Scalable parallel I/O on a Blue Gene/Q supercomputer using compression, topology-aware data aggregation, and subfiling

Huy Bui, Jason Leigh
Electronic Visualization Laboratory
University of Illinois at Chicago
Chicago, Illinois, 60612
Email: abui4,spiff@uic.edu

Venkatram Vishwanath, Hal Finkel, Salman Habib,
Katrin Heitmann, Michael Papka and Kevin Harms
Argonne National Laboratory
9700 S. Cass Ave, Argonne, IL 60439
Email: venkat,hfinkel,habib,heitmann,papka@anl.gov, harms@alcf.anl.gov

Abstract—**In this paper, we propose an approach to improving the I/O performance of an IBM Blue Gene/Q supercomputing system using a novel framework that can be integrated into high performance applications. We take advantage of the systems tremendous computing resources and high interconnection bandwidth among compute nodes to efficiently exploit I/O bandwidth. This approach focuses on lossless data compression, topology-aware data movement, and subfiling. The efficacy of this solution is demonstrated using microbenchmarks and an application-level benchmark.**

I. Introduction

I/O remains a major problem in high performance computing systems. Much of the research to date has focused on finding ways to improve I/O performance on a particular system; however, if the problem is examined holistically from the time the data is created at the compute nodes, to when it is written to disk there is a strong argument for combining several approaches to achieving multi-fold performance gains.

In our previous work [1] on the Blue Gene/P system, we leveraged a topology-aware data movement and data staging mechanism to improve I/O by several orders of magnitude. System interconnects have continued to increase in complexity: Blue Gene/P and Cray XK7 have a 3D torus; Blue Gene/Q has a 5D torus; and the K-machine has a 6D torus. Effectively exploiting each topology is critical to I/O data movement.

File system locking is another aspect that degrades I/O performance. Applications typically write data to a single shared file or to a file for each process. As we move towards future systems, this mechanism could lead to significant locking overheads associated with parallel filesystems. We investigate the use of subfiling [2], wherein a few large files are written out instead for I/O.

Finally, we observe that for many scientific simulations, the numerical datasets that are generated have high entropy – which suggests the possibility of using compression to reduce the file size prior to its transfer. These large datasets also take considerable time to transfer to data storage, which increases overall application execution time. Therefore, reducing both the file size and the transfer time to storage are essential to our approach. The use of compression is further leveraged by the abundant computing power available in current systems.

In this paper we present an I/O framework that incorporates compression, topology-aware data movement and aggregation, and subfiling. In the next section, we briefly present an overview of Mira, Argonne National Laboratory's Blue Gene/Q supercomputer, and background of our research. Section III describes our compression-based, topology-aware, and subfiling-based I/O framework. In section IV, we show microbenchmarks for the framework and describe an evaluation of I/O performance improvement using an application at scale. We then discuss related work in section V. Finally we present our conclusions in section VI.

II. Background

In this section, we begin by describing the high performance system used to conduct this research. We then present our related work conducted on the previous generation IBM Blue Gene/P. Next we introduce two compression libraries: Blosc and zlib. While we use Blosc in our ongoing research efforts to improve I/O performance, we use zlib here to illustrate Bloscs distinguishing features.

A. Mira – a Blue Gene/Q Supercomputer

The IBM Blue Gene/Q system [3] is designed to provide high-performance, low power consumption supercomputing. Argonnes Mira system contains 48 racks, 768K cores, and has a theoretical peak performance of 10 petaflops. Each node has 16 cores in use, with 16 GB of RAM per node. I/O and interprocessor communication travels on a 5D torus network both for point-to-point and collective communications. This 5D torus interconnects each compute node with its 10 neighbors at 2 GB/s theoretical peak over each link in each direction, making a total of 40 GB/s bandwidth in both directions for one single compute node. Every 128 compute nodes has two 2 GB/s bandwidth links to two different I/O nodes, making 4 GB/s bandwidth for I/O at most. I/O nodes are connected to file servers through QDR IB. Mira uses a GPFS file system with 24 PB of capacity and 240 GB/s bandwidth.

B. GLEAN

In our effort to improve I/O performance on Blue Gene/P and Q systems, we developed GLEAN, a topology-aware data movement and staging framework for I/O acceleration. Instead of allowing every process to transfer its data to storage or external analysis clusters, GLEAN chooses a subset of processes, namely aggregators, based on network topology to do it. This approach reduces congestion due to the high number of data movements and number of write requests seen by the file system. It also increases the size of the write buffer, hence increasing write performance. GLEAN has been proven to increase the I/O performance of a Blue Gene/P system to close to the peak of the I/O system.

C. Data compression

Data compression is gaining importance for improving I/O performance. Compression reduces the size of the data, thereby decreasing the transfer time. It plays a key role in reducing the overall storage bandwidth and space requirements of the parallel storage design. There exist several general and specific purpose data compression libraries. We give an overview of the zlib [4] and Blosc [5] used in this work.

1) zlib: zlib is a general purpose lossless data compression library [4]. It uses a combination of the LZ77 algorithm and Huffman coding. Though widely used, it is not optimal for any specific data types. There are variations of zlib with trade-offs between speed and compression ratio.

2) Blosc: Blosc is a blocking, shuffling, and lossless compression library [5]. It reduces the size of data by assuming that the data within a certain space/volume does not change much, i.e., that the most significant bits do not change as much as the less significant bits. It shuffles the data to put the most significant bits together then uses BloscLZ, which is heavily based on FastLZ, to compress the data.

III. I/O FRAMEWORK

Numerous challenges are encountered when designing an I/O framework for high performance computing systems. These include: the interconnect networks to transfer data are complex, heterogeneous, and rapidly change over time; and existing filesystems may have different configurations to handle files differently depending on whether they are shared or are exclusive to a process. Our framework is divided into three components to handle these challenges. It is implemented in C/C++, uses MPI, and provides interfaces for both Fortran and C-based parallel applications.

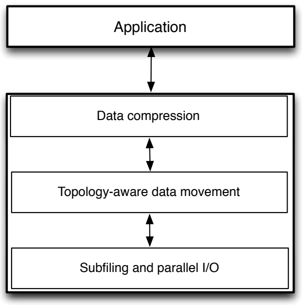


Fig. 1: Compression-based, topology-aware, subfiling-based I/O Framework

A. Framework components

1) Parallel data compression: As the application calls our framework to handle the write requests. It applies user-specified preconditioners and appropriate compression libraries on data. In our current implementation, we use only Blosc for data compression and decompression; however, programmers can use their own compression libraries with minimum modification to achieve the best speed/compression ratio. Numerous compression libraries provide the capability of parallel compression, allowing systems to take advantage of all available computing resources.

2) Topology-aware data movement: After compression, the data is transferred out of the compute nodes to I/O nodes and eventually is written to storage. The parallel data movement used by MPI-IO for writing data out uses an algorithm that is agnostic of the underlying network topology [1]. We present an explicit data movement mechanism that takes underlying network topology into consideration.

Our goal is to leverage the interconnection network topology to move data out of the compute nodes as soon as possible. We do this by developing a custom two-phase "collective I/O." In the first phase, data is aggregated to a subset of processes called aggregators. These aggregators then write the aggregated data out in the second phase. This technique reduces network contention and reduces the number of writing requests to the file servers. The underlying network and amount of data determines the appropriate number and location of aggregators as well as the data movement strategy. The detail of this work is presented in a simple two-part algorithm, called 1.

In the first part we determine the number of aggregators

Algorithm 1 Determine number and location of aggregators

```
1. Calculate number of aggregators.
Total amount of data of processes sharing one file: D.
Calculate number of aggregators (N) based on stripe size on filesystem(S): N = D/S.
2. Calculate location of aggregators.
Choose the split factors along dimensions sA, sB, sC, sD, sE that N = sA*sB*sC*sD*sE.
Partition the cluster along each dimension by split factors to have N blocks.
Color all processes in the same block with an unique color.
Create subcomms and select aggregators by using MPI_Comm_split.
```

based on two factors: the stripe size of the filesystem and the total amount of data needed to write. We calculate the total data size by gathering the data from all processes sharing a file to a preselected process. We then divide the total size by the stripe size of the filesystem to get the number of aggregators. The more data we have, the more aggregators we need. However, as we show later in microbenchmarks IV-B, that for a certain size of data, increasing the number of aggregators does not further improve overall performance. This calculation assures that the amount of data aggregated at each aggregator is more than the stripe size of the filesystem and hence mitigates expenses file locking in the underlying filesystem. In the second part of the algorithm, we determine the location of aggregators. This location is calculated based on an assumption that the data is distributed approximately equally among all processes. Under that assumption, aggregators are uniformly distributed among all processes. On the Blue Gene/Q, compute nodes are allocated for applications as a 5D hyper-cube cluster. To designate aggregators, the I/O framework partitions the cluster along each dimension into blocks. The I/O framework then selects a single aggregator to each block. The algorithm does so by assigning a unique color to processes in a block and using *MPI_Comm_split* to set up communications within the block (or subcomm) and to assign numbers to the processes; the algorithm assigns processes #0 as the aggregators. The subcomm is used to aggregate data. If aggregated data size is larger than available buffer, multiple rounds of compression and aggregation are needed.

3) Subfiling and parallel I/O: We observe that the number of output files for an application indeed affects I/O performance. This number can vary from a shared file to a few files to a file per process. As we experimented on our system, too few files (such as shared file/application) or too many files (file per process) both resulted in the I/O bottleneck of the system. In our system, I/O nodes manage the file metadata. Too many files per I/O node or too many I/O nodes sharing a file both lead to bottleneck in metadata management. Therefore, in this design, we choose to write one output file per I/O node to avoid inter-I/O node communication or file metadata management overhead. We saw better performance than shared file and file per process options. Aggregators belonging to the same I/O node first gather data from processes sharing that I/O node and then communicate to find out its file offset and write data in parallel to a shared file. The metadata of compressed data is gathered by a predefined process per file and written to the beginning of every file.

B. *Physical data layout*

Our framework also provides a file format for writing data to the storage system and for reading data back in a parallel fashion for restarts and simulation post-processing and analysis. Our file format includes four primary sections:

- Application metadata: the specific metadata for applications. Information such as number of variables, names, data types of variables, and length of data.
- Compression metadata: the metadata about compressed data helps to read data back from storage, decompress it, and reconstruct it to its original form. We need file offsets and lengths of compressed data to read it from storage. After decompressing it, we then need the application-specific metadata to reconstruct the original data.
- Compressed data: contains actual compressed data.
- Checksum: contains the checksum of the file to make sure that the data reading back is the data written before.

IV. Benchmarks

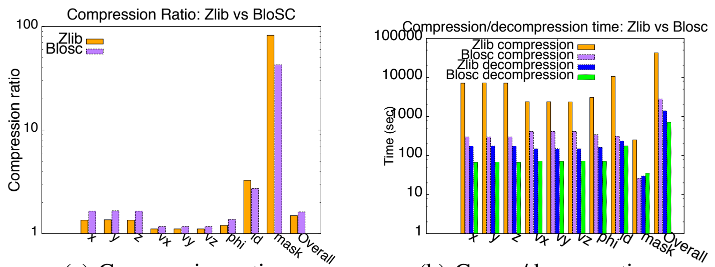
We chose a physics application called HACC (Hardware/Hybrid Accelerated Cosmology Code) to evaluate the efficacy of our I/O framework. HACC [6] is a large-scale cosmology code suite that simulates the evolution of the Universe through the first 13 billion years after the Big Bang. The simulation tracks the movement of trillions of particles as they collide and interact. Data for each particle includes nine variables (x, y, z, vx, vy, vz, phi, id, mask).

We evaluated our framework with microbenchmarks on two compression techniques, zlib and Blosc, and the performance of topology-aware aggregators on the achievable I/O throughput. We compared this with IOR [7] to understand the tradeoffs of performing I/O using a single shared file, a file per process, and subfiling, wherein we create a few large files. Finally, we show the efficacy of our framework in the HACC simulation.

A. *Efficacy of Compression*

In the following benchmarks, we define the compression ratio as follow: compression ratio $= \frac{\text{Size of data before compression}}{\text{Size of data after compression}}$

In HACC, the particles exhibit a Monte Carlo-like behavior, therefore the variables have a very high dynamic range and are difficult to compress. On both zlib and Blosc benchmarks, we used a single thread on a single core of a BG/Q node. Figure 2a compares the achievable compression ratios for the HACC data. The trade-off between ratio and speed can be determined either by the user or automatically by the system using a pre-defined utility function. For this paper, we fixed the compression level at level 9. Overall, Blosc achieved a better compression ratio than zlib. Figure 2b compares the time taken by zlib and Blosc for compression and decompression of the HACC datasets. The chief advantage of using Blosc is that it takes as input the data type size of the variable being compressed so it knows the possible repeatable patterns and can look for those patterns faster. Zlib scans for possible patterns byte by byte. In addition, Blosc loads the data to be compressed into cache L1 and confines it there while it performs its search for repeatable sequences, which speeds up the compression process. For both libraries, the decompression process took less time and was more stable than the compression process. This is because the compression process also involves the search for repeated sequences.



(a) Compression ratio (b) Compr/decompr time

Fig. 2: Compression libs Blosc vs. zlib for HACC datasets

B. *Efficacy of topology-aware aggregation*

We evaluated the efficacy of our algorithm to select the number of aggregators on one rack of Mira (1,024 nodes) with 8 I/O nodes, executed with 8 MPI processes per node and varying the data size from 1 KB up to 16 MB. We varied the number of aggregators per I/O node from 2 to 128. The number of aggregators eventually determines the location of aggregators in the 5D torus, as well as the data size of each write request, the number of processes directly working with file servers, and the memory needed to temporarily store aggregated data.

In Fig. 3, each line represents the bandwidth achieved using a particular number of aggregators per I/O node. Depending on data size, our framework chooses the number of aggregators (in the range of 4 to 32) to achieve the best performance.

The data shown in Fig. 3 validates of our aggregator number calculations and therefore our algorithm. Take for example a 32 KB data size point. According to our algorithm, with the data size at each rank being 32 KB, for one I/O node (128x8 ranks) we need 32Kx128x8/8MB=4 aggregators. As the figure shows, at data size 32KB the green line with num_agg=4 achieves the highest performance, exactly the same as our algorithm. We also compared this with the default aggregation mechanism used by MPI-IO. The aggregation mechanism [1] fails to account for the topology of the system interconnect and also the number of aggregators involved. We achieved a 10X improvement at 4K bytes per rank, a 3X improvement at 2M bytes message, and a 2X improvement at 16M. Improved performance for smaller messages is the primary role for MPI Collective I/O and we observed a significant improvement in this realm. Thus, our aggregation scheme is effective in improving the parallel I/O performance. As the interconnect network topology is getting more complex, it is important to exploit the topology to improve I/O performance.

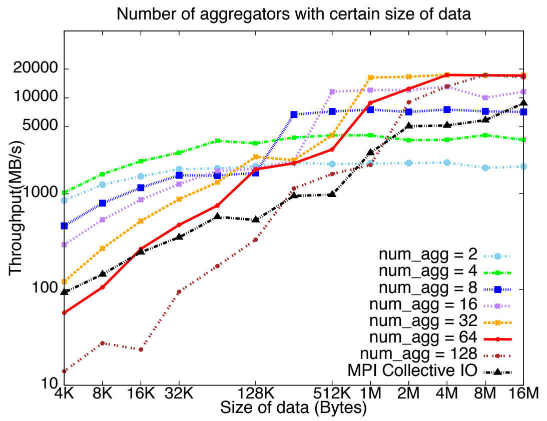


Fig. 3: Choosing number of aggregators per I/O node with various data size

C. Efficacy of subfiling

We evaluated the subfiling mechanism of our I/O framework by comparing it to IOR. IOR allows to write a file-per-process and a single shared file in writing data size 8 MB per rank. We scaled the I/O experiments from 2,048 cores to 256K cores. For the file-per-process case, we used POSIX I/O for IOR, and in the single shared file case, we used MPI-IO for IOR. As IOR does not support subfiling, we wrote a benchmark using our I/O framework, CGIO, to mimic IOR while incorporating subfiling. In the subfiling case, all the ranks associated with an I/O node of BG/Q write to a file. Thus, the total number of files written out is equal to the number of I/O nodes associated with the job (On Mira, this is 8 files per rack, or 16K cores).

Figure 4 shows that as the number of cores increased to 32,768, IOR performance for a single shared file also increased. However, as we scaled beyond to 256K cores, the performance failed to scale. This is primarily due to the poor scaling performance associated with GPFS filesystems locking performance for a single shared file. IOR with the file-per-process option generally has better performance, however, after 16,384 cores, its performance degrades due to the large number of files that the fileservers and file system have to handle, and the associated control information at this scale. In contrast, we see that with subfiling, our performance increases linearly as we scale the number of cores. At 256K cores, we are able to sustain 205GB/s – this is 85% of the peak I/O performance on the Mira BG/Q supercomputer, and a 40X improvement over a single shared file and a 10X improvement over the file-per-process performance. Thus, for scalable parallel I/O performance, subfiling is of paramount importance to mitigate the locking overheads associated with the underlying file system.

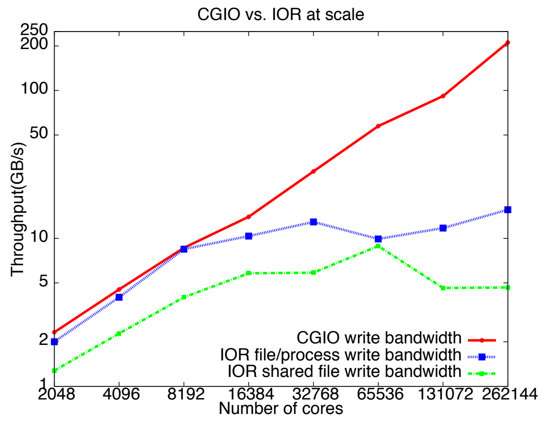


Fig. 4: Subfiling performance of our I/O framework in comparison to IOR write performance.

D. Weak Scaling Performance for HACC cosmology

We integrated and evaluated the performance of our I/O framework with the HACC code. We performed a weak scaling study and scaled the number of cores from 16,384 to 262,144 with a total number of particles ranging from $2,048^3$ to $5,012^3$. The total data per rank varies between 38 MB and 57 MB i.e., 1-1.5 million particles per rank. Thus, we wrote 400 GB at 16K cores and 4.8 TB of data at 256K cores. We compared the performance of performing I/O for HACC using four configurations: (1) MPI collective I/O to a single shared file; (2) subfiling wherein we have a file per ION; (3) using both topology-aware aggregation and subfiling; and (4) using all the three components of our framework i.e., compression, aggregation, and subfiling. We performed the simulation for 10 steps with each I/O configuration and reported the maximum performance achieved by each configuration.

Figure 5 depicts the performance of the various I/O configurations. Subfiling yielded a 5X improvement at 16K cores

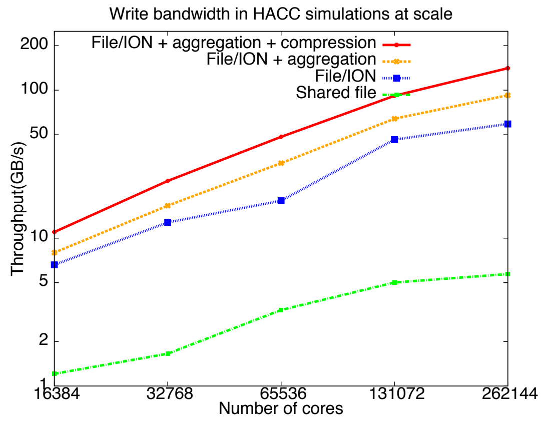


Fig. 5: Write bandwidth

and a10X improvement at 256K cores over a single shared file. Thus, subfiling is of critical importance as it mitigates the impact of the GPFS locking mechanism at scale. Topology-aware aggregation yields a 20% improvement over subfiling at 16K cores and up to an 80% improvement over subfiling at 256K cores. Thus, we observe that leveraging system topology plays an increasingly important role as we scale to larger core counts. Using compression, and thus using all three components of our I/O framework, we saw an additional 40% increase in performance over using aggregation. This is primarily due to our ability to achieve 50% compression for the HACC datasets, and thus writing less data to the storage system. Overall, by using subfiling, aggregation, and compression, we observed a 10X improvement over MPI collective I/O at 16K cores and a 14X improvement at 256K cores, achieving 130 GB/s. Thus, all three are critical to achieving scalable parallel I/O performance.

With the subfiling option (one file per I/O node) applied,

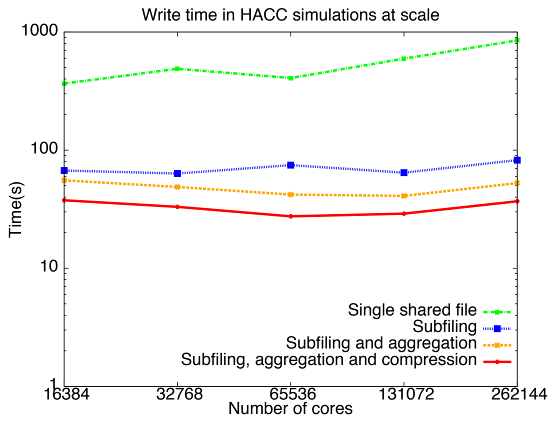


Fig. 6: Write time

the write performance increased 5 to 10 times. The aggregation adds up to 80% write performance and compression adds around 50% performance on top of that. The latter makes the write performance 10 to 20 times better. The time associated with each configuration is shown in Fig. 6. The combination of compression, topology-aware data aggregation, and the subfiling mechanism improved I/O performance on HACC application multifold. Thus, achieving optimal I/O system performance required improvements to each of the components, from compute nodes to storage disks.

V. Related work

I/O forwarding and I/O staging are common techniques to improve I/O performance. These two techniques have been studied previously; a scalable I/O forwarding framework for high-performance computing systems is described in [8]–[10], and an augmentation for I/O forwarding and asynchronous data staging for Blue Gene/P systems is presented in [11], [1] and [12], as well as for the Cray systems [13]

Data compression is employed to reduce storage space and to improve the bandwidth of data transfers at various locations in system. In [14], compression happens transparently at the network level, in [15] transparent compression applies to data stored on the SSD-based caches. In [16], the authors developed their own compression library and provided a framework for multi-threaded retrieval and multi-threaded compression when prefetching and caching data. ISOBAR [17] is a hybrid lossless compression algorithm for large-scale parallel I/O systems; developers evaluated several interleaving strategies to achieve high-speed I/O, including leveraging the overlap of I/O time for easy-to-compress and hard-to-compress data.

Overall I/O performance can also be improved via work in file systems. Bent et. al. proposed PLFS [18], a checkpoint file system for parallel applications that utilizes multiple checkpoint files, thereby avoiding the bottleneck caused by using a single shared checkpoint file. Our work can be distinguished from previous efforts as follows:

- We use compression and data aggregation together. Data compression reduces the size of the data files to write. However, write requests with sizes smaller than the stripe size can degrade system performance due to (1) requests share a stripe and wait for lock/unlock on the stripe and/or (2) overwhelm the file system with too many requests. Our I/O framework not only guarantees that the size of data at each process working with the file system is at least the stripe size, but also gathers a big bulk of data to write, thus improving I/O performance.
- Our topology-aware aggregation mechanism distributes the aggregators uniformly among the processes, therefore reducing network congestion and improving I/O performance.
- We choose a suitable subfiling mechanism that matches the number of processes to the capacity of the file system. In Blue Gene/Q systems, we write one file per I/O node, avoiding bottlenecks caused by either a single shared file or by a file-per-process mechanism.
- We compress all data instead of selecting a subset of the data to compress or to send out, based on our observation that compression time is relatively small compared to I/O time.
- We process data depending on its size and the amount of memory available. If the data size is too large to compress and aggregate, we cut the data into appropriately sized chunks and carry out multiple rounds of compression and aggregation.

VI. Conclusions

I/O is one of the critical performance bottlenecks for scientific applications on high-performance computing systems. The existing MPI-IO solution for the Blue Gene/Q systems fails to fully scale in terms of performance. The BG/Q system is a precursor to technologies we will need to scale in order to develop scalable solutions for future supercomputers. To mitigate this I/O bottleneck, we developed a framework to improve the I/O performance on BG/Q systems significantly by: utilizing compute power and appropriate compression/decompression libraries to reduce the size of data; using fast and dedicated interconnections to aggregate the data before it writing out; and, subfiling to mitigate the locking contention with parallel filesystems. We demonstrated the efficacy of our work through a set of microbenchmarks and application benchmarks with I/O performance improved by orders of magnitude. We plan to build analytical models to tune our I/O framework in order to deploy it onto other supercomputing systems.

Acknowledgment

This research used resources of the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory, and is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. We thank the NERSC support team and the ALCF team for discussions and help related to this paper.

References

[1] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka, “Topology-aware data movement and staging for i/o acceleration on blue gene/p supercomputing systems,” in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’11. New York, NY, USA: ACM, 2011, pp. 19:1–19:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063409

[2] K. Gao, W.-k. Liao, A. Nisar, A. Choudhary, R. Ross, and R. Latham, “Using subfiling to improve programming flexibility and performance of parallel shared-file i/o,” in *Proceedings of the 2009 International Conference on Parallel Processing*, ser. ICPP ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 470–477. [Online]. Available: http://dx.doi.org/10.1109/ICPP.2009.68

[3] D. Chen, N. Eisley, P. Heidelberger, S. Kumar, A. Mamidala, F. Petrini, R. Senger, Y. Sugawara, R. Walkup, B. Steinmacher-Burow, A. Choudhury, Y. Sabharwal, S. Singhal, and J. J. Parker, “Looking under the hood of the ibm blue gene/q network,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 69:1–69:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389090

[4] J. loup Gailly and M. Adler. zlib. [Online]. Available: http://zlib.net

[5] F. Alted, “Why modern cpus are starving and what can be done about it,” *Computing in Science and Engg.*, vol. 12, no. 2, pp. 68–71, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2010.51

[6] S. Habib, V. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. Insley, D. Daniel, P. Fasel, N. Frontiere, and Z. Lukić, “The universe at extreme scale: multi-petaflop sky simulation on the bg/q,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 4:1–4:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389002

[7] H. Shan, K. Antypas, and J. Shalf, “Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark,” in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, 2008, pp. 1–12.

[8] N. Ali, P. H. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. B. Ross, L. Ward, and P. Sadayappan, “Scalable i/o forwarding framework for high-performance computing systems,” in *CLUSTER*, 2009, pp. 1–10.

[9] K. Ohta, D. Kimpe, J. Cope, K. Iskra, R. Ross, and Y. Ishikawa, “Optimization techniques at the i/o forwarding layer,” in *Proceedings of the 2010 IEEE International Conference on Cluster Computing*, ser. CLUSTER ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 312–321. [Online]. Available: http://dx.doi.org/10.1109/CLUSTER.2010.36

[10] K. Iskra, J. W. Romein, K. Yoshii, and P. Beckman, “Zoid: I/o-forwarding infrastructure for petascale architectures,” in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, ser. PPoPP ’08. New York, NY, USA: ACM, 2008, pp. 153–162. [Online]. Available: http://doi.acm.org/10.1145/1345206.1345230

[11] V. Vishwanath, M. Hereld, K. Iskra, D. Kimpe, V. A. Morozov, M. E. Papka, R. B. Ross, and K. Yoshii, “Accelerating i/o forwarding in ibm blue gene/p systems,” in *SC*, 2010, pp. 1–10.

[12] J. Freche, W. Frings, and G. Sutmann, “High throughput parallel-i/o using sionlib for mesoscopic particle dynamics simulations on massively parallel computers,” in *Parallel Computing: From Multicores and GPU’s to Petascale*, ser. Advances in Parallel Computing, B. Chapman, F. Desprez, G. R. Joubert, A. Lichnewsky, F. J. Peters, and T. Priol, Eds., vol. 19. IOS Press, 2010, pp. 371 – 378.

[13] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, “Flexible io and integration for scientific codes through the adaptable io system (adios),” in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, ser. CLADE ’08. New York, NY, USA: ACM, 2008, pp. 15–24. [Online]. Available: http://doi.acm.org/10.1145/1383529.1383533

[14] B. Welton, D. Kimpe, J. Cope, C. M. Patrick, K. Iskra, and R. Ross, “Improving i/o forwarding throughput with data compression,” *2012 IEEE International Conference on Cluster Computing*, vol. 0, pp. 438–445, 2011.

[15] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas, “Using transparent compression to improve ssd-based i/o caches,” in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys ’10. New York, NY, USA: ACM, 2010, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755915

[16] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt, “Integrating online compression to accelerate large-scale data analytics applications,” *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 1205–1216, 2013.

[17] E. R. Schendel, S. V. Pendse, J. Jenkins, D. A. Boyuka, II, Z. Gong, S. Lakshminarasimhan, Q. Liu, H. Kolla, J. Chen, S. Klasky, R. Ross, and N. F. Samatova, “Isobar hybrid compression-i/o interleaving for large-scale parallel i/o optimization,” New York, NY, USA, pp. 61–72, 2012. [Online]. Available: http://doi.acm.org/10.1145/2287076.2287086

[18] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, “Plfs: a checkpoint filesystem for parallel applications,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC ’09. New York, NY, USA: ACM, 2009, pp. 21:1–21:12. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654081