

Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems

Venkatram Vishwanath, Mark Hereld, Vitali Morozov and Michael E. Papka
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439
venkatv,hereld,morozov,papka@mcs.anl.gov

ABSTRACT

There is growing concern that I/O systems will be hard pressed to satisfy the requirements of future leadership-class machines. Even current machines are found to be I/O bound for some applications. In this paper, we identify existing performance bottlenecks in data movement for I/O on the IBM Blue Gene/P (BG/P) supercomputer currently deployed at several leadership computing facilities. We improve the I/O performance by exploiting the network topology of BG/P for collective I/O, leveraging data semantics of applications and incorporating asynchronous data staging. We demonstrate the efficacy of our approaches for synthetic benchmark experiments and for application-level benchmarks at scale on leadership computing systems.

1. INTRODUCTION

Today's largest computational systems are providing unprecedented opportunities to advance science in numerous fields, such as climate sciences, biosciences, astrophysics, computational chemistry, high-energy physics, materials sciences, and nuclear physics [17]. Current Department of Energy (DOE) leadership-class machines such as the IBM Blue Gene/P (BG/P) supercomputer at the Argonne National Laboratory (ANL) and the Cray XT system at the Oak Ridge National Laboratory (ORNL) 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.

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. The *ExaScale Software Study: Software Challenges in Extreme Scale Systems* explains it this way: "Not all existing applications will scale to terascale, petascale, or on to exascale given current application/architecture characteristics" citing "I/O bandwidth" as one of the issues [19]. On top of this, one often finds that existing solutions only achieve

a fraction of quoted capabilities. The *International Exascale Software Project Roadmap* [5] notes that management, analysis, mining, and knowledge discovery from data sets of this scale are very challenging problems, yet a critical one in petascale systems and an even greater one for exascale.

In [20], we focused on the performance of the BG/P I/O nodes (ION) in passing data from the collective network to the external network. In this paper we consider the performance details of all available networks (torus, tree, external) and how best to deploy them in order to minimize the time spent blocked on I/O. Special consideration is given to network topologies. We have developed an infrastructure called GLEAN as our vehicle by which we develop, demonstrate and deploy these improvements. We are motivated to help increase the scientific output of leadership facilities. GLEAN provides a mechanism for improved data movement and staging for accelerating I/O, interfacing to running simulations for co-analysis, and/or an interface for in situ analysis via a zero to minimal modification to the existing application code base.

The novel contributions of our paper include the following:

- Identifying the performance bottlenecks in data movement for I/O in BG/P.
- Design of topology-aware data movement and aggregation mechanisms for BG/P.
- Identifying the performance benefits of restricting communication for aggregation to a reduced set of nodes instead of global communication.
- Demonstrating our approaches using synthetic and application benchmarks to yield multi-fold improvement on leadership-class systems.

The remainder of the paper is organized as follows. We present a brief overview of the Argonne Leadership Computing Facility, I/O forwarding in BG/P, and a summary of our previous findings in Section 2. Sections 3 and 4 describe the network performance issues under consideration and our approach to exploiting topological concerns. In Section 5, we present our performance results for synthetic benchmarks. In Section 6, we evaluate the effectiveness of GLEAN in application performance tests at scale. Section 7 covers related work to provide the reader with context for the reported results and design choices. Finally, we close in Section 8 with a discussion of conclusions and future work.

2. BACKGROUND

We present an overview of the Argonne Leadership Computing Facility (ALCF) resources, specifically the IBM Blue

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SC11, November 12-18, 2011, Seattle, Washington, USA
Copyright 2011 ACM 978-1-4503-0771-0/11/11 ...\$10.00.

Gene/P architecture, and describe the I/O subsystem of the BG/P. The ALCF resources described are used for the system evaluation in this paper.

2.1 Argonne Leadership Computing Facility

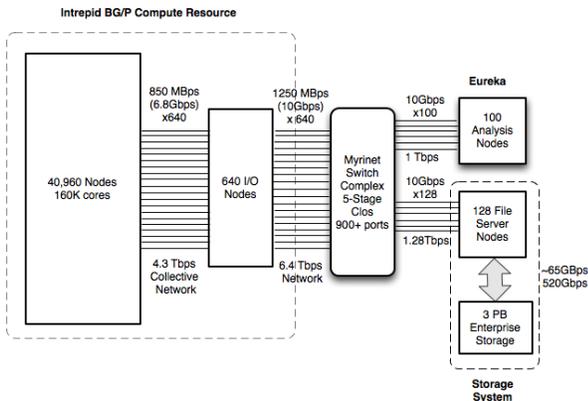


Figure 1: The Argonne Leadership Computing Facility (ALCF) maintains a 160K core BG/P compute cluster (Intrepid), data analysis cluster (Eureka), and the file server nodes all interconnected by a 5-stage Myricom 10G Ethernet switch complex, and other compute infrastructure including several test systems and a large computing cloud resource.

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 connected by a large Myricom switch complex.

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. 64 nodes are grouped into a *pset*, and 16 *psets* together form a rack consisting of 1024 nodes i.e. 4096 cores.

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 communication between compute nodes (CNs), while a global collective 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, the I/O forwarding nodes are referred to simply as I/O nodes (IONs) and have the same architecture as the CNs. 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. A CN is connected to each of its six neighbors in the 3D Torus via a 425 MBps link in each direction, and to two neighbors via the

global collective networks over a 850 MBps link.

The IONs are connected to an external 10 gigabit Ethernet switch, which provides I/O connectivity to file server nodes (FSNs) of a cluster-wide file system as well as connectivity to Eureka, the data analysis (DA) cluster nodes. Eureka 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 is a dual-core dual-processor AMD Opteron with 8 GB RAM per core. Each FSN is connected to the Myricom switch complex over 10 Gbps. The FSNs are connected via InfiniBand 4X DDR to 16 Data Direct Network 9900 storage devices. The production filesystem is a 4.5 TB GPFS parallel filesystem shared between Intrepid and Eureka.

2.2 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. Control and I/O Daemon (CIOD) [15] is the BG/P I/O-forwarding infrastructure provided by IBM. 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 tree network and copies them to a shared-memory region. The CIOD daemon then communicates with the proxy process using this shared memory, which in turn executes the I/O function.

2.3 Prior findings with the BG/P I/O system

In [20], we reported on the performance of BG/P’s I/O forwarding mechanisms. The theoretical achievable collective network throughput from CNs in a *pset* to their designated ION, taking into account 16 bytes of header information for the CIOD I/O forwarding for every 256-byte payload, as well as 10 bytes of hardware headers related to operation control and link reliability—is ≈ 731 MiBps.

As we increase the message size, the ratio of the time spent by CIOD on the actual data transfer to the time spent on sending the control information increases, thus leading to increased network utilization. For a given *pset* and a message size of 1 MiB, the tree network is able to sustain up to 680 MiBps, or 93% efficiency. For moving data (memory-to-memory) for the CNs in a *pset* to a *DA* node, we are able to sustain an end-to-end throughput of 420 MiBps.

3. PERFORMANCE OF I/O MECHANISMS

MPI collective I/O, POSIX I/O and MPI independent I/O are the primary interfaces used by applications running on leadership class systems to interact with the filesystem. High-level libraries such as Parallel-netCDF (pnetcdf) [12] and HDF5 [9] provide a much more convenient abstraction for computational scientists interacting with storage, because they focus on familiar constructs: variables with multiple dimensions and attributes of the variables. These libraries map variables into regions of a file and record information on those variables in a self-describing format, and

typically use the aforementioned file access interfaces to interact with files.

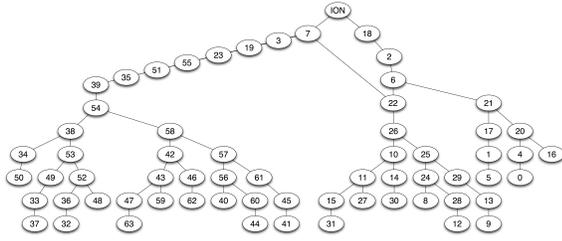


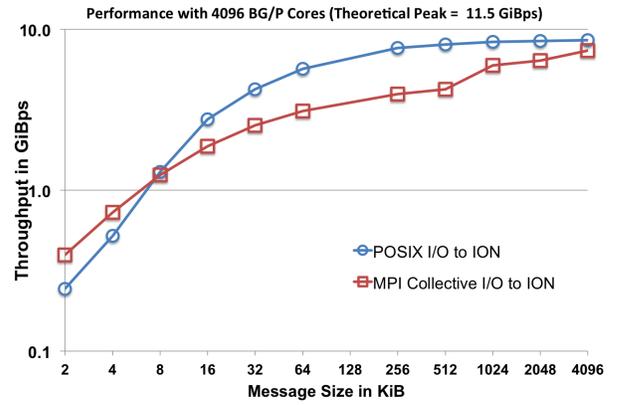
Figure 2: Collective network topology of the CNs in a BG/P *pset*.

Figure 2 shows the collective network topology connecting the CNs in a *pset* to the ION. The collective network architecture design takes into account multiple factors including optimizing global collective operations that are latency bound, I/O operations that are bandwidth intensive, and accounting for a scalable cost-effective mechanism to interconnect BG/P boards. The ION is connected to the file servers and analysis nodes over the Myricom 10G Ethernet network. On BG/P, MPI independent I/O defaults to POSIX I/O. POSIX I/O uses the collective network to move data from the CNs via the ION over to the external I/O network. MPI collective I/O uses a three-phase procedure to move the data out. In MPI collective I/O, in each *pset*, 8 processes are assigned to performing aggregation. In the first phase, the buffer offsets and sizes are aggregated on the aggregator nodes using *AllToAllv*. Next, the actual data is aggregated using *AllToAllv* over the collective network. Finally, this data is moved out from the aggregator nodes using POSIX I/O to the ION via the collective network.

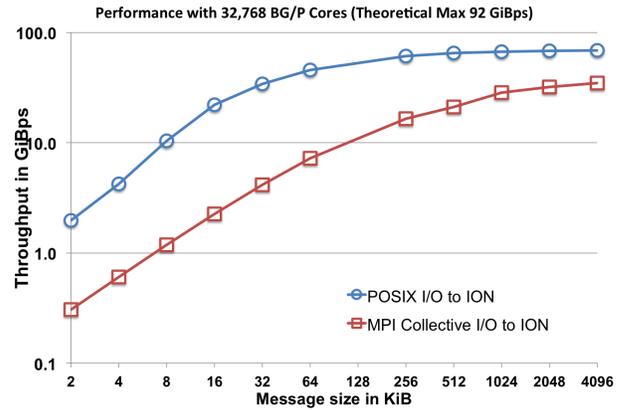
In order to evaluate the performance of the MPI collective I/O and POSIX I/O in moving data out of the IBM BG/P system we wrote a parallel benchmark to write data using Collective I/O and POSIX I/O to /dev/null on the compute nodes. As the I/O operations on the CN are forwarded and executed on the ION, this benchmark effectively measures the achievable throughput of the I/O mechanisms to move data out of the BG/P system without the effects of the storage system. This represents the maximum throughput one could expect by using these interfaces.

Figure 3 (a) compares the performance of the MPI collective I/O and POSIX I/O at 4,096 BG/P cores as we vary the message size per MPI process from 2 KiB to 4,096 KiB. Figure 3 (b) compares the performance of MPI collective I/O and POSIX I/O at 32,768 BG/P core as we vary the message size per MPI process from 2 KiB to 4096 KiB. In both cases, the reported throughput is the average of 5000 iterations. For a 4K message per MPI process at 4,096 cores, collective I/O achieves **6%** of the Peak throughput and POSIX I/O achieves **4%** of the peak. At 64K, the collective I/O achieves **27%** of the peak while POSIX I/O achieves **50%** of the peak. As we increase the message size, the control overhead associated with the CIOD forwarding mechanism to move data out to the ION decreases relative to the data transmission time leading to increased performance.

At 32,768 cores, for a 4K message per MPI process, collective I/O achieves **0.3%** of the peak throughput and POSIX I/O achieves **4%** of the peak. At 64K, the collective I/O achieves **4.5%** of the peak while POSIX I/O achieve **50%** of



(a)



(b)

Figure 3: Performance comparison of I/O mechanisms on the BG/P.

the peak. In case of 32,768 processes, POSIX I/O performs better than collective I/O for all message sizes to move data out of the BG/P system, and, as we increase the number of processes, the collective I/O mechanism is unable to scale. For small messages, both MPI collective I/O and POSIX I/O fail to scale in performance. Investigating this further, we found that in BG/P for collective I/O, the first 8 nodes in a *pset* are designated as aggregators as depicted in Figure 5 (b). Additionally, a considerable amount of time is spent in the aggregation data movement phase as the designated aggregator node assigned could be in another *pset* several hops away. This leads to an increased global communication and network contention in the collective network leading to a decreased performance with collective I/O.

At 4,096 KiB message size per process, MPI collective I/O defaults to using independent I/O (POSIX) for the I/O operations. We would expect the performance of collective I/O mechanisms to be similar to that of the independent I/O. However, in case of collective I/O, the first phase in which the buffer sizes and offsets are exchanged using an *MPIAlltoAllv* still occurs. The overhead of this exchange is small at small core counts; However, it leads to a 100 % reduction in collective I/O performance in comparison to POSIX I/O as we scale to 32K cores.

As we move toward exascale systems where the memory per core is expected to be reduced significantly, scalability for small messages will become a key factor. In this paper, we address scalable end-to-end data movement.

4. GLEAN ARCHITECTURE

In designing GLEAN we are motivated to improve the performance applications that are impeded by their own demanding I/O. We strive to move the data out of the simulation application with as little overhead as possible to the system. To achieve this while providing clean interfaces to existing and future applications, the GLEAN design aims to:

- Exploit the underlying network topology to speed data motion off of the supercomputer,
- Provide asynchronous data I/O via staging nodes,
- Develop scalable mechanisms for collective I/O by reducing synchronization requirements,
- Mitigate variability in I/O performance of shared filesystems using staging,
- Leverage data semantics of applications,
- Enable simulation-time data analysis, transformation and reduction,
- Provide non-intrusive integration with existing applications, and,
- Provide transparent integration with native application data formats.

Figure 4 compares the traditional mechanism for I/O with GLEAN. The simulation running on the compute may invoke GLEAN directly or transparently through a standard I/O library such as parallel-netcdf and hdf5. The data is moved out to dedicated analysis/staging nodes. Using GLEAN, one can apply custom analyses to the data on the compute resource or on the staging nodes. This can help reduce the amount of data written out to storage. On the staging nodes, GLEAN using MPI-IO or higher-level I/O libraries to write the data out asynchronously to storage.

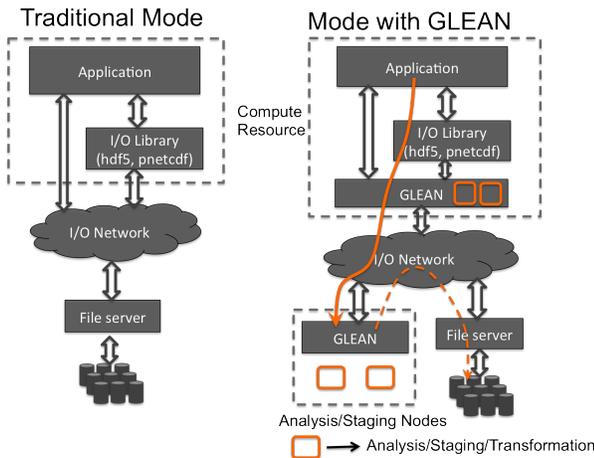


Figure 4: Relationships between GLEAN and principal components of a HPC application.

GLEAN is implemented in C++ leveraging MPI, threads and provides interfaces for Fortran and C-based parallel applications. It offers a flexible and extensible API that can be customized to meet the needs of the application. In the following subsections we describe our GLEAN design in terms of three principal features: network topology, data semantics, and asynchronous data staging.

4.1 Exploiting BG/P network topology for I/O

As we move towards systems with heterogenous and complex network topologies, effective ways to fully exploit their heterogeneity is critical. BG/P has different networks with varying throughputs and topologies. The 3D torus interconnects a compute node with its six neighbors at 425 MB/s over each link. In contrast, the collective network is a shared network with a maximum throughput of 850 MBps to the ION. The collective network is the only way to get to the ION in order to perform I/O. BG/Q is expected to have a more complex network topology. An important goal in GLEAN is to leverage the topologies to move the data out of the machine as soon as possible enabling the simulation to continue on with its computation.

MPI collective I/O uses *AllToAllv* wherein, depending on an aggregator’s rank, the aggregation traffic can cross the *pset* boundaries and, this leads to global communication and network contention. In GLEAN, we restrict the aggregation traffic to be strictly within the *pset* boundary. The goal is to move the data out of the system as fast as possible to the staging nodes and use the staging nodes to perform any data shuffling and reduce any global communication critical for scaling to large core counts. Figure 5(a) shows GLEAN’s selection of 8 aggregator groups formed within a *pset*. Each group is depicted by a distinct color and the aggregator node for the group is highlighted. For comparison 5(b) shows the 8 aggregator nodes used in MPI I/O collective operations. In this case, the first 8 nodes are selected as aggregators and highlighted. We would like to note that the aggregation traffic in the MPI collection I/O is global and not restricted to a *pset*.

In each aggregator group in GLEAN, the node where the aggregation is performed is chosen such that the aggregator nodes are distributed across the collective network in a *pset* (Figure 2). We first measured the achievable throughput from a single CN to the ION and found no significant performance variation from various points in the collective network. We did not, however, perform the analogous multi-point test to find the cost of contention and plan to investigate this in future. Assuming that this might be an issue when many CNs are sending data to the ION, we choose to distribute the GLEAN aggregator nodes across the collective network in a *pset*.

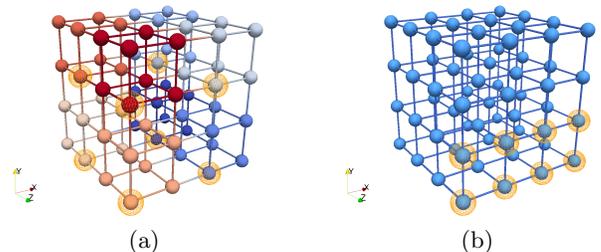


Figure 5: Aggregation groups formed within a *pset* by GLEAN (a) with aggregator nodes highlight, and aggregator node configuration used by MPI I/O (b).

To create the aggregators, for each *pset*, we use an MPI sub-communicator giving us only the MPI processes belonging to the *pset*. We split the sub-communicator into aggregation sub-groups and created a communicator for each sub-group. We would like to note that these communicators

are created once at initialization and reused during an entire simulation run. To create the aggregation groups, we leveraged the BG/P "personality" information to get the X,Y,Z rank in the 3D torus of the each process and group the nearest neighbors in all 3 dimensions. The data aggregation also includes the associated data semantics. Once the data is aggregated, the aggregator nodes send the data out to the staging nodes via the IONs.

4.2 Leveraging application data semantics

A key design goal in GLEAN is to make application data semantics a first-class citizen. This enables us to apply various analytics to the simulation data at runtime to reduce the data volume written to storage, transform data on-the-fly to meet the needs of analysis, and enable various I/O optimizations leveraging the application's data models. Toward this effort, we have worked closely with FLASH [7], an astrophysics application, to capture its adaptive mesh refinement (AMR) data model. We have interfaced with PHASTA [18], which uses an adaptive unstructured mesh, to make unstructured grids supported in GLEAN. Additionally, we have mapped Parallel-netCDF and hdf5 APIs to relevant GLEAN APIs, thus enabling us to interface with simulations using pnetcdf and hdf5. We have worked with many of the most common HPC simulation data models ranging from AMR grids to unstructured adaptive meshes.

4.3 Asynchronous data staging

Asynchronous data staging refers to moving the application's I/O data to dedicated nodes and next writing this out to the filesystem asynchronously while the application proceeds ahead with its computation. A key distinguishing characteristic of GLEAN's data staging is that it leverages the data models and semantics of applications for staging instead of viewing data simply as files and/or buffers. On the staging nodes, typically the Eureka analysis cluster nodes, GLEAN runs as an MPI job and communicates with the GLEAN aggregator nodes over sockets, as sockets are the only way to communicate between BG/P CNs and the external I/O network. A key requirement is to scale to the large number of connections from BG/P. In the case of 8 GLEAN aggregators per *pset*, an entire machine run (160K cores - 640 *psets*) will have 5,120 connections. These connections are distributed among various GLEAN staging nodes. Further, each staging node is designed with a thread-pool wherein each thread handles multiple connections via a poll-based event multiplexing mechanism. Asynchronous data staging blocks the computation only for the duration of copying data from the CN to the staging nodes. The data staging serves as burst buffer for the simulation I/O that can be written out asynchronously to the filesystem while the computation proceeds ahead. Data staging also significantly reduces the number of clients seen by the parallel filesystem, and thus mitigates the contention including locking overheads for the filesystem. The data semantics enables GLEAN to transform the data on-the-fly to various I/O formats. On the staging nodes, we envision GLEAN to be deployed either as an "always ON" service, run on dedicated set-aside nodes of a simulation or co-scheduled along with the simulation. If there is insufficient memory on the staging nodes (receivers), the transfer is blocked until sufficient buffer/memory space is made available. We are currently pursuing the option of using node-local storage to stage data.

5. MICROBENCHMARKS

We describe a series of experiments to evaluate the performance of GLEAN using synthetic I/O loads.

5.1 Data movement from compute nodes

Our goals were to determine the number of aggregators that maximize performance, to compare performance the performance of GLEAN with MPI collective I/O and POSIX I/O as we vary the message size and number of Intrepid cores sending data, and to test GLEAN strong scaling.

5.1.1 Optimizing the number of aggregators

Having determined the best placement of aggregator nodes in the collective network topology, it is important to determine the best number of aggregator nodes per *pset*. Figure 6 depicts the total throughput as a function of message size as we move data from the BG/P CNs to the IONs for 1024 BG/P cores. The baseline performance achieved by POSIX I/O and by MPI collective I/O is shown for comparison.

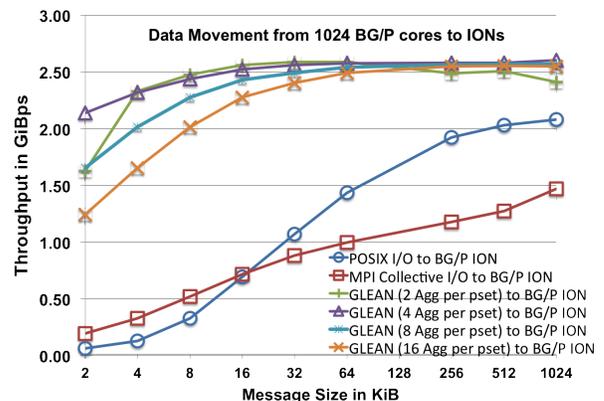


Figure 6: Comparing performance of GLEAN for varying number of aggregators (log-linear scale).

Note that four aggregators give the best performance across the measured message sizes. With only two aggregators, throughput falls short at small message sizes as the number of ION threads processing the data is unable to sustain the higher throughput due to the low clock speed of the ION. At large message sizes, the two aggregator configuration begins to underperform because the aggregated memory required increases and the amount of network parallelism is restricted to only two aggregators. For even larger message sizes, we expect the same drop in performance to happen with four aggregators. At message sizes larger than 64K per core, the performance difference as we vary the number of aggregators from four to sixteen is not very significant. By having more aggregators, the amount of memory needed for the aggregation buffer reduces, and, in our final implementations, we adopt an adaptive scheme where larger message sizes invoke larger numbers of aggregators. We believe that in future systems, auto-tuning the number of aggregators based on the message sizes as well as the processing capabilities of the IONs will be critical in the design of collective I/O operations.

5.1.2 Performance with number of BG/P cores

Because GLEAN is designed to send data within a *pset* without interfering with other *psets*, we expect GLEAN to

scale perfectly as we scale the number of cores for this test. In this test, the data is moved only to the IONs and is not affected by the external I/O network. But we do want to benchmark GLEAN’s performance to the I/O mechanisms at scale for various message sizes to demonstrate the efficacy of our approach.

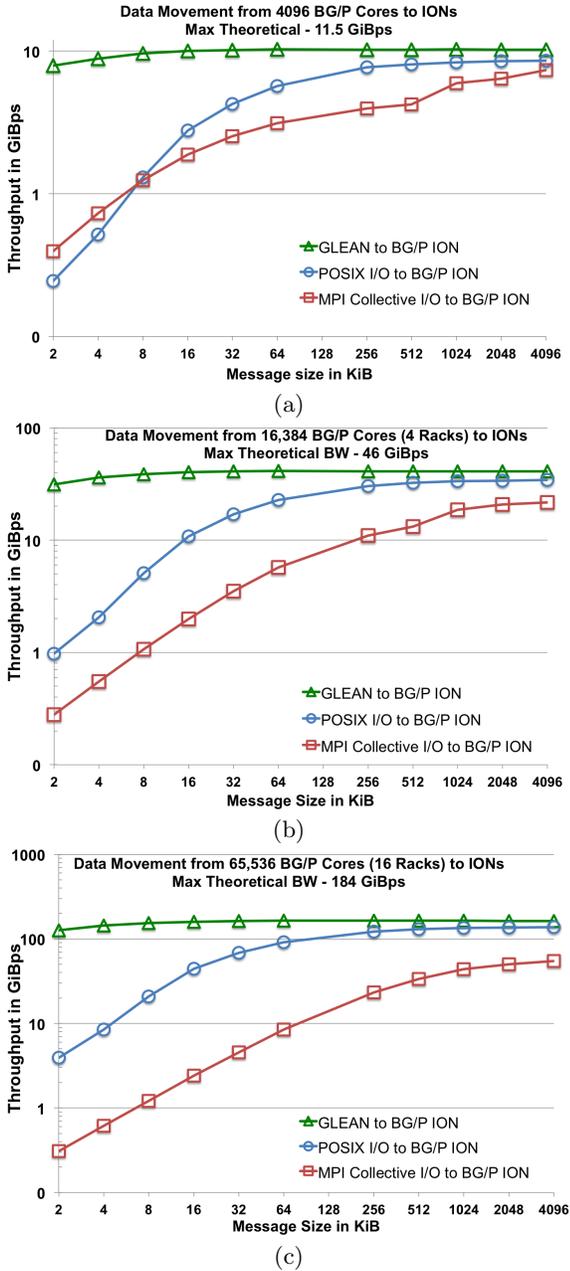


Figure 7: Performance comparison of I/O mechanisms to move data out to the IONs for various message sizes per core at 4K, 16K and 64K BG/P cores (log-log scale).

Figure 7 depicts the results of these experiments for 4,096 cores (1 BG/P Rack), 16,384 cores (4 racks) and 65,536 cores (16 racks). Each plot shows aggregate throughput in GiBps against the message size per MPI process in KiB. GLEAN shows the expected perfect scaling from 4,096 cores to 65,536

cores for each message size from 2 KiB to 4,096 KiB. POSIX I/O approaches GLEAN performance at large message size and, like GLEAN, it scales perfectly as the various *psets* do not communicate with one another. MPI collective I/O approaches GLEAN performance more slowly at higher message sizes, but fails to scale – evidence of the aggregation being done across *psets*.

For small message sizes, the impact on strong scaling is most pronounced. GLEAN achieves up to **30-fold** improvement in throughput over POSIX I/O and up to **400-fold** improvement over MPI collective I/O. For the specific case of moving 1 GiB of data from 256 cores up to 65,536 cores, the strong scaling figures are shown in Figure 8. GLEAN achieves nearly perfect strong scaling.

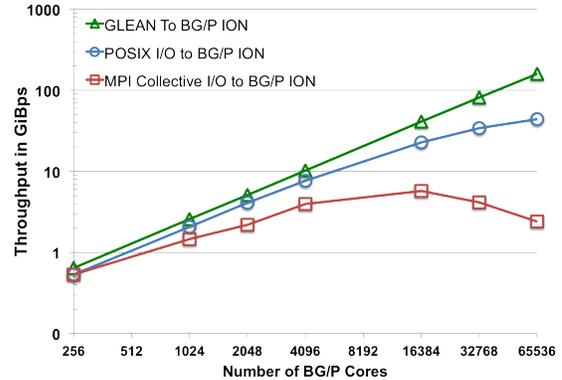


Figure 8: Strong scaling performance of the I/O mechanisms to write 1 GiB data to the BG/P IONs (log-log scale).

5.2 End-to-end performance

Our final experiments with synthetic benchmarks were designed to evaluate the end-to-end performance of GLEAN in moving data from Intrepid to the Eureka staging nodes.

5.2.1 Scaling with the number of staging nodes

In [20], we employed a thread pool mechanism to efficiently move data out of the IONs. We employ a similar thread pool mechanism on the staging nodes to efficiently receive the data from Intrepid. By performing extensive tuning, we found four threads per node provide us with best performance. Next, we evaluated the number of staging nodes needed to efficiently to receive the data from the Intrepid system across the Myricom network. In this case, we determine that the Myricom I/O network infrastructure effects plays a key role.

In the previous section, we demonstrated how to optimally send data out from Intrepid compute nodes with the result that one rack of Intrepid can sustain a maximum of 6.7 GiBps out over the external I/O network - This limit is set by the forwarding performance of CIOD on the ION. Using *nuttcp*, a network performance benchmark, we determined that GLEAN on a Eureka node can sustain a maximum performance of 670 MiBps. Thus we would expect 10 Eureka nodes to receive this data in ideal network conditions.

In Figure 9, we present the data movement performance from 16,384 Intrepid cores to the data staging nodes (Eureka). Ideally, Intrepid could send a maximum of 26.8 GiBps.

We see from the graph that as we add Eureka nodes, we are able to sustain additional aggregate throughput. The performance flattens out between 36 and 48 Eureka nodes to 22.1 GiBps indicating that we are hitting the network routing limit of the CLOS-based Myricom network to the subset of Eureka nodes.

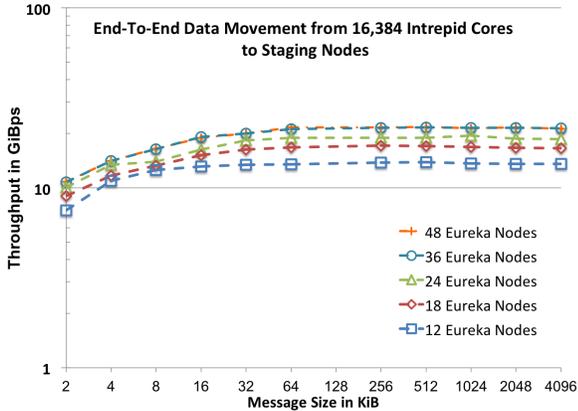


Figure 9: Scaling with number of staging nodes.

5.2.2 End-to-end performance

We present a mix of end-to-end performance tests to benchmark GLEAN at scale. Figure 10 and Table 1 summarizes tests scaling from 4,096 cores (1 BG/P rack) to 131,072 cores (32 BG/P racks) to various fractions of Eureka nodes. In selecting the number of Eureka nodes to receive the data, we tried to roughly match 10 Eureka nodes to each BG/P rack until we ran out of Eureka nodes. We have included columns in the table for simple extrapolations of performance from the earlier measurements. Note that in the one and two rack cases, we expect to be limited by Intrepid’s outgoing BW and in the four to 32 rack cases, we expect Eureka to be increasingly responsible for limiting the throughput. The final column of the table shows that we are achieving over **80%** of the expected throughput in all of the cases. At 132K Intrepid cores, we are able to sustain **54 GiBps** of aggregate throughput to the staging nodes.

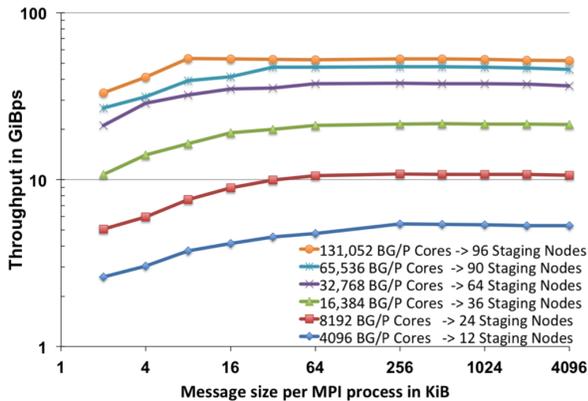


Figure 10: End-to-end performance of GLEAN between Intrepid and Eureka.

Intrepid Cores (Racks)	Eureka Nodes	Intrepid Capacity [GiBps]	Eureka Capacity [GiBps]	Measured [GiBps]	Efficiency
4096 (1)	12	* 6.7	8.0	5.4	0.80
8192 (2)	24	* 13.4	16.1	10.7	0.80
16384 (4)	36	26.8	* 24.1	21.6	0.89
32768 (8)	64	53.6	* 42.9	37.4	0.87
65536 (16)	90	107.2	* 60.3	47.0	0.78
131072 (32)	96	214.4	* 64.3	52.4	0.81

Table 1: End-to-end throughput for select configurations.

In a separate experiment, we achieved 84 GiBps between Intrepid using the 128 fileservers nodes and 100 Eureka nodes for staging. In this experiment, we did not have the efficient thread-pool receiving mechanism implemented on the staging nodes. We believe with that improvement in place, we could now achieve 126 GiBps. It is clear that the CLOS network is capable of more than can be delivered to Eureka alone, a fact that could lead to better I/O performance for Intrepid in the future.

6. EVALUATION WITH APPLICATIONS

We have integrated GLEAN with several applications and report its performance at leadership class scale with realistic data patterns. The applications selected span a wide gamut of data models, and enabled us to experiment with several integration schemes. In the following section, we describe our methods and results for GLEAN as used by three applications: FLASH, S3D, and PHASTA on leadership-class systems.

6.1 FLASH

FLASH code [7] is an adaptive mesh, parallel hydrodynamics code developed to simulate astrophysical thermonuclear flashes in two or three dimensions, such as Type Ia supernovae, Type I X-ray bursts, and classical novae. It solves the compressible Euler equations on a block-structured adaptive mesh. FLASH provides an Adaptive Mesh Refinement (AMR) grid using a modified version of the PARA-MESH package and a Uniform Grid (UG) to store Eulerian data.

For our study, we used the Sedov simulation included in the FLASH distribution. Sedov evolves a blast wave from a delta-function initial pressure perturbation [6]. The Sedov problem exercises the infrastructure (AMR and I/O) of FLASH with the minimal use of physics solvers. It can, therefore produce representative I/O behavior of FLASH without spending too much time in computations. We ran the application in 3D with 16^3 cells per block. Each block consists of 10 mesh variables, and the problem size is controlled by adjusting the global number of blocks. We choose to advance four time steps and produce I/O output at every single step so that most application runtime is mostly spend on I/O. The I/O in each step consists of checkpoint files for restart purposes and plot files for analysis. A checkpoint file is a dump of the complete state of a runtime application, including mesh data in double precision. A plot file is a user-selected subset of mesh variables stored in single precision. In these experiments checkpoint I/O writes all 10 mesh variables. The plot file I/O writes only selected variables of interest (in these experiments, the 1st, 6th, and 7th variables).

We evaluate the performance of pnetcdf, hdf5 and GLEAN

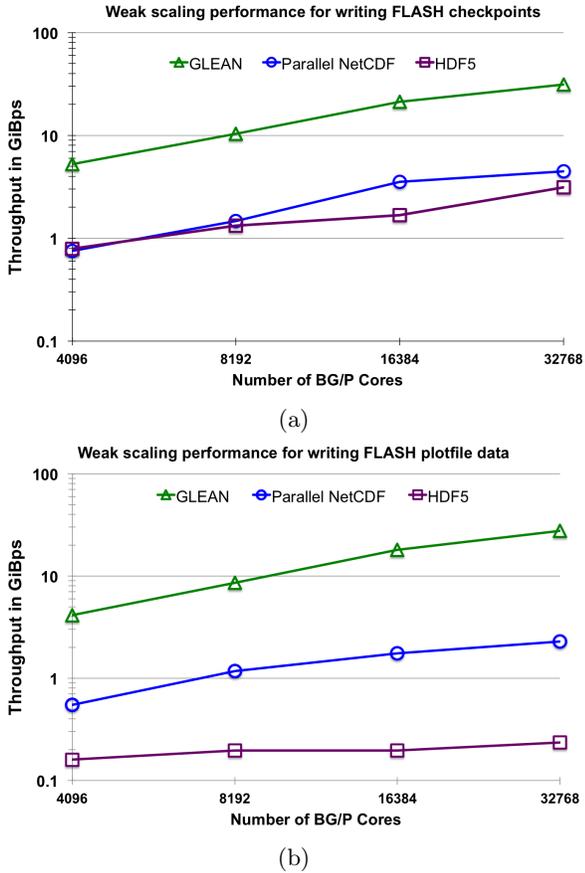


Figure 11: Weak scaling results for FLASH.

for two standard FLASH output streams: checkpoint and plot data. We configured FLASH to use both pnetcdf and hdf5 with collective I/O to the GPFS parallel filesystem on Intrepid. It was demonstrated in [11] that the collective interfaces outperformed independent I/O interfaces for the given FLASH Sedov simulation on ALCF systems. Working closely with the FLASH developers, we designed GLEAN to have an API to capture the data semantics of FLASH including the AMR hierarchy. To interface with FLASH, we transform pnetcdf into appropriate GLEAN API calls. By setting an environment variable that is passed to the flash executable when the job is launched with pnetcdf, one can use GLEAN with FLASH. Thus, we are able to integrate with FLASH without modification to the FLASH simulation code. On the staging nodes (Eureka), GLEAN can be configured to write data out asynchronously using either pnetcdf, or transformed on-the-fly to hdf5 and written out. This is possible because GLEAN captures the data semantics of FLASH. As we scaled from 4,096 cores to 32,768 cores in powers of two, the number of Eureka nodes used for staging data in GLEAN was 12, 24, 36 and 72 respectively. We would like to note that using GLEAN, we wrote FLASH checkpoint and plot files to the GPFS parallel filesystem and verified that the generated files were consistent with the files written directly without GLEAN.

In our weak scaling study, each run wrote five checkpoints and six plotfiles. For 4,096 cores, each checkpoint file is 12 GiB, and for 32,768 cores, it is 96 GiB. From Figure 11 (a), we see that the achievable throughput for pnetcdf and

hdf5 increases as we weak scale. This is because the number of IONs involved increases. The maximum throughput achieved by hdf5 is 3.13 GiBps and pnetcdf is 4.4 GiBps. GLEAN, using the pnetcdf interface, is able to sustain an observed throughput of **31.3 GiBps** - a **10-fold** improvement over hdf5 and a **7-fold** improvement over pnetcdf. The observed throughput is the throughput achieved to transfer the data to the staging nodes. We argue that this is the throughput seen by the application as the data is written asynchronously to the filesystem. We also performed experiments to synchronously write the data to the parallel filesystem, and at 32K cores with 72 staging nodes, we achieve a storage throughput of **16.3 GiBps** - a **4-fold** end-to-end improvement to storage. This higher throughput achieved is primarily due to the reduced number of clients seen using staging and demonstrates end-to-end I/O acceleration using GLEAN. As we can overlap the writing to the filesystem with the application's computation once the data is staged, we limit the focus of the remaining results in this paper to the throughput achieved to the staging nodes.

For plot file data, at 4,096 cores, each file is 1.8 GiB, and at 32,768 cores, it is 16 GiB each. In plot files, the data written per process is smaller than checkpoint data. At 32,768 cores, GLEAN is able to sustain an observed throughput of **27.2 GiBps** - a **117-fold** improvement over hdf5 and a **12-fold** improvement over pnetcdf. In the case of plot files, the amount of data written per process is less than that written in checkpoint files. At these scales, the efficacy of GLEAN becomes even more evident.

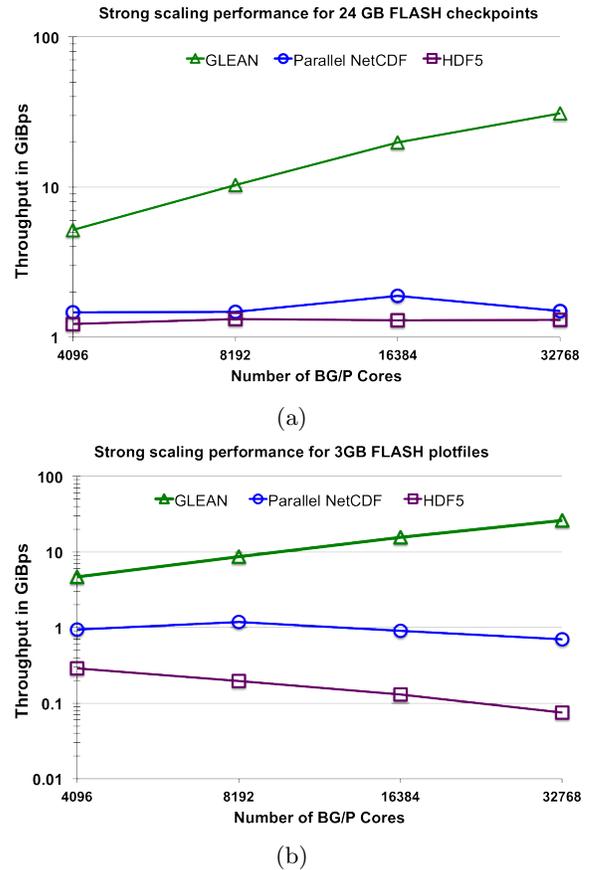


Figure 12: Strong scaling results for FLASH.

Figure 12 depicts the strong scaling performance for writing out 3.4 GiB plot files and 24 GiB checkpoint files as we scale the number of processes from 4,096 to 32,768 processes. For plot files, at 4,096 processes, GLEAN achieves a **5-fold** improvement over pnetcdf and **16-fold** improvement over hdf5. At 32,768, GLEAN achieves a **37-fold** improvement over pnetcdf and a **347-fold** improvement over hdf5 and sustains an average throughput of 26 GiBps. In the case of checkpoint files, GLEAN is able to sustain **31 GiBps**. Thus, effectively exploiting the network topology of the BG/P, leveraging the data semantics of the applications and asynchronous data staging are critical as we scale towards larger core counts. These will be of paramount importance as we move towards future exascale architectures with larger core counts and smaller per-core memory footprints.

6.2 S3D

The S3D application [3] simulates turbulent combustion using direct numerical simulation of a compressible Navier-Stokes flow. It uses a structured grid, with variables stored separately, one block per variable over the whole grid. The variables are declared as 3D or 4D arrays. The domain is decomposed among MPI processes in 3D. Periodically, all processes participate in writing out a restart file. This restart file can be used to resume computation and as input for visualization and analysis tools. S3D-IO extracts just the portion of S3D concerning restart dumps, allowing us to focus only on I/O characteristics.

S3D supports multiple output methods, including MPI collective I/O and pnetcdf. In order to better understand the API requirements in GLEAN and to interface with Fortran-based simulations, we added a new I/O method in S3D-IO to use GLEAN. This integration took about two days and was valuable in helping design GLEAN interfaces for Fortran-based applications. We present scaling results from 4,096 BG/P cores (one rack) to 32,768 BG/P cores (eight racks) with MPI collective I/O, pnetcdf and GLEAN. In the case of GLEAN, data was transferred from Intrepid to Eureka, and could be written out asynchronously using pnetcdf. As we scaled from 4,096 cores to 32,768 cores in powers of two, the number of Eureka nodes used for staging data in GLEAN used was 12, 24, 36 and 72 respectively. In these runs, S3D data consists of 4 variables, namely, species (11 elements), pressure, temperature, velocity (three components).

Figure 13 (a) depicts the weak scaling of the various I/O mechanisms for a block size of $22 \times 36 \times 22$ per MPI process. This block size is representative of the recent S3D production runs on Jaguar at OLCF and the Hopper2 system at NERSC. In the case of 32,768 processes, the size of each checkpoint file is 68GiB, and we wrote out 20 time-steps, producing a total data of 1.38 TiB for each run. The throughput performance reported is the average throughput achieved for the 20 time-steps. From the Figure 13 (a), we see that MPI collective I/O achieves a higher throughput with respect to pnetcdf, as in S3D, this I/O mechanism does not write out any metadata to the file. Therefore, this leads to a 1.5 fold improvement in performance over pnetcdf. MPI collective I/O and pnetcdf demonstrate multi-GiBps throughput as S3D writes large messages sizes per process in this weak scaling experiment. GLEAN is able to achieve 5.1 GiBps at 4,096 cores - A 2.5 fold improvement over MPI Collective I/O and 3.5 fold speed up over pnetcdf.

At 32,768 cores, GLEAN achieves an observed throughput of **26.2 GiBps** - a **4.5 fold** improvement over MPI Collective I/O and **5.8 fold** improvement over pnetcdf.

Figure 13 (b) depicts the strong scaling performance for the various I/O mechanisms for a total 3D domain size of $512 \times 512 \times 512$. We varied the number of BG/P cores from 4,096 to 32,768, thereby, strong scaling the block size per process from $32 \times 32 \times 32$ (4,096 cores) to $16 \times 16 \times 16$ (32,768 cores). The data size per time-step is 16 GiB, and we configured S3D to write out 20 time-steps to generate a total of 320 GiB of data for each run. GLEAN is able to achieve an observed throughput of 4.8 GiBps at 4,096 cores - a 2-fold improvement over both MPI Collective I/O and pnetcdf. At 32,768 cores, GLEAN achieves **25.7 GiBps** - a **9-fold** improvement over MPI collective I/O and **13-fold** improvement over pnetcdf. Thus, we observe the it is critical to incorporate topology-aware data movement and asynchronous data staging in order to achieve strong scaling - a key requirement in future systems.

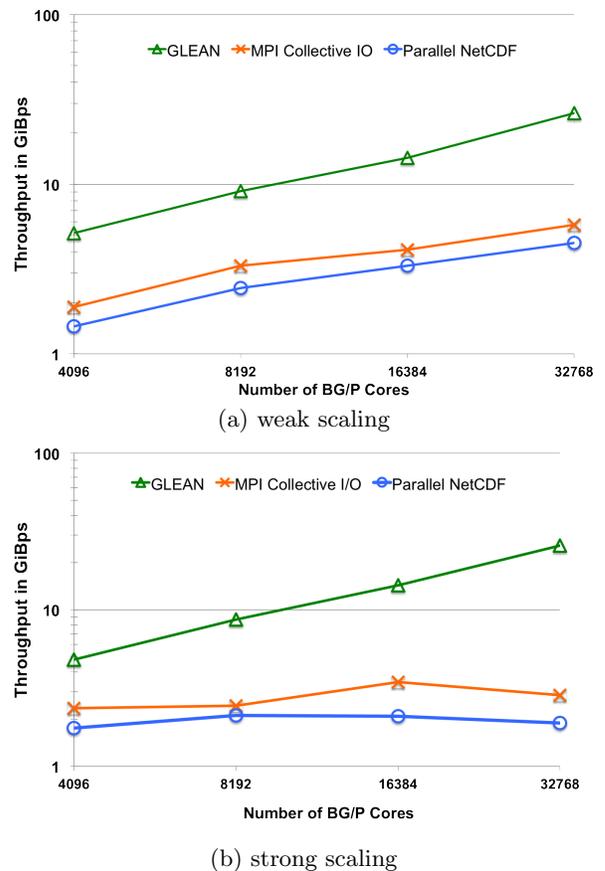


Figure 13: Scaling results for S3D I/O.

6.3 PHASTA

The PHASTA code [18] performs computational fluid dynamics (CFD) using a finite element discretization using a LES-based turbulence model. PHASTA uses an adaptive, unstructured tetrahedral grid. The number and locality of elements changes frequently, based on solution characteristics and load balancing. Grid and field structures are stored in dynamically-allocated memory, due to frequent adaptivity updates.

We describe our success with GLEAN at enabling simulation-time analysis and visualization of PHASTA running on 128K cores of Intrepid (32 racks) with ParaView [10] running on 80 Eureka nodes. To achieve this we worked closely with PHASTA developers to capture its unstructured tetrahedral mesh data model. Additionally, to facilitate visualization of the PHASTA data using ParaView, we added support for ParaView’s visualization meshes. On Eureka, GLEAN was able to transform on-the-fly PHASTA’s staged data into ParaView’s mesh format. ParaView is one of the highly used visualization toolkits on leadership class systems. Simulations using GLEAN would be able to visualize data at run-time.

We briefly describe two scaling studies for co-visualization of the PHASTA simulation with ParaView enabled by GLEAN. For 416 million element case of PHASTA on 32 Intrepid racks consisting of 20 GiB, GLEAN was able to transfer this data in 0.6 secs achieving **34 GiBps**. For 3.32 billion elements (160 GB) using 32 racks and 80 Eureka nodes, GLEAN achieved **41 GiBps**. GLEAN enabled PHASTA developers to visualize live a 128K cores simulation.

7. RELATED WORK

In RBIO for BG/P [8], a dedicated set of BG/P compute nodes per *pset* are set aside to provide a cache for checkpointing data. This is achieved by splitting the main MPL-COMM-WORLD into a communicator for caching data and another for the simulation. The caching nodes do not participate in the simulation’s computation. Various MPI collective optimizations that rely on the global communicator are no longer available to the simulation. Additionally, RBIO requires changes to a simulation to use the RBIO API.

I/O delegation [16] is a portable MPI-IO layer where certain tasks, such as file caching, consistency control, and collective I/O optimization are delegated to a small set of compute nodes, collectively termed as I/O delegate nodes. A collective cache design is incorporated to resolve cache coherence and alleviate the lock contention at I/O servers. I/O delegation resides at the ROMIO layer and does not need an application to be modified. On BG/P, I/O delegation could use GLEAN to move data onto the delegate nodes. Additionally, These nodes could be part of the staging area, thus enabling the simulation to have access to all the cores and optimized collectives.

The ADIOS high-level I/O library [13, 2] has demonstrated high-performance I/O for applications on large-scale systems. To use ADIOS, applications need to be modified via a lightweight API. Additionally, ADIOS writes data into a custom BP format. The library provides several post-processing tools to convert the BP format into formats including HDF5 and netCDF. As the I/O bottleneck becomes even more critical in future, this conversion time could become significant. In GLEAN, we support the commonly used I/O libraries and do not need any offline data conversion. ADIOS supports asynchronous data staging via decoupled and asynchronous remote transfers (DART) [4] and DataStager [1]. DART is an asynchronous communication and data transport substrate for large-scale parallel applications. It uses one-sided communication mechanisms including RDMA for extracting and transporting data. To the best of our knowledge, these mechanism supports multi-dimensional arrays. In GLEAN, we strive to capture a range of data models including AMR and unstructured grids.

Active buffering with threads [14] 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, performing 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.

8. CONCLUSIONS

The performance mismatch between the computing and I/O components of current-generation HPC systems has made I/O the critical bottleneck for scientific applications. It is therefore crucial that software take every advantage available in moving data between compute, analysis, and storage resources as efficiently as networks will allow. Currently available mechanisms often fail to perform as well as the hardware infrastructure would allow, suggesting that improved optimization and perhaps adaptive mechanisms deserve increased study.

Our evaluation of I/O performance in IBM Blue Gene/P shows that significant improvements are possible by optimizations driven by the topological strengths and weaknesses of available networks. In particular, we find that there are significant opportunities to (1) improve performance of data movement from CNs to IONs on each pset, and (2) exploit the combined bandwidth from BG/P system to the analysis and the filesystem nodes. We have developed an infrastructure called GLEAN as our vehicle by which we develop, demonstrate and deploy these improvements. The GLEAN infrastructure hides significant details from the end user, while at the same time providing them with a flexible interface offering the fastest path to their data and in the end scientific insight. GLEAN provides a mechanism for leveraging topology-aware data movement for accelerating I/O, interfacing to running simulations for co-analysis, and an interface for in situ analysis with minimal modifications to the existing application code base. By fully exploiting the network topology of BG/P, GLEAN achieves both weak and strong scaling to move data out from the BG/P system. In our evaluations with applications at scale, GLEAN is able to sustain an observed throughput of upto 31 GiBps for checkpointing data and up to 41 GiBps for simulation-time visualization at 128K cores on Intrepid BG/P.

As we move toward systems with heterogeneous and complex network topologies, effective ways to fully exploit their heterogeneity is critical. The topology-aware mechanisms and reduced synchronization requirements of GLEAN can be used to optimize the performance of MPI-IO implementations on various platforms. This will greatly benefit the higher-level I/O libraries including hdf5, pnetcdf and ADIOS built on top of MPI-IO. GLEAN has been ported to laptops running OSX and Linux, Linux-based clusters and BG/P systems. We are currently working on porting GLEAN to Cray stems and making our topology-aware mechanisms generic. Additionally, we plan to incorporate the topology-aware data movement optimizations to the MPI-IO layer on BG/P to allow the broader community of applications to benefit from our efforts (including hdf5 and pnetcdf) and to make our topology-aware mechanisms more generic. 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.

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 thank Wei-Keng Liao for comments related to the paper and Jen Salazar for helping edit the paper. We acknowledge Joseph Insley, Rob Latham, Rob Ross, Phil Carns, Randy Hudson, Cal Jordan, Chris Daley, Kamil Iskra, Valerio Pascucci, Jackie Chen, Ray Grout, Pat Marion, Berk Geveci, Ken Jansen, Michel Rasquin, Ben Matthews, Bill Allcock, Susan Coghlan, Tisha Stacey, Ray Loy, Kevin Harms, Andrew Cherry, and the ALCF team for discussions and help related to the paper.

9. REFERENCES

- [1] 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.
- [2] ADIOS. <http://www.nccs.gov/user-support/center-projects/adios/>.
- [3] R. Sankaran C. S. Yoo and J. H. Chen. Three-dimensional direct numerical simulation of a turbulent lifted hydrogen jet flame in heated coflow: flame stabilization and structure. *Journal of Fluid Mechanics*, pages 453–481, 2009.
- [4] C. Docan, M. Parashar, and S. Klasky. Enabling high speed asynchronous data extraction and transfer using DART. *Proceedings of the 17th International Symposium on High-Performance Distributed Computing (HPDC)*. IEEE Computer Society Press, Boston, MA, 2008.
- [5] Jack Dongarra, Pete Beckman, and Terry Moore. International exascale software project roadmap. Technical report, DOE and NSF, November 2009.
- [6] FLASH user guide. <http://flash.uchicago.edu/website/>.
- [7] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, and H. Tufo. FLASH: An adaptive mesh hydrodynamics code for modelling astrophysical thermonuclear flashes. *Astrophysical Journal Supplement*, 131:273–334, 2000.
- [8] J. Fu, O. Sahni, K. Jansen, M. Shephard, and C. Carothers. Scalable Parallel I/O Alternatives for Massively Parallel Partitioned Solver Systems. In *Proceedings of the Workshop on Large-Scale Parallel Processing in conjunction with the IEEE International Parallel and Distributed Processing Symposium*. IEEE, April 2010.
- [9] HDF Group. HDF5: Hierarchical Data Format. <http://www.hdfgroup.org/HDF5>.
- [10] Kitware. ParaView. <http://www.paraview.org/>.
- [11] R. Latham, C. Daley, W.-K. Liao, K. Gao, R. Ross, A. Dubey, and A. Choudhary. A Case Study for Scientific I/O: Improving the FLASH Astrophysics Code. <http://www.mcs.anl.gov/uploads/cels/papers/P1819.pdf>.
- [12] Jianwei Li, W. Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. Parallel netCDF: A high-performance scientific I/O interface. In *ACM/IEEE Conference on Supercomputing*, Phoenix, AZ, Nov. 2003.
- [13] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Adaptable, metadata rich io methods for portable high performance io. In *In Proceedings of IPDPS'09, May 25-29, Rome, Italy*, 2009.
- [14] 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.
- [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] Arifa Nisar, Wei keng Liao, and Alok Choudhary. Scaling parallel I/O performance through I/O delegate and caching system. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.
- [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] O. Sahni, M. Zhou, M. Shephard, and K. Jansen. Scalable Implicit Finite Element Solver for Massively Parallel Processing With Demonstration to 160K Cores. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, Portland, Oregon, 2009. ACM.
- [19] Vivek Sarkar. Exascale software study: Software challenges in extreme scale systems. Technical report, DARPA, September 2009.
- [20] Venkatram Vishwanath, Mark Hereld, Kamil Iskra, Dries Kimpe, Vitali Morozov, Michael E. Papka, Robert B. Ross, and Kazutomo Yoshii. Accelerating i/o forwarding in ibm blue gene/p systems. In *SC*, pages 1–10, 2010.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.