What Do Scientific Applications Need? An Empirical Study of Multirail Network Bandwidth

Edgar A. León, Chris Chambreau, and Matthew L. Leininger

Lawrence Livermore National Laboratory
Livermore, California, USA
Email: {leon,chambreau1,leininger4}@llnl.gov

Abstract—**High performance computing applications are commonly executed on large parallel machines composed of commodity components. These commodity clusters utilize high-speed interconnects that provide low latency and high bandwidth such as InfiniBand. Understanding the characteristics of scientific applications is important to properly configure and tune these machines and their software stacks. Are applications limited by network performance? Can they leverage increased network bandwidth? What type of network operations and message sizes do they use? This work provides a better understanding of the communication requirements of scientific applications by investigating the impact of multirail networking on their performance. We measure the performance of a suite of high performance computing mini-applications under different multirail configurations to determine their sensitivity to network bandwidth. The selected mini-applications provide simplified source code containing data access patterns and computational characteristics of larger production codes. The type of analysis presented in this paper can be applied to inform the procurement of future systems maximizing application productivity within a given capital budget.**

Keywords–HPC; performance measurement; multirail networking; network characterization; scientific applications.

I. Introduction

Most scientific applications use the Message Passing Interface (MPI) to leverage parallelism across multiple nodes. As such, network performance is important for application and system developers and those responsible for hardware procurements. When procuring a supercomputing cluster, for example, there is a range of options in terms of processor, memory, and network capabilities. A question one may ask is whether doubling the network bandwidth is worth sacrificing upgrades in processor speed. The right balance depends on the suite of applications that will execute on such system. The more we understand the characteristics and requirements of our applications, the better decisions we can make to provide the best performance within a given capital or power budget.

In this work, we provide an empirical study of the network requirements of a suite of high performance computing (HPC) applications and the impact of network speed on their performance. The codes we employ are part of a suite of mini-applications that contain data access patterns and computational characteristics of larger production codes within the U.S. Department of Energy (DOE). These include sparse linear algebra, shock hydrodynamics, Monte Carlo particle transport, finite element, and radiation transport codes.

We start by characterizing the communication requirements of these applications in terms of their point-to-point (P2P) and collective operations and message sizes. Point-to-point operations, such as `Send` and `Receive`, require some form of synchronization between two processes and scale primarily in terms of message size. Collective operations, such as `Barrier` and `Allreduce`, require the synchronization of all tasks in a particular group (often all MPI processes) and scale with respect to the number of participating nodes and message size.

We leverage multirail networking (multiple network interfaces per node) as a vehicle to examine the impact of network speed on application performance. When using multirail we evaluate several policies to understand locality and affinity tradeoffs. Our sensitivity study includes two parts. In the first part, we evaluate the performance improvements of multirail on micro-benchmarks to determine the potential gains of this technology. Although micro-benchmark evaluations have been done in the past [1]–[4], we need information to assess how much of these gains is actually realized by applications. This is the focus of the second part, where we evaluate the sensitivity of our codes to network bandwidth. Our goal is to better understand how dependent these applications are to network performance, information that can be used by application and system developers and for system procurements.

The reminder of the paper is organized as follows. Section II describes the machine environment used for the empirical study. Sections III and IV describe the multirail policies and applications employed, respectively, while Section V characterizes the communication characteristics of the selected applications. We measure the impact of multirail on micro-benchmarks and applications in Sections VI and VII, respectively. The limitations of this study are presented in Section VIII and the related work presented in Section IX. Finally, Section X presents our conclusion and future work.

II. Machine Parameters

We employed the *Catalyst* machine at Lawrence Livermore National Laboratory (LLNL), a 324 node Linux cluster with two Intel IvyBridge (Xeon E5-2695 v2) processors and 128 GB of memory per node. Each processor has 12 cores at 2.4 GHz with 2 hardware threads per core (Intel Hyper-Threading). The nodes are connected via an InfiniBand dual-rail, Quad Data Rate network (Intel QDR-80) in a fully-provisioned Fat-Tree topology. It runs the Tri-Lab Operating System Software (TOSS) [5]. At the time the experiments were executed, Catalyst was running TOSS version 2.2, which is based on Red Hat Enterprise Linux Server release 6.5. Each processor has a theoretical peak memory bandwidth of 59.7 GB/s and uses DDR3 memory at 1866 MHz. The MPI library we used is MVAPICH2 version 1.9 with the GNU compiler.

III. Multirail Networking Policies

Multirail networks consist of multiple network interface controllers (NICs) per compute node. Catalyst has two InfiniBand cards per node (4x10 Gb/s links per card) connected to a single plane. Each card is placed in proximity to a processor socket. The MPI library is implemented on top of a low-level network layer: Intel Performance Scaled Messaging (PSM). PSM provides options for binding network traffic from a given MPI task or process to a given card [6].

To understand the impact of the increased bandwidth provided by multirail and the affinity to a local NIC, we instrumented the policies shown in Table I.

TABLE I. Multirail network policies.

Default	PSM default dual-rail policy. It allocates MPI processes to the NICs in an alternating or round robin fashion
NIC-0	Route all traffic through card 0
NIC-1	Route all traffic through card 1
Local-NIC	Route traffic from each socket through its local NIC

We make the following observations about our policies and network configurations. First, we consider both policies NIC-0 and NIC-1 because even though the hardware configuration is symmetric, NIC 0 is used for system-level traffic. Depending on the application, this network *noise* may have an impact on performance. Second, the PSM driver version installed on Catalyst does not instrument striping of messages across NICs or using a local NIC at the PSM level (thus the need for our Local-NIC policy). And, third, even though MVAPICH provides dual-rail policies, they were not implemented for Intel fabrics at the time the experiments were executed.

Our goal is not to provide a comprehensive study of multirail policies but to understand the impact of the network on application performance.

IV. Applications Suite

Our codes consist of five mini-applications from the CORAL benchmark suite [7], which represent DOE workloads and technical requirements [8]. CORAL is the result of a joint Collaboration between Oak Ridge, Argonne, and Lawrence Livermore National Laboratories, and provides simplified source code to emulate the data access patterns and computational characteristics of larger production codes. These codes were used in the procurement of LLNL's next advanced technology system, *Sierra*, which will provide over 100 petaflop/s and is expected to be operational in 2018.

The CORAL benchmarks are grouped into several categories including scalable science, throughput science, and data-centric benchmarks. In this paper, we use the throughput codes shown in Table II and will study the other two categories in future work. The throughput benchmarks represent particular subsets of codes used as part of the everyday workload of science applications frequently executed on commodity clusters.

AMG2013 is a benchmark application derived directly from the BoomerAMG solver in the Hypre linear solvers library. AMG2013 is an algebraic multigrid solver for linear systems built from problems on unstructured grids. The default problem is a Laplace-type problem with various jumps and anisotropy in one part.

TABLE II. Our CORAL throughput benchmark suite.

AMG2013	Algebraic multi-grid linear system solver
LULESH	Shock hydrodynamics for unstructured meshes
MCB	Monte Carlo transport
miniFE	Finite element code
UMT2013	Unstructured mesh deterministic radiation transport

LULESH is an explicit hydrodynamics application performed on a staggered grid mesh. It solves the Sedov problem on one octant of a sphere using Lagrangian hydrodynamics. Originally developed as one of the five DARPA UHPC challenge problems, it is now used in DOE co-design activities and machine procurements.

MCB is a Monte Carlo particle transport proxy-application for multi-physics simulation codes. It is written in C and uses MPI+OpenMP for parallelism. MCB performs a significant number of integer operations as well as branches.

miniFE is designed to be the "best approximation to an unstructured implicit finite element or finite volume application, but in 8000 lines or fewer." The benchmark attempts to provide as much coverage of the implicit finite element application space as possible given its size constraints.

UMT is an unstructured-mesh deterministic radiation transport benchmark. It is a single physics package code that performs three-dimensional, non-linear, radiation transport calculations using deterministic methods. UMT exercises memory bandwidth and is compute intense.

V. MPI Characteristics of Applications

We collect MPI statistics using mpiP, an MPI profiling library [9]. Figure 1 shows the percentage of execution time spent on communication (left) and the average message size exchanged by our applications (right). These metrics are itemized in terms of P2P and collective operations using two configurations: MPI-only—1 task per core—and MPI+OpenMP—1 task per socket with 12 threads per task. Given that each node has two sockets, each with 12 cores, the MPI-only configuration has 24 processes per node (PPN) while the MPI+OpenMP configuration has 2 PPN. The MPI performance characteristics were similar for these two configurations.

We observe that most applications spend more than half of their MPI execution time performing collective operations. The exceptions are AMG and MCB. AMG spends a large fraction of time on P2P calls. In addition, it spends by far the highest percentage of execution time on communication. Despite taking a large percentage of execution time, AMG sends mostly small P2P messages. Similarly, MCB uses small messages for P2P operations but larger messages for collectives.

With the exception of MCB, collective operations send significantly less data across the network than P2P operations in the same application. LULESH spends nearly all of its MPI time making calls to Allreduce with a message size of 8 bytes, while a number of `Isend` calls incur orders of magnitude more bandwidth while requiring roughly one tenth of the time compared to the Allreduce calls.

Based on the communication characteristics of applications, we expect their sensitivity to network bandwidth to be application-dependent. In AMG or UMT, we would expect increased bandwidth to be helpful. However, in LULESH or

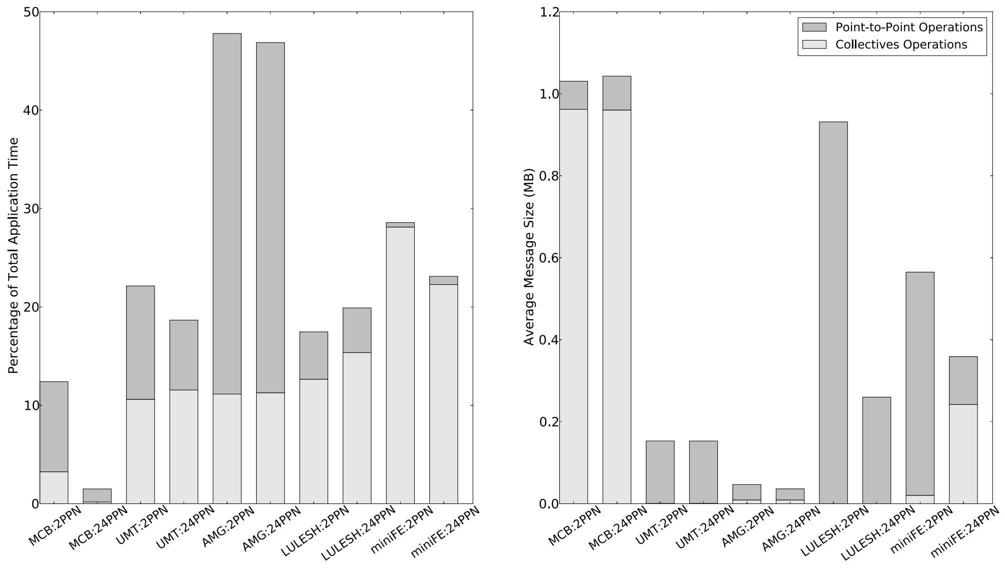

Figure 1. Collectives and P2P communication breakdown as a percentage of total execution time and their average message size. Experiments executed under two configurations, MPI-only (24 PPN) and MPI+OpenMP (2 PPN), on 256 nodes.

miniFE factors affecting latency, such as system noise and message injection rate, may impact their performance more than efforts to increase bandwidth.

VI. Characterizing multirail bandwidth

To assess the potential benefits of multirail we employ the Phloem MPI Benchmarks [10]. They provide a collection of vendor-independent benchmarks that measure various aspects of MPI performance including aggregate bandwidth, interconnect messaging rate, collective latencies, and point-to-point latency. We focus on network bandwidth.

A significant advantage of dual-rail over single-rail is the potential for doubling the off-node communication bandwidth. Using the Phloem *Presta* aggregate bandwidth benchmark [10], we measured off-node aggregate bandwidth by exchanging messages between 24 tasks on one node and 24 tasks on a different node. Message sizes range between 32 bytes and 8 MB. As Figure 2 shows, for small message sizes, single and dual-rail performance is similar, with only a slight advantage for dual-rail. However, for message sizes greater than 256 bytes, the dual-rail aggregate bandwidth provides better performance reaching twice the bandwidth at around 1 KB. For message sizes of 32 KB and greater, the aggregate bandwidth for both single-rail and dual-rail plateaus with the dual-rail aggregate bandwidth essentially twice that of single rail.

Thus, using micro-benchmarks, dual-rail can double the off-node bandwidth when messages are sufficiently large. The next aspect to evaluate is whether the selected applications can benefit from this doubling of network bandwidth.

VII. Impact on Applications

Each of the CORAL benchmarks reports a metric called *Figure of Merit (FOM)* indicative of application performance. The FOM is defined independently for each application and designed to scale linearly with performance [8]. We measured the FOM for each application with our network policies between 80 and 100 times because of nontrivial levels of

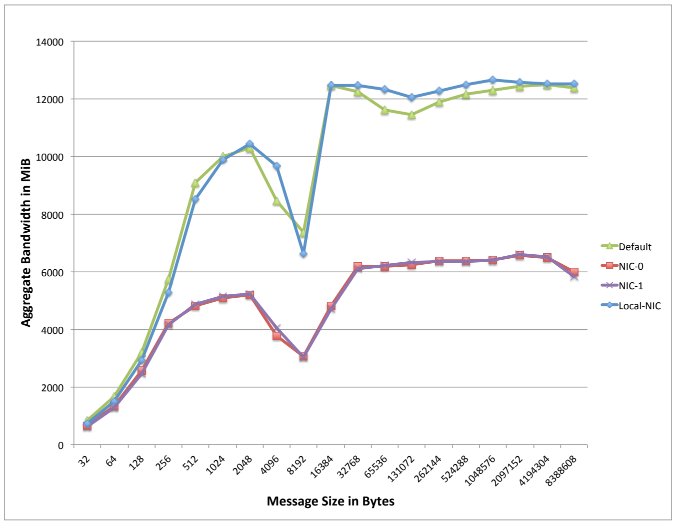

Figure 2. Network bandwidth between two nodes, each node uses 24 tasks.

runtime variations between runs. Using box-and-whisker plots, we show the first and third quartiles, median, minimum, maximum, and outliers. Figure 3 shows the MPI-only results for AMG, miniFE, and LULESH, where higher (FOM) is better. The MCB results are not shown because they are similar to miniFE and we discuss UMT separately (Figure 4) since this application demonstrated the most gains with dual-rail.

We observe negligible difference in application performance with the different network policies, except for slight improvements with the default dual-rail on AMG and UMT. Second, there is a significantly higher variability in execution time with NIC-0 compared to NIC-1 as shown by miniFE and MCB. As mentioned previously, system messages use NIC-0 for communication affecting the larger messages sent with collective operations of these two applications.

As shown in Figure 3c, despite significant time spent on P2P communications, AMG only benefits from the increased

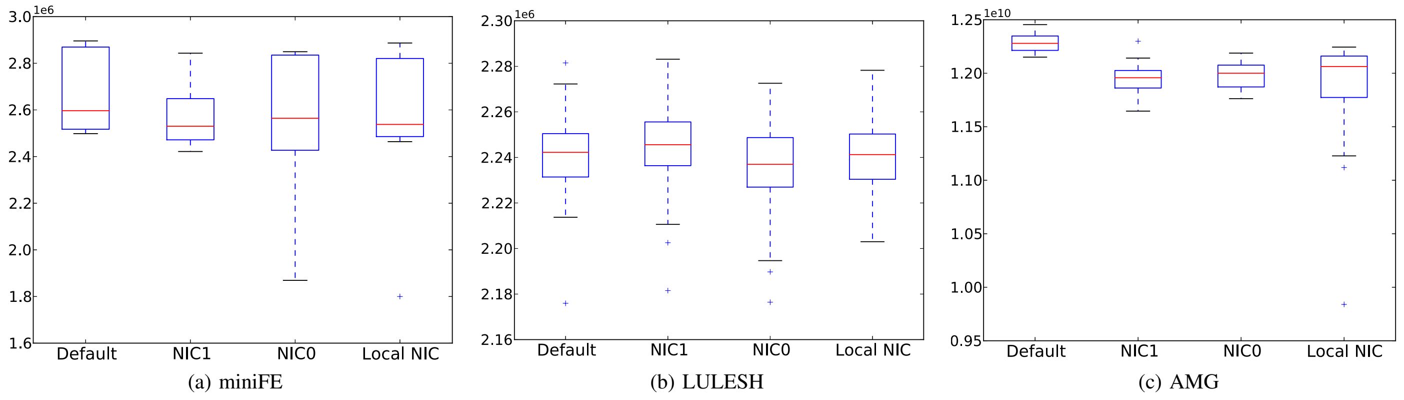

Figure 3. Application performance, measured as Figure of Merit, as a function of different network configurations. Higher is better.

bandwidth provided by multirail on the order of 2%. This is due to the relatively small average P2P message size, which indicates that MPI performance for AMG is driven more by latency and synchronization overhead than bandwidth.

UMT exhibited the most gains from multirail but even here performance only increased by approximately 3.2%. To understand this result, we measure the percentage of execution time spent on MPI (vertical axis) as a function of P2P and collective operations over many runs (horizontal axis) and plot it in Figure 4. Most of the UMT gains with dual-rail are due to reductions in time spent on P2P operations (contrast solid lines at the top and bottom), which are as much as 18% faster overall, with collectives showing modest improvements but significant variations in performance.

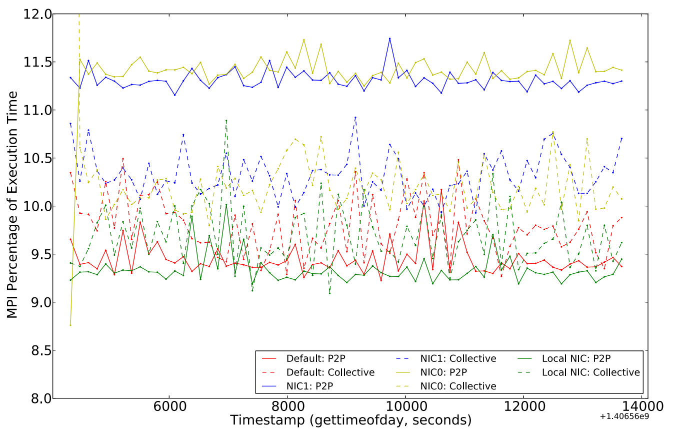

Figure 4. UMT percentage of execution time spent on MPI itemized by P2P (solid lines) and collective operations (dashed lines) over successive runs. Colors represent different network configurations.

While both AMG and UMT spend significant time on P2P operations, UMT's messages are larger on average, making it more sensitive to bandwidth increases. In contrast, miniFE and LULESH heavily rely on collective communications, where performance is dominated by issues of synchronization, communicator size, and latency. As Figure 4 also shows, collectives show significant variations in performance because of their sensitivity to system noise.

VIII. Discussion and Limitations

In this work, we demonstrate that a suite of scientific HPC codes are, mostly, not sensitive to network bandwidth. The current bandwidth provided by a single rail is sufficient for our codes resulting in negligible improvements with an additional rail. In addition, these applications spend a significant percentage of communication time in collectives of small sizes, which are more sensitive to system noise rather than network speed. Furthermore, as scale increases so does the impact of noise and, thus, the amount of time spent on collectives.

The throughput benchmarks studied here represent important subsets of applications that are executed as part of the everyday workload of science applications on commodity clusters. This, however, does not mean that *all* scientific applications behave this way. In particular, other scalable science applications designed to run at the highest scale on the largest non-commodity machines may present a different sensitivity. Similarly, data-centric applications such as data analytics may impose more demanding network requirements. Furthermore, the differences between the mini-applications and the full-fledged applications from which they are derived may have an impact on their communication.

The scale of this study involves up to 6,144 MPI tasks over 256 nodes. Even though this scale is representative of workloads in commodity systems, it is likely that as scale increases to a few thousand nodes the ratio of compute to communication may change significantly resulting in different communication bottlenecks than what we observed here.

Finally, we want to emphasize that using realistic benchmarks such as the CORAL mini-applications and, ideally, full-fledged applications, provides a more accurate representation of the real impact of a proposed technique or approach. While micro-benchmarks are useful in the design of hardware and software components, ultimately, these are not the codes that are run on production systems. As we demonstrated, multirail can double network bandwidth on micro-benchmarks, but this improvement does not necessarily translate in a significant improvement in application performance. As shown in Section V, a medium-sized run with 6 K processes shows that codes are not necessarily bound by network bandwidth.

IX. Related Work

Related work includes the evaluation of different approaches to leverage multiple rails including striping messages across two NICs [1]–[4]. These mechanisms have shown significant improvements in network bandwidth and latency. Communication libraries such as MPI have been instrumented accordingly to take advantage of multirail networks. Simulation approaches have also been useful in complementing empirical studies with added flexibility in terms of network topology, routing algorithms, etc. [11], [12]. Furthermore, multiple rails have been shown to address network congestion and network failures [13], [14]. For many of these studies, micro-benchmarks have been used to demonstrate the improvements in latency and bandwidth with multi-rail.

The investigations closer to ours include empirical studies that analyze the impact of multirail on the performance of applications. Choi et al. evaluates the AMBER and LAMMPS applications on a Torus network using 256 and 512 cores [15]. Schreiber et al. shows performance improvements of the LS-DYNA application on a Hypercube network on 256 cores [16]. Liu evaluates the performance of LAMMPS under different input tests on a Hypercube network on 1,536 cores [17]. Depending on the application and input test, performance improvements with multirail vary.

In this paper, we focus our multirail evaluation on several codes of interest to the DOE. The scale of our study involves 6,144 cores on a fully-provisioned Fat-Tree network. While there are multiple factors in multirail performance including topology, scale, and congestion, ultimately, its performance impact depends on the communication characteristics of the workloads of interest.

X. Conclusion and Future Work

This work provides a better understanding of the network characteristics of a suite of scientific mini-applications representing data access patterns and computational characteristics of larger applications within the U.S. Department of Energy. Increased network bandwidth via multirail may be of limited benefit for many scientific codes running on commodity clusters. Despite substantial improvements on micro-benchmarks, the communication patterns of scientific workloads seem to benefit only slightly. These applications are not bandwidth bound sending mostly small messages, and many of the larger messages are performed asynchronously. These codes are more sensitive to network latency and load imbalance.

There are multiple avenues for future work. First, characterizing the selected codes at larger scales, particularly to identify any potential changes in compute to communication ratios. Second, analyzing other types of applications including scalable science and emerging data-centric codes. Third, comparing the effects of varying network bandwidth on application performance via other mechanisms including down clocking network links, say, from 50 Gb/s (HDR link speed) to 25 Gb/s (EDR link speed). Then, we can compare the tradeoffs of having one fast NIC per node vs. two slower NICs per node, e.g., HDR single rail vs. EDR dual rail.

Acknowledgment

We would like to thank the anonymous reviewers for their useful feedback. Our appreciation to our colleagues Trent D'Hooge, Adam Moody, and Jim Foraker for their help and support with the Catalyst system.

Prepared by LLNL under Contract DE-AC52-07NA27344. LLNL-CONF-670475.

References

[1] J. Cai, A. P. Rendell, and P. E. Strazdins, "Non-threaded and threaded approaches to multirail communication with uDAPL," in International Conference on Network and Parallel Computing, ser. NPC'09. Gold Coast, Australia: IEEE, Oct. 2009, pp. 233–239.

[2] V. Vishwanath, T. Shimizu, M. Takizawa, K. Obana, and J. Leigh, "Towards Terabit/s systems: Performance evaluation of multi-rail systems," in High Speed Networks Workshop. Anchorage, AK: IEEE, May 2007.

[3] A. Vishnu, G. Santhanaraman, W. Huang, H.-W. Jin, and D. K. Panda, "Supporting MPI-2 one sided communication on multi-rail InfiniBand clusters: Design challenges and performance benefits," in International Conference on High Performance Computing, ser. HiPC'05, Dec. 2005, pp. 137–147.

[4] J. Liu, A. Vishnu, and D. K. Panda, "Building multirail InfiniBand clusters: MPI-level design and performance evaluation," in Conference on Supercomputing, ser. SC'04. Pittsburgh, PA: IEEE, Nov. 2004, p. 33.

[5] "TOSS: Speeding up commodity cluster computing," Apr. 2016. [Online]. Available: https://computation.llnl.gov/projects/toss-speeding-commodity-cluster-computing

[6] True Scale Fabric OFED+ Host Software. User Guide., Intel Corporation, Sep. 2013.

[7] "CORAL benchmark codes," Dec. 2013. [Online]. Available: https://asc.llnl.gov/CORAL-benchmarks/

[8] CORAL: Collaboration of Oak Ridge, Argonne and Livermore National Laboratories, "Draft CORAL build statement of work," Office of Science and the National Nuclear Security Administrations Advanced Simulation and Computing (ASC) Program, U.S. Department of Energy, RFP No. B604142, LLNL-PROP-636244, Dec. 2013.

[9] "mpiP: Lightweight, scalable MPI profiling," Mar. 2014. [Online]. Available: http://mpip.sourceforge.net/

[10] "ASC Sequoia benchmark codes," Jun. 2013. [Online]. Available: https://asc.llnl.gov/sequoia/benchmarks/

[11] N. Wolfe, M. Mubarak, N. Jain, J. Domke, A. Bhatele, C. D. Carothers, and R. B. Ross, "Preliminary performance analysis of multi-rail fat-tree networks," in International Symposium on Cluster, Cloud and Grid Computing, ser. CCGrid'17. Madrid, Spain: IEEE/ACM, May 2017.

[12] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits, "Using multirail networks in high-performance clusters," in International Conference on Cluster Computing. Newport Beach, CA: IEEE, Oct. 2001, pp. 15–24.

[13] S. P. Raikar, H. Subramoni, K. Kandalla, J. Vienne, and D. K. Panda, "Designing network failover and recovery in MPI for multi-rail InfiniBand clusters," in International Parallel and Distributed Processing Symposium Workshops & PhD Forum, ser. IPDPSW'12. IEEE, 2012, pp. 1160–1167.

[14] A. Nukada, K. Sato, and S. Matsuoka, "Scalable multi-GPU 3-D FFT for TSUBAME 2.0 supercomputer," in International Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC'12. Salt Lake City, Utah: IEEE, 2012, pp. 44:1–44:10.

[15] D. J. Choi, G. K. Lockwood, R. S. Sinkovits, and M. Tatineni, "Performance of applications using dual-rail InfiniBand 3D Torus network on the Gordon supercomputer," in Conference on Extreme Science and Engineering Discovery Environment, ser. XSEDE'14. Atlanta, GA: ACM, 2014, pp. 43:1–43:6.

[16] O. Schreiber, M. Raymond, and S. Kodiyala, "LS-DYNA performance improvements with multi-rail MPI on SGI Altix ICE clusters," in International LS-DYNA Users Conference, Dearborn, MI, Jun. 2008, pp. 5:21–5:26.

[17] J. Liu, "LAMMPS on advanced SGI architectures," SGI, White Paper, 2011.