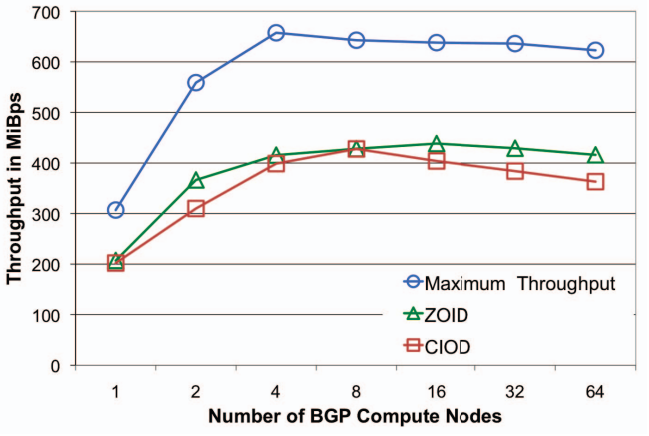


Fig. 6. Performance of I/O forwarding between an I/O node and analysis node

IV. Design

A key factor impacting the performance of I/O forwarding in BG/P is the resource contention on the ION among the various threads (ZOID) and processes (CIOD) performing the I/O operation. Additionally, in CIOD and ZOID, the I/O operations are synchronous; that is, the application on the CN is blocked until the I/O operation is completed by the I/O forwarding mechanism. We explore two potential approaches to improve I/O forwarding performance: (1) an I/O scheduling mechanism incorporating a work-queue model to mitigate resource contention and (2) asynchronous data staging to overlap the application's computation with the I/O operations. We implement these features in ZOID because of the flexibility and extensibility of ZOID's design.

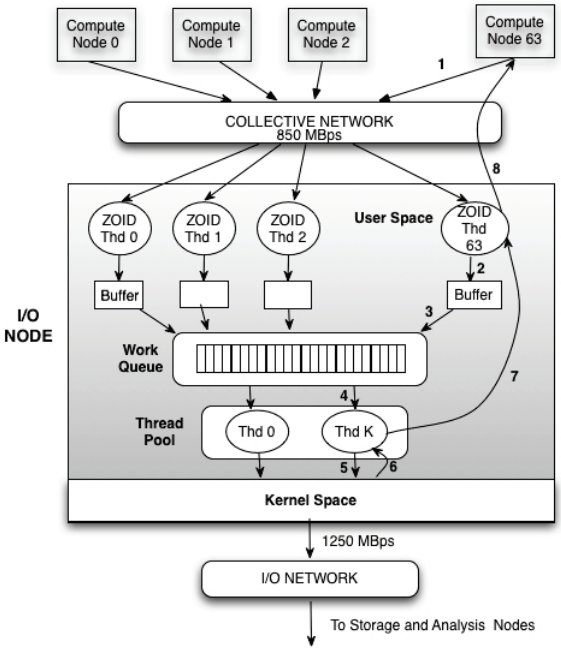


Fig. 7. I/O forwarding using a thread-pool and worker queue for I/O forwarding

As seen in Figure 2b, ZOID uses a thread-per-CN to perform I/O forwarding. To enable I/O scheduling, we augmented ZOID's thread model with a work queue model using a shared first-in first-out (FIFO) work queue. This is depicted in Figure 7. Instead of executing the I/O operation, the ZOID thread now enqueues the I/O task into the work queue. We use a pool of worker threads to handle the I/O tasks in the work queue. These worker threads are launched at job startup, and the number of worker threads can be controlled via an environment variable during job submission. The worker threads dequeue the I/O tasks from the work queue and execute them. To facilitate I/O multiplexing per thread, a worker thread dequeues multiple I/O requests and executes them in an event loop. The current implementation uses a poll-based event mechanism. We use a simple load-balancing heuristic to balance the tasks among the work threads. One could easily augment this to take the data sizes into account as well as maintain separate queues based on the priority of data. Once the worker thread completes an I/O task, it wakes up the associated ZOID thread and passes the status of the I/O operation to the ZOID thread. The ZOID thread passes this status back to the application, which is now free to

progress with its computation. In this case, the data staging is synchronous because the application is blocked until the I/O operation is completed.

The second optimization we incorporate is to overlap computation with the I/O operation via asynchronous data staging. Figure 8 depicts the architecture of ZOID augmented to support asynchronous data staging as well as I/O scheduling using a work queue. Asynchronous data staging blocks the computation only for the duration of copying data from the CN to the ION. The I/O operation can now concurrently occur with the computation. To facilitate asynchronous data staging, we designed a custom buffer management layer (BML) in ZOID. This enables each ZOID thread to receive data from the collective network into buffers allocated via BML. The ZOID thread inserts the I/O task into the work queue and returns, enabling the computation to progress concurrently with the I/O operation. In addition, we maintain a database of open I/O descriptors; for each, we keep a list of completed and in-progress operations and their associated status, including errors. We distinguish the various I/O operations performed on a particular descriptor via a counter. Errors are passed to the application on subsequent operations on the descriptor.

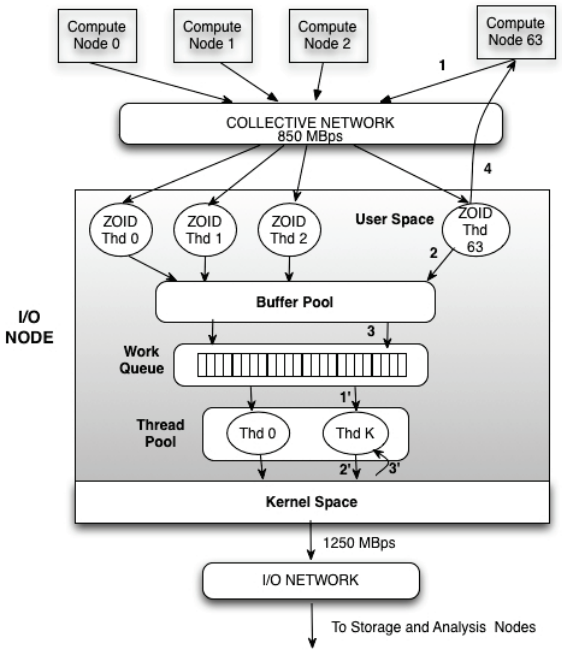


Fig. 8. Augmenting I/O forwarding with asynchronous data staging

The total memory managed by BML can be controlled by an environment variable during the application launch. In the current implementation, the buffer management allocates buffers that are powers of 2 bytes. We plan to augment the implementation to support arbitrary message sizes by using memory allocators such as tcmalloc [1] and hoard [4]. On completion of the I/O operation, the worker thread returns the memory buffer to the buffer pool and updates the operation entry related to the descriptor in the database with the function return value. In the current implementation, asynchronous data staging is used only for the data operations such as reads and writes to sockets and files. Operations for opening and closing files and sockets or querying their attributes are handled synchronously. The amount of data that can be buffered is limited by the available memory on the ION. If there is insufficient memory to stage the data, the I/O operation is blocked until a number of queued I/O operations complete and sufficient memory is available. For large transfers, both CIOD and ZOID block the I/O operation till sufficient memory is present on the I/O Node.

V. Evaluation

We evaluate the efficacy of I/O scheduling and asynchronous data staging to forward I/O in BG/P using a set of microbenchmarks and an application-level benchmark.

A. Microbenchmarks

We use the memory-to-memory data transfer microbenchmark used in Section III-C to measure the effective end-to-end performance of the I/O network throughput achievable between BG/P CNs and the DA node. We compare performance of the existing I/O forwarding approaches and our proposed approach as we scale the number of CNs in a pset concurrently performing I/O. Next, we evaluate the performance as we vary the message size and evaluate the impact of the number of threads in the worker pool on the I/O forwarding performance. We then present large-scale weak scaling experiments of the various I/O forwarding mechanisms. Since the I/O network is shared, the results presented are the maximum of five runs, each consisting of 1,000 iterations.

1) Performance with Number of Nodes: We evaluate the performance as we scale the number of nodes in a BG/P pset. In this experiment, we use a message size of 1 MiB and 4 worker threads for executing the I/O operation in the work queue.

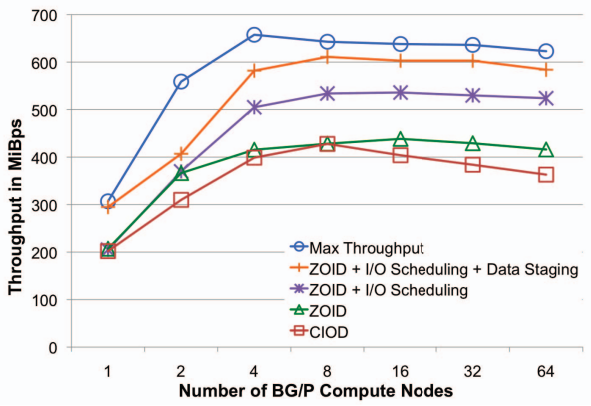


Fig. 9. Performance comparison of I/O forwarding mechanism as we increase the number of CNs sending 1 MiB messages over the I/O network to a DA node

Figure 9 depicts the achievable end-to-end throughput in MiBps using the various I/O forwarding strategies as we increase the number of CNs concurrently performing writes in a partition. In the figure, the efficacy of I/O scheduling is clearly evident as we increase the number of CNs concurrently performing I/O. Typically in HPC applications, all the nodes concurrently perform I/O operations. By incorporating I/O scheduling in ZOID, we notice up to 38% improvement in performance over CIOD for 32 CNs and up to 23% improvement over the default ZOID thread mechanism and up to 83% throughput efficiency. With 4 CNs, we notice an improvement primarily due to the I/O multiplexing per thread. Thus, I/O scheduling plays a key role in mitigating resource contention and improving the achievable throughput.

Asynchronous data staging and I/O scheduling together give 57% improvement over CIOD for 32 CNs and up to 40% over the default ZOID performance. The combination yields a 14% improvement over the I/O scheduling alone and achieves approximately 95% efficiency with respect to the maximum achievable throughput.

2) Performance with Message Size: We evaluate the performance of the various forwarding mechanism as we vary the message size. The number of nodes concurrently performing the I/O operations was set to 64. CIOD and ZOID use a two-step approach wherein the function parameters are first sent from the CN to the ION and the data is then transferred over the collective network to the ION. This is the primary performance gating factor for smaller message sizes. As we increase the message size, the time spent exchanging the control information with respect to the time spent sending the message decreases, leading to increased network utilization of the collective network.

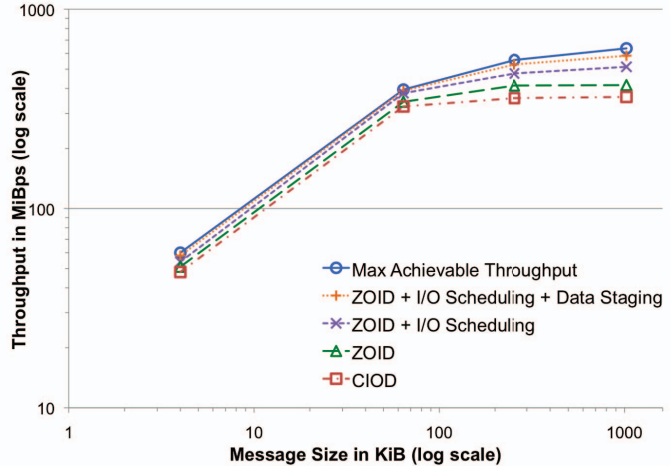


Fig. 10. Performance comparison of I/O forwarding mechanism for 64 CNs over the I/O network to a DA node with varying message size

The benefits of I/O scheduling and asynchronous data staging are evident as we increase the message size. With a 256 KiB message size, CIOD and ZOID achieve 64% and 74% efficiency, respectively. With I/O scheduling we achieve 86% efficiency, and asynchronous data staging increases this to 95% efficiency. We notice a significant improvement in performance for the various message sizes with both I/O scheduling and asynchronous data staging. Thus, asynchronous data staging together with I/O scheduling achieves close to the maximum throughput achievable.

3) Efficacy of Thread Pool Size: We evaluate the impact of the number of worker threads on the performance of I/O forwarding. In this case the I/O forwarding mechanism involves both I/O scheduling and asynchronous data staging. Figure 11 depicts the maximum achievable throughput for a message size of 1 MiB as we vary the number of worker threads handling the I/O operations.

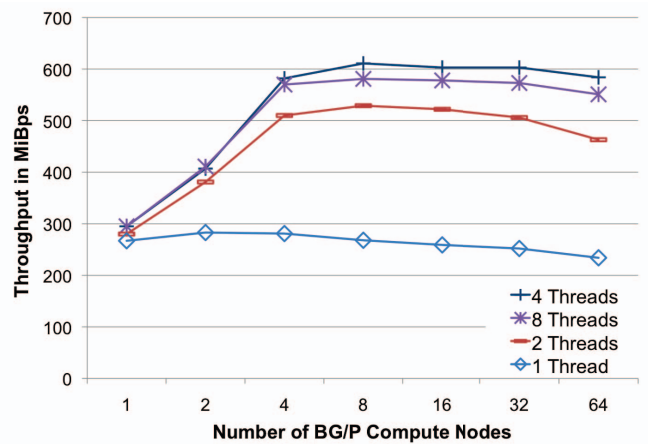


Fig. 11. Impact of the number of threads on I/O forwarding.

We see that a single thread is unable to sustain more than 300 MiBps. The reason is that a single thread, because ofthe lower clock speed of a single BG/P core, is unable to sustain more than 307 MiBps (as evidenced in Figure 5). At 2 and 4 threads, we see an improved performance relative to a single thread, primarily because multiple threads are available to drive the network. As we increase the number of threads to 8, however, we notice a drop in performance relative to 4 threads. The main reason is the resource contention among the threads for the 4 cores. The maximum performance is obtained with 4 threads due to the lower contention among resources.

4) Weak Scaling Performance: We compare the weak scaling performance of the various I/O forwarding mechanisms to move data (memory-to-memory) between the BG/P compute nodes and DA nodes as we scale the number of BG/P nodes. We use a message size of 1 MiB. For the data transfer, 20 DA nodes are used as sinks. The connections from the compute nodes were distributed among the DA nodes. This is similar to how the MxN problem [5] redistributes the data from M nodes to N nodes as well as is the typical distribution of data from the compute nodes to the file server nodes for parallel file systems. Since the I/O network infrastructure is shared, we report the maximum performance achieved in five runs, each run sustained for 1,000 iterations.

Figure 12 compares the weak scaling of the various I/O forwarding mechanisms. As we scale the number of BG/P nodes, we notice a performance improvement in case of all the three I/O forwarding mechanisms. This is mainly due to the fact that as the number of BG/P psets(and nodes) involved in the experiments increases, more I/O nodes are involved in the data movement thus leading to an increase in I/O network resources. In case of 256 BG/P nodes, 512 nodes, and 1024 nodes, we have 4, 8, and 16 I/O nodes, respectively. Using asynchronous data staging and I/O scheduling, we observe a 53% improvement over CIOD and 33% improvement over ZOID. With 512 nodes and 1024 nodes, respectively, our approaches yield a 43% and 47% improvement over the performance of CIOD and a 25% and 34% improvement over ZOID. Thus, our proposed approaches accelerate parallel data movement, a critical factor in performing real-time data visualization and analytics as well as in moving data from the compute nodes to the file server nodes.

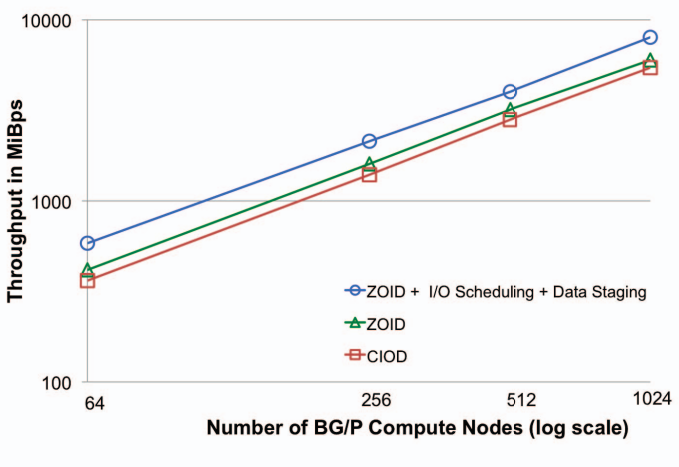


Fig. 12. Weak scaling performance of the I/O forwarding mechanisms

B. Application Benchmark

The results presented so far indicate the peak performance possible. However, the success of our approach is defined by its ability to provide performance improvement for applications. In this section we evaluate the efficacy of our approach using the MADbench2 [6] [14] benchmark that simulates I/O from an HPC application. MADbench2 is derived from the MADspec data analysis code, which estimates the angular power spectrum of cosmic microwave background radiation in the sky from noisy pixelized datasets. As part of its calculations, the MADspec code (and likewise the MADbench2 benchmark) performs extremely large out-of-core matrix operations, requiring successive writes and reads of large contiguous data from either shared or individual files. Because of the large, contiguous mixed read and write patterns that MADbench2 performs and its ability to test various parameters (shared files, POSIX vs. MPI-IO, etc.), it has become a leading benchmark in the parallel I/O community [12].

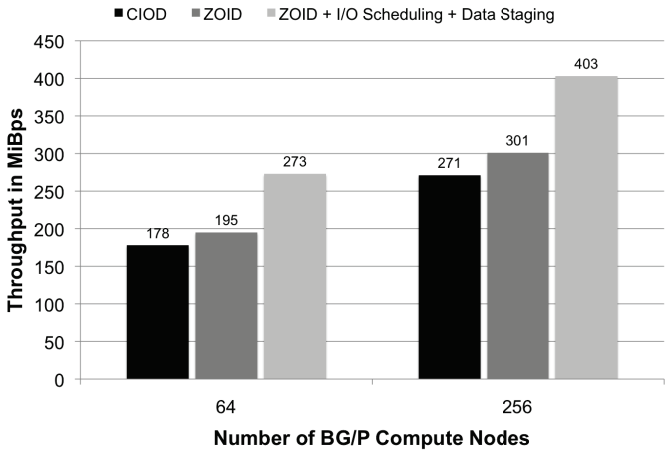


Fig. 13. Performance of the MADBench2 application benchmark using the I/O forwarding mechanisms.

In our tests, we examine the efficacy of the I/O forwarding mechanism using asynchronous data staging and I/O scheduling and compare it with the CIOD and ZOID I/O forwarding mechanism to perform I/O to the GPFS storage on Intrepid. We ran MADbench2 with a single pset containing 64 nodes and scaled (weak) to 256 nodes. MADbench2 was compiled to run in I/O mode, so our tests did not include MPI communication or perform significant computation (the busy-work exponent α was set to 1). Thus, these results demonstrate just the capability of BG/P I/O system. The file alignment used by MADbench2 for these runs was the default of 4,096. We allowed all processes to perform I/O simultaneously (RMOD and WMOD both set to 1). For 64 nodes we fixed the problem size at NPIX = 4096, and for 256 nodes we fixed NPIX at 8192, enabling each process to performing I/O operations of roughly 2 MiB per operation. The number of component matrices was set to 1024. In aggregate, the I/O performed by the benchmark totaled 128 GB for 64 nodes and 512 GB for 256 nodes. Since the I/O subsystem is shared and heavily used, we report the maximum achievable throughput of five runs.

The performance of MADbench2 using the various I/O forwarding mechanisms is shown in Figure 13. With 64 nodes, I/O forwarding using asynchronous data staging and I/O scheduling achieves 53% improvement in performance over CIOD and 40% improvement over ZOID. With 256 nodes, it achieves 49% improvement over CIOD and 34% improvement over ZOID. Thus, asynchronous data staging and I/O scheduling are critical to accelerate the I/O performance for applications on leadership-class machines.

VI. Related Work

Remote Procedure Call (RPC) is a communication mechanism that enables applications to execute the called procedure on a different host machine. RPCs serialize function parameters that are then passed over the network to a remote server. The server deserializes the parameters, executes the function call, and sends the results back to the client. I/O forwarding is

essentially a specialized form of RPC, where the I/O function calls are sent to the I/O node for execution.

The Computational Plant (Cplant) [19] machine at Sandia National Laboratories introduced the concept of I/O forwarding in HPC systems. The Cplant compute nodes forwarded the I/O requests to a NFSv2 proxy that performed I/O on behalf of the compute nodes. Limitations of the Cplant I/O forwarding infrastructure include lack of data caching and I/O aggregation, limited support for multithreading, and the absence of locking.

The fast collective network protocol (FCNP) [18] for IBM BG/P was designed to improve the performance of data transfer between the compute nodes and the I/O nodes. FCNP can take advantage of the I/O scheduling and asynchronous data transfer mechanism proposed to improve the end-to-end performance.

The I/O forwarding scalability layer project (IOFSL) [3] is a collaboration between multiple DOE national laboratories to provide a scalable and portable I/O forwarding layer for large-scale systems including BG/P, Cray XT, Roadrunner, and commodity Linux clusters. Like our work, IOFSL uses a thread pool to decouple the number of I/O threads from the number of compute clients. However, IOFSL does not provide asynchronous data staging. We plan to integrate this into IOFSL.

Using the IONs for filesystem caching has achieved improved performance with parallel file systems [9]. In this work, a thread on the I/O node is used to aggregate data in order to perform larger writes to storage. However, just a single I/O thread is used. As seen in our results, a single thread is unable to saturate the external I/O network. The write-back caching and prefetching optimization can leverage our work to improve the performance of data movement.

Asynchronous data staging has been used in Grid computing [21] to improve I/O performance by hiding the latency associated with wide-are networking using dedicated staging nodes and novel data transfer protocols. In large-scale HPC systems, end-systems are one of the primary source of bottleneck. In BG/P, the resource contention on I/O nodes is a critical performance bottleneck, and our work involves mitigating this resource bottleneck.

DART [7] and DataStager [2] have demonstrated high-performance transfers for Cray XT5 and use dedicated data staging nodes. Staging nodes are an integral part of the BG/P architecture. A focus of our I/O forwarding is to forward all I/O operations transparently without any changes to an application. DART, on the other hand, requires applications to be modified via a lightweight API. Additionally, our contribution includes efficient scheduling of data movement from the BG/P I/O forwarding nodes to the storage and analysis nodes. Neither DART nor DataStager are available for the BG/P architecture. These could benefit from our work for the BG/P architecture.

Active buffering with threads (ABT) [13] transforms blocking write operations into nonblocking operations by buffering data and subsequently transferring it to storage in the background. Unlike our work, ABT is implemented in ROMIO and performs the buffering on the compute client. Since it requires a thread to perform the write-back, it cannot be used on systems that do not offer full thread support for compute nodes, as is the case with the BG/P system. In addition, since in our work the buffering occurs on the I/O node, no additional CPU cycles are needed on the compute node. In ABT, the I/O thread on the compute node competes for CPU time with the application.

VII. Conclusions and Future Work

The performance mismatch between the computing and I/O components of current-generation HPC systems has made I/O the critical bottleneck for data-intensive scientific applications. I/O forwarding attempts to bridge this increasing performance gap by regulating the I/O traffic. Our evaluation of the I/O forwarding mechanism in IBM Blue Gene/P revealed significant performance bottlenecks caused by resource contention on the I/O node. Our approaches to overcome this bottleneck include an I/O scheduling mechanism leveraging a work-queue model and an asynchronous data staging mechanism. On the leadership computers at Argonne National Laboratory, these approaches yield up to 53% improvement in performance over the existing I/O forwarding mechanism and close to 95% efficiency. We believe this is a significant step toward scaling the performance of applications on current large-scale systems and will provide insight for the design of I/O architectures for exascale systems.

In the near future, we plan to integrate the approaches presented in this paper into the IOFSL framework. Since the compute capabilities of the I/O forwarding nodes are usually underutilized, we are investigating techniques to offload data filtering onto the I/O forwarding nodes in order to reduce the amount of data written to storage as well as to facilitate in situ analytics.

Acknowledgment

We gratefully acknowledge the use of the resources of the Argonne Leadership Computing Facility at Argonne National Laboratory. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357 and an Argonne National Laboratory Director Postdoctoral Fellowship. We are grateful to Rob Latham and Phil Carns for the application benchmarks and related discussions. We acknowledge Gail Pieper, Tom Uram, Joe Insley, Eric Olson, Randy Hudson, Tom Peterka, Jason Cope, Loren Wilson, Bill Allcock, Tisha Stacey, Kevin Harms, Andrew Cherry, Susan Coghlan and the ALCF team for discussions and help related to the paper.

References

[1] Thread caching malloc (tcmalloc).

[2] Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. Datastager: Scalable data staging services for petascale applications. In *HPDC*, pages 39–48, 2009.

[3] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. In *Proceedings of IEEE Conference on Cluster Computing, New Orleans, LA*, September 2009.

[4] Emery D. Berger, Kathryn S. McKinley, Robert D. Blumofe, and Paul R. Wilson. Hoard: A scalable memory allocator for multithreaded applications. *SIGPLAN Not.*, 35(11):117–128, 2000.

[5] Felipe Bertrand, Randall Bramley, Alan Sussman, David E. Bernholdt, James A. Kohl, Jay W. Larson, and Kostadin B. Damevski. Data redistribution and remote method invocation in parallel component architectures. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 40.2, Washington, DC, USA, 2005. IEEE Computer Society.

[6] J. Borrill, J. Carter, L. Oliker, and D. Skinner. Integrated performance monitoring of a cosmology application on leading HEC platforms. In *ICPP '05: Proceedings of the 2005 International Conference on Parallel Processing*, pages 119–128, Washington, DC, USA, 2005. IEEE Computer Society.

[7] C. Docan, M. Parashar, and S. Klasky. Enabling high speed asynchronous data extraction and transfer using DART. *Proceedings of the 17th International Syposium on High-Performance Distributed Computing (HPDC). IEEE Computer Society Press, Boston, MA*, 2008.

[8] Green500. http://www.green500.org/.

[9] Florin Isaila, Javier García Blas, Jesús Carretero, Robert Latham, Samuel Lang, and Robert B. Ross. Latency hiding file I/O for Blue Gene systems. In *CCGRID*, pages 212–219, 2009.

[10] Kamil Iskra, John W. Romein, Kazutomo Yoshii, and Pete Beckman. Zoid: I/o-forwarding infrastructure for petascale architectures. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 153–162, New York, NY, USA, 2008. ACM.

[11] Chris Johnson and Rob Ross. DOE/ASCR report from the workshop on visual analysis and data exploration at extreme scale, October 2007.

[12] Samuel Lang, Philip Carns, Robert Latham, Robert Ross, Kevin Harms, and William Allcock. I/o performance challenges at leadership scale. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.

[13] Xiaosong Ma, Marianne Winslett, Jonghyun Lee, and Shengke Yu. Improving MPI-IO output performance with active buffering plus threads. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, pages 22–26, Washington, DC, USA, 2003. IEEE Computer Society.

[14] MADBench2. http://crd.lbl.gov/~borrill/MADbench2/.

[15] José Moreira, Michael Brutman, José Casta nos, Thomas Engelsiepen, Mark Giampapa, Tom Gooding, Roger Haskin, Todd Inglett, Derek Lieber, Pat McCarthy, Mike Mundy, Jeff Parker, and Brian Wallenfelt. Designing a highly-scalable operating system: the Blue Gene/L story. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 118, New York, NY, USA, 2006. ACM.

[16] nuttcp. http://www.lcp.nrl.navy.mil/nuttcp/.

[17] Committee on the Potential Impact of High-End Computing on Illustrative Fields of Science and National Research Council Engineering. The potential impact of high-end capability computing on four illustrative fields of science and engineering. http://www.nap.edu/catalog.php?record_id=12451.

[18] John W. Romein. FCNP: Fast i/o on the Blue Gene/P. In *Parallel and Distributed Processing Techniques and Applications (PDPTA'09)*, volume 1, pages 225–231, Las Vegas, NV, July 2009.

[19] Sandia National Laboratories. Computational Plant. http://www.cs.sandia.gov/cplant.

[20] TOP500. http://www.top500.org/.

[21] Venkatram Vishwanath, Robert Burns, Jason Leigh, and Michael Seablom. Accelerating tropical cyclone analysis using LambdaRAM, a distributed data cache over wide-area ultra-fast networks. *Future Generation Comp. Syst.*, 25(2):184–191, 2009.

[22] Zepto-OS. http://www.zeptoos.org.