Enabling Parallel Simulation of Large-Scale HPC Network Systems

Misbah Mubarak, Christopher D. Carothers, Robert B. Ross, and Philip Carns

Abstract—With the increasing complexity of today's high-performance computing (HPC) architectures, simulation has become an indispensable tool for exploring the design space of HPC systems—in particular, networks. In order to make effective design decisions, simulations of these systems must possess the following properties: (1) have high accuracy and fidelity, (2) produce results in a timely manner, and (3) be able to analyze a broad range of network workloads. Most state-of-the-art HPC network simulation frameworks, however, are constrained in one or more of these areas. In this work, we present a simulation framework for modeling two important classes of networks used in today's IBM and Cray supercomputers: torus and dragonfly networks. We use the Co-Design of Multi-layer Exascale Storage Architecture (CODES) simulation framework to simulate these network topologies at a flit-level detail using the Rensselaer Optimistic Simulation System (ROSS) for parallel discrete-event simulation. Our simulation framework meets all the requirements of a practical network simulation and can assist network designers in design space exploration. First, it uses validated and detailed flit-level network models to provide an accurate and high-fidelity network simulation. Second, instead of relying on serial time-stepped or traditional conservative discrete-event simulations that limit simulation scalability and efficiency, we use the optimistic event-scheduling capability of ROSS to achieve efficient and scalable HPC network simulations on today's high-performance cluster systems. Third, our models give network designers a choice in simulating a broad range of network workloads, including HPC application workloads using detailed network traces, an ability that is rarely offered in parallel with high-fidelity network simulations.

Index Terms—Massively parallel discrete-event simulation, interconnect networks, trace-based simulation.

1 Introduction

Interconnect performance largely determines the overall effectiveness of modern high-performance computing (HPC) systems. Network topology—which includes the layout of network nodes, routers, and channels—plays a key role in determining network performance. To analyze the performance of network topologies and their different configurations prior to building them, network designers turn to analytical modeling and simulation. Although analytical modeling provides a flexible and fast estimate for making design decisions about networks, this advantage comes at the cost of accuracy because these models tend to make unrealistic simplifications about network design [1]. Serial time-stepped simulators can provide a high-fidelity and accurate picture of network performance by simulating the network cycles [2], but their scalability is limited by the simulation performance. In order to make informed design choices about large-scale networks, a network simulator must be fast, accurate, and scalable and have high fidelity at the same time. In this context, parallel discrete-event simulation (PDES) provides an opportunity to model large-scale networks with a sufficient fidelity [3]. However, many packet-level discrete-event simulations are limited in performance because of excessive global synchronization. When coupled with optimistic event scheduling [4], performance and scalability requirements can also be met.

When modeling HPC interconnection networks, using workloads of a stochastic nature is the most common approach for performance evaluation. These synthetic workloads have a specific interarrival time, message size, and destination distribution [3]. Although such traffic patterns are helpful in assessing a given network configuration under a variety of controlled scenarios, however, they are not representative of the communication phases in large-scale scientific applications [3]. Postmortem traces that reflect the communication behavior of scientific applications can be used to drive the simulation and achieve realistic estimates of network performance. However, handling these traces in simulation is a nontrivial task. The first challenge is that running large network traces (e.g., in the range of several gigabytes representing 100,000 MPI processes) on top of a detailed network simulation can become a bottleneck. For this reason, many simulation frameworks use simplified analytical models or ignore network contention details altogether when using real communication traces [5], [6]. The second challenge is that the simulation must also preserve the causal dependencies between the operations in the traces. For example, if an application performs frequent synchronization through the MPI wait operation, then the simulation must also wait for the MPI wait operation to complete before the next operation can progress.

In this paper, we address these constraints of simulation accuracy, detail, efficiency, and realistic workload representation that limit the use of simulation in the codesign of large-scale networks. We present two high-fidelity models of large-scale networks developed as part of the Co-Design of Multi-layer Exascale Storage Architecture (CODES) simula-

- *Misbah Mubarak, Robert B. Ross, and Philip Carns are with the Mathematics and Computer Science (MCS) Division at Argonne National Laboratory, 9700 South Cass Ave., Lemont, IL 60439. Email: {mubarak,rross,carns}@mcs.anl.gov*
- *Christopher D. Carothers is with the Computer Science Department, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180. E-mail: chrisc@cs.rpi.edu*

Manuscript received June 1, 2015.
Preliminary version of this work has previously appeared in PMBS 2012 workshop (Supercomputing) and ACM SIGSIM PADS 2014.

tion toolkit. CODES enables the exploration of HPC network and storage system design by providing a higher-level modeling API and high-fidelity and scalable models for HPC network and storage systems [7]. CODES uses the Rensselaer Optimistic Simulation System (ROSS) discrete-event simulator as its underlying discrete-event simulation framework, which has been shown to process billions of events per second on leadership-class supercomputers [8]. Using the optimistic event scheduling capability of ROSS, we model and explore the design space of two popular classes of networks in CODES at a flit-level fidelity at the size of next-generation HPC systems [9], [10]. The first design is the low-diameter, low-latency dragonfly network, which uses a high-radix router to provide high network bandwidth [11], [12]. The second class of networks is the torus [2], which uses multiple network links to exploit locality between the network nodes. Our network models provide the ability to explore various network configurations using both synthetic and trace-based workloads. To accurately simulate trace-based communication, we preserve the causal dependencies among the MPI operations through a simulated MPI layer, as discussed in Section 4.2.

The contributions of this paper are as follows.

1) We present a methodology for modeling MPI point-to-point messaging on extreme-scale torus and dragonfly networks at a detailed fidelity on modern HPC systems using optimistic event scheduling, which enables faster simulations by having a higher simulation event rate and reducing extensive global synchronization [13].
2) Our work enables network designers to gain insight into network design decisions through both synthetic workloads and traces from real applications that are considered representative of future large-scale compute systems [14].
3) We present examples of architecture design space exploration in which we investigate the network performance at a large scale to provide useful information about how a given torus or dragonfly network behaves with representative application traces.
4) We demonstrate that our large-scale trace-based network simulations with up to 110,000 simulated network nodes can execute on a modest number of cores (up to 64 cores) on a high-performance cluster in a reasonable amount of time.
5) To demonstrate the performance benefits of using discrete-event simulation for network modeling, we compare the simulation runtime of the CODES discrete-event dragonfly network model with that of the time-stepped series simulator *booksim* and observe a 5x-12x speedup over the *booksim* simulation on a single node with sequential execution.

2 Background

We first briefly describe the ROSS simulator. We then discuss the two network designs of interest here: the dragonfly and the torus.

2.1 ROSS event-driven simulation

Parallel discrete-event simulations are made up of logical processes (LPs), where each LP models a component of the system and has a distinct state. These LPs interact with one another via events in the form of timestamped messages. Since each LP maintains its own local simulation time and since not all LPs progress at the same rate, events can arrive at an LP that have a timestamp earlier than its local simulation time. Many parallel discrete-event simulators handle this issue by taking a *conservative* approach of delaying to process an event with timestamp t until it is is guaranteed that no other event having timestamp less than t will arrive. This conservative approach involves excessive global synchronization, however, which limits its scalability.

Optimistic event scheduling prevents this extensive global synchronization by allowing events to be processed until an out-of-order event is detected. The events are then rolled back and re-executed in the correct order [4]. ROSS supports both conservative and optimistic event scheduling. The latter is enabled through a technique called *reverse computation*, in which model designers write rollback functions. Optimistic scheduling in ROSS has been shown to dramatically improve the runtime of parallel simulations and reduce the amount of state-saving overhead [15]. The reverse computation feature enables us to run our large-scale network simulations at a much higher event rate than with the conservative event scheduling approach.

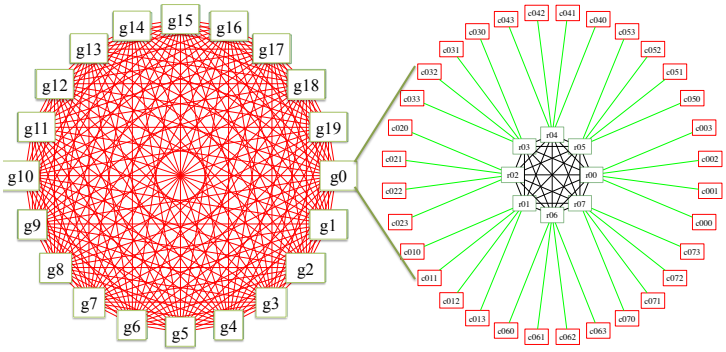

Fig. 1. Dragonfly intergroup and intragroup settings with a=8, p=4, h=4 (some groups are not shown).

2.2 Dragonfly networks

The dragonfly network (an example is shown in Figure 1) is a hierarchical topology composed of several virtual groups connected by all-to-all links. It has been deployed successfully in the Cray XC series of supercomputers [16]. Many next-generation HPC architectures such as the Aurora system at Argonne [9] and Cori at NERSC [10] have a dragonfly network topology. Each router in the dragonfly has p compute nodes connected to it, with a routers in each group. Routers within a group are connected by all-to-all links called local channels. Each router has h global channels, which are intergroup connections through which routers in a group connect to routers in other groups. Thus, the radix of a router in the dragonfly is $k = a + p + h - 1$. For balancing the global and local network bandwidth, the recommended dragonfly configuration is $a = 2p = 2h$.

If this recommendation is followed, the total number of groups g in the network is $g = a * h + 1$. Since each group has p nodes and a routers, the total number of nodes N in the network is determined by $N = p * a * g$ [12]. In our dragonfly simulation, we follow the recommended dragonfly configuration.

Research efforts have been carried out to evaluate several routing algorithms on a dragonfly topology. These routing algorithms include minimal routing (MIN), nonminimal routing, adaptive routing [11], [12], [17] and progressive adaptive routing [18]. Minimal routing always takes the shortest path; and since the dragonfly topology has a single global channel connecting each pair of groups, the minimal routing algorithm traverses the same global channel when communicating with nodes of another group. To avoid congesting the single global channel between two groups, one can use randomized nonminimal routing (e.g., Valiant's algorithm), which first diverts the traffic to a randomly selected group and from there to its destination group. However, this approach traverses twice the global channels when compared with minimal routing [11]. For each packet being sent, an adaptive routing algorithm dynamically selects a minimal or nonminimal path based on the network queue length, which approximates the congestion on the local and global channels. With progressive adaptive routing, the decision to route a packet minimally gets re-evaluated at each hop of the source dragonfly group [18].

2.3 Torus networks

A torus is a k-ary n-cube network with $N = k^n$ nodes arranged in an n- dimensional grid having k nodes in each dimension [19]. Each node of a torus network is connected to $2 * n$ other nodes. A torus network node can be identified with a unique n-digit radix k address. Torus networks have been used extensively in the Blue Gene [2], [20], Cray XT, and Cray XE [21], [22] series of supercomputers. Since each torus node is connected to its neighbors via dedicated links, torus networks typically have high throughput for traffic patterns that involve nearest-neighbor communication.

Network designers determine the properties of a torus network primarily by torus dimensionality (i.e., number of dimensions, number of nodes in each dimension) and link bandwidth. With a high number of torus dimensions per node and a limited link bandwidth, the serialization delay of the packet becomes a significant overhead. Likewise, with a limited number of torus dimensions and a higher link bandwidth, the network diameter increases, in turn increasing the end-to-end packet latency [19]. Therefore, when designing a torus network, one must find the right balance of dimensions and channel bandwidth in order to achieve high performance for the target workloads.

3 Workloads for HPC network models

HPC applications tend to have nearest-neighbor or local communication patterns. Such applications yield high performance on networks that exploit communication locality for example, torus networks have direct physical connections between nearest neighbors. However, certain classes of applications, such as those using fast Fourier transforms or adaptive mesh refinement, tend to communicate over a broader range of the network, leading to communication with the far end of the network [23], [24]. Such applications are well suited for network topologies that have a low network diameter, such as the dragonfly [11], [12].

We have used two performance evaluation approaches to explore the design space of the torus and dragonfly network models. First, we use synthetic traffic patterns where workloads are characterized by using stochastic processes with network packets being generated with a specific interarrival time, packet size, and destination distribution [3], [25], [26]. Second, we replay trace-based traffic, where post mortem traces capture the network communication of scientific applications running on leadership-class supercomputers.

3.1 Synthetic workloads

Synthetic traffic patterns are used primarily to stress the network topologies and evaluate the effectiveness of a network with a specific form of traffic. In our prior work [25], [26], we have demonstrated how torus and dragonfly with up to a million simulated network nodes perform with synthetic traffic patterns. We use two variations of two forms of traffic with the dragonfly and torus network models. The first is an adversarial traffic pattern such as nearest-neighbor traffic on the dragonfly, which congests the local and global channels connecting the routers. For the torus networks, we used a diagonal traffic pattern that is a measure of determining how effective the network can sustain its bisection bandwidth [27]. The second variation is a traffic pattern that is useful for evaluating the best-case network performance such as a uniform random traffic pattern on the dragonfly [11] and nearest-neighbor communication on the torus.

3.2 Trace-based workloads

While synthetic traffic patterns are simple to implement and useful in assessing whether the network behaves in an expected way, using scientific application workloads provides a way to evaluate the performance implications of the desired networks on scientific applications.

We have used publicly available HPC network traces provided by the Design Forward Program [28] with our CODES network models. These traces provide detailed instrumentation of large-scale applications including point-to-point and collective operations and are collected by using the DUMPI MPI trace package [29], which provides a library to collect and read these MPI traces. The DUMPI traces include detailed information about the type of MPI operations executed by the application as well as the time spent executing that operation. We have used traces from the following applications that represent a variety of relevant communication pattern and network scale:

(i) **The AMG application workload**: AMG is a parallel algebraic multigrid solver used for linear systems in unstructured grids [30]. The AMG network trace track the communication for a single "V cycle" of the multigrid sequence [14], [28]. AMG achieves parallelism by using data decomposition to divide the grid into equally sized logical 3D data chunks, which exhibits a 3D nearest-neighbor

communication pattern. The network traces of the AMG application used in our experiments have 13,824 MPI ranks, the largest available trace for AMG, with an approximate point-to-point data transfer of 10 GiB in the form of point-to-point messages. The AMG application had a runtime of 7.5 seconds, with 13,824 MPI ranks executed on 576 hosts with 39.66% time spent in communication.

(ii) **Multigrid application workload**: Multigrid is a geometric multigrid cycle from the production elliptic solver, Boxlib, that provides the ability to write parallel, block-structured adaptive mesh refinement codes [31]. The communication pattern of Multigrid involves communication across the diagonal, which has similarities with the neighborhood communication in AMG. However, Multigrid communication is distributed across a larger set of ranks as compared to AMG, and it can be considered as many-to-many communication. The network traces of the Multigrid application used in our experiments have 110,592 MPI ranks, the largest available of the Design Forward traces. The overall data transfer in the form of point-to-point messages is 326 GiB. The Multigrid application had a runtime of 5.199 seconds, with 110,592 MPI processes executed on 4608 hosts with 3.4% time spent in communication.

(iii) **Crystal Router application workload**: Crystal Router is a scalable code that represents the many-to-many MPI communication pattern coming from the application code Nek5000. The application trace is from the extracted kernel of the production application Nek5000 [28]. Crystal Router uses a recursive doubling approach. Each rank conforms to an n-dimensional hypercube and is recursively split into $n-1$ dimensional hypercubes that involves communication among a small group of ranks. The trace of the Crystal Router application used in our experiments has 10,000 MPI ranks, the largest available trace for Crystal Router, with an overall data transfer of around 2 TiB in the form of point-to-point messages. The Crystal Router application had a runtime of 1.37 seconds with 10,000 MPI processes executed on 417 hosts with 89% time spent in communication.

4 Simulation Environment

In this section, we describe the interconnect simulation support in CODES developed to drive the dragonfly and torus network models with synthetic traffic patterns as well as post mortem communication traces from scientific applications. For post mortem traces, we describe how the CODES network models and trace-based workloads interact with each other by using an intermediate "MPI simulation layer" that simulates the semantics of MPI network calls.

4.1 CODES network workloads component

The CODES network workload component is an abstraction layer that allows MPI trace generators/readers such as SST DUMPI [29] to drive CODES network models with real application traces through a consistent network workload API. It also provides the ability to drive network models with synthetic traffic patterns as described in Section 3.1. Our network workload model follows the design principles established in the CODES I/O workload component [32] and applies them to the HPC network traces. We have used network traces described in Section 3.2 to drive the network models via the CODES workload abstraction layer.

4.2 CODES MPI Simulation Layer

In order to accurately simulate the MPI operations coming from scientific applications, the simulation needs to maintain the causality among these operations. The job of the MPI simulation layer is to take the operations from the CODES workload layer and simulate the MPI tasks on top of the network models while maintaining the correct causality order. For example, simulating a MPI_WaitAll operation inhibits sending further messages until a set of messages complete.

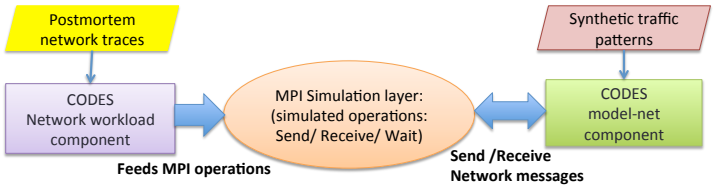

Fig. 2. Interaction of network workload component and model-net component.

Figure 2 shows the interaction of the network models, the network workloads layer and the simulated MPI layer. For trace-based simulations, the MPI operations simulated are MPI sends and receives (blocking and nonblocking) and MPI waits (MPI_WaitAll and MPI_Wait). The MPI send operations are passed onto the model-net layer described in Section 4.3, so that the MPI message can be transported over the network. In order to accurately simulate MPI send, receive, and wait operations, the posted receives are matched with the arrived messages. In the case of blocking send/receive, no further operation is issued until the blocking send or receive is completed. To simulate MPI wait operations, the simulation keeps track of the request IDs of the MPI send/receive operations that have been completed so far. When a MPI_Wait or MPI_WaitAll operation is posted, the list of request IDs in the wait operation is matched against the completed request IDs tracked by the simulation. If all request IDs match, the simulation processes the next MPI operation. If some of the request IDs have not completed so far, the simulation does not proceed with the next MPI operation until it gets notification that all request IDs listed in the wait operations have completed. MPI Wait_any and Wait_some operations are currently not being simulated as they will cause a mismatch of requests completed by the simulation and the trace.

For synthetic network traffic, MPI messages generated according to a specific size, interarrival time, and destination distribution are handed directly to the network models. We can also simulate collective communication operations by using a data-driven approach with both the torus and dragonfly network models, although that is beyond the scope of this paper [13].

4.3 CODES network models

We use a simulation abstraction layer in CODES, "model-net", that allows multiple network models (e.g., torus and

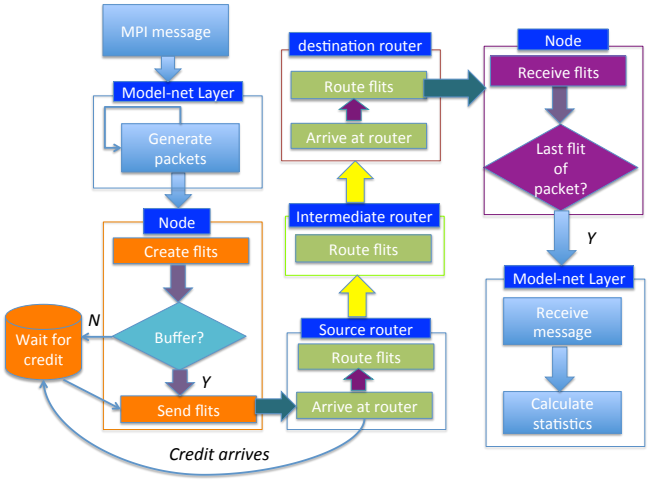

Fig. 3. Event flow in the dragonfly network model (credit flow control shown at dragonfly nodes only).

dragonfly) to be used interchangeably as the underlying network of higher-level storage and application models. The model-net framework in CODES provides a consistent networking API through which one can easily switch multiple network models while making minimal changes to the higher-level models. The model-net framework also unifies common functionality across network models, such as mapping the simulated network entities to the MPI processes executing the parallel discrete-event simulation.

In our dragonfly and torus network models, we have implemented the following features that are common in a typical HPC network:

(i) To regulate traffic on the network, the network models use a flow control methodology based on a credit-based flow control scheme in which the upstream node/router keeps a count of free buffer slots in the virtual channels (VCs) [33] of the downstream nodes/routers [2], [11].

(ii) We model an input-queued virtual channel router for the networks. The router has input and output ports, where each port supports up to v virtual channels (VCs) [33] and each VC has a specific buffer capacity. We assume that the router can send/receive packets in parallel as long as the packets are not being sent/received over the same router link. The dragonfly network uses dedicated high-radix routers where a router has k input ports and k output ports, where k is the radix of the router. For the torus network, a router is embedded on the compute node, with each router having $2n$ ports where n is the torus dimensionality.

4.3.1 Dragonfly network model

The dragonfly network model in CODES supports the network configuration as suggested by Kim et al. [11]. Each router port has multiple virtual channels in order to prevent deadlocks. Our dragonfly model supports minimal, randomized nonminimal (Valiant's algorithm) and adaptive routing algorithms. The randomized nonminimal routing randomly selects a global channel to which packets are diverted for load balancing, as suggested in [11]. The adaptive routing algorithm checks the queue lengths of the minimal and non-minimal ports to select the one having less congestion [18]. With minimal routing, our model uses two virtual channels per router to prevent deadlocks. With nonminimal and adaptive routing, three virtual channels per router are used for to prevent deadlock.

Our dragonfly network model comprises three LP types: a dragonfly node LP, a router LP, and a model-net LP. At the model-net abstraction layer on top of the network models is a model-net LP type through which MPI messages are injected by the MPI Simulation layer described in Section 4.2. These messages are scheduled on the underlying network (e.g., torus or dragonfly) in the form of network packets. Figure 3 shows the event flow in the dragonfly network model. The dragonfly node LP receives the packets from the model-net LP and breaks the packet into flits that are communicated to the dragonfly router LPs. The dragonfly router LPs receive the flits and determine the next router/dragonfly node LP for the flits based on the routing algorithm (minimal, nonminimal, or adaptive routing). All dragonfly virtual channels have a certain buffer capacity in flits. Before injecting flits over the network or passing them over to another channel, the sender dragonfly LP checks for available buffer space on the virtual channel. If buffer space is not available for the next channel, the flit is placed in a pending queue until a credit arrives for that channel. Based on the routing algorithm and destination, the flits may traverse multiple routers before arriving at the destination dragonfly node LP. Once all flits of a packet arrive at the destination node LP, they are forwarded to the receiving model-net LP. When using nonminimal or adaptive routing, each flit is forwarded to a random global channel; as a result, the flits may arrive out of order at the receiving destination node LP. Therefore, we keep a count of the flits arriving at the destination dragonfly node LP and once all flits of a message arrive, an event is invoked at the corresponding model-net LP, which notifies the higher level MPI simulation layer about message arrival.

4.3.2 Torus network model

The CODES torus model closely follows the design features and configuration parameters of the torus networks of the Blue Gene (BG) series of supercomputers. Hence, our model uses realistic design parameters of a torus network, and we can validate our simulation results against the existing Blue Gene torus architecture. Similar to the Blue Gene architecture, our torus model uses a bubble escape virtual channel to prevent deadlocks [2]. Following the specifications of the BG/Q 5D torus network, the CODES torus models have packets with a maximum size of 512 bytes, in which each packet is broken into flits of 32 bytes for transportation over the network [20]. Our torus network model uses dimension-order routing to route packets. In this form of routing, the radix-k digits of the destination are used to direct network packets, one dimension at a time [34].

The torus network model has a two LP types, a model-net LP and a torus node LP. The MPI messages are passed on to the model-net LP by the MPI simulation layer. These messages are scheduled on the underlying torus node LP in the form of network packets, which are then further broken into flits. Each torus node LP is connected to its neighbors via channels having a fixed buffer capacity. Similar to the

dragonfly node LP, the torus network model uses a credit-based flow control to regulate network traffic. Whenever a flit arrives at the torus network node, a hop delay based on the router speed and port bandwidth is added in order to simulate the processing time of the flit. Once all flits of a packet arrive at the destination torus node LP, they are forwarded to the receiving model-net layer LP, which notifies the higher level MPI simulation layer about message arrival.

5 Validation of Network Models

To validate the accuracy of the simulation, one can compare the simulation measurements either with a real architecture or with another validated simulation framework. For the CODES torus model, we validated against real Blue Gene/P and Q architectures. For the CODES dragonfly network model, we validated against the serial time-stepped simulation framework *booksim*. that was used to validate the dragonfly topology proposal [25], [35]. One can argue that dragonfly model can also be validated against real hardware such as Cray XC30. However, Cray XC30's dragonfly topology layout does not follow Dally's a=2p=2h configuration for load-balancing traffic [12].

5.1 Dragonfly network model validation

We verified that the dragonfly network model agrees with the *booksim-1.0* version under a variety of packet arrival rates, routing algorithms, and synthetic traffic patterns. We present here simulation results for the dragonfly model network latency of ROSS and *booksim* with 1,024 simulated nodes and 264 routers, using configuration $p = h = 4$ and $a = 8$. The *booksim's* dragonfly topology uses virtual channels [33] to avoid routing deadlock. Each VC maintains a buffer state that has a depth in packet flits. Similar to the CODES dragonfly model, the *booksim* dragonfly model also uses credit-based flow control. Our previous work provides further information about the specific routing algorithms and configuration parameters used for validation [25].

To make our dragonfly model consistent with the *booksim* dragonfly model, we make two assumptions in our model that comply with *booksim*. First, all packets encounter a fixed delay when being transmitted over global or local channels. For both simulators, the latency is set to 1, 10, and 100 cycles for node channels, local (intragroup) channels, and global (intergroup) channels, respectively. Second, single flit packets are used to avoid routing complications such as those associated with virtual cut through or wormhole routing [11], [12].

Figure 4 compares the communication latency reported by the two simulators using minimal and adaptive routings with uniform random (UR) traffic having varying traffic loads. With minimal routing, our dragonfly model has an average of 4.2% and a maximum of 7% difference from *booksim* results. The adaptive routing resembles the behavior of minimal routing under UR traffic. For the adaptive routing configuration, our dragonfly model reports an average of 3.0% and a maximum of 7.8% difference from *booksim* results.

In the worst-case (WC) traffic, all packets from one group are sent to the next group over the single global channel connecting the two groups. This produces high latency with minimal routing as congestion builds on the single global channel connecting the two groups. The traffic pattern can be load balanced by using either randomized nonminimal or adaptive routing. Nonminimal routing gives slightly under 50% throughput with the worst-case traffic pattern [11], [17]. Figure 5 reports communication latency under worst-case traffic with minimal and adaptive routing. As the load starts increasing, both simulators report a high latency with minimal routing under worst-case traffic. With adaptive routing, both simulators report a better latency than with minimal routing, because adaptive routing senses congestion on the channels and opts for the nonminimal route [11].

Our model results are close to the observed *booksim* results for the minimal and adaptive routing algorithm using both synthetic communication patterns explored by Kim et al. [11], [12]. We believe the small differences lie in the way we model the router architecture. Specifically, *booksim* uses an internal speedup for routers by accelerating the router's cycle counter. We approximated the internal speedup of *booksim* by adjusting the router's internal delays accordingly. Additionally, the simulators use different random number generators.

5.2 Torus network model validation

We validated the accuracy of the CODES torus network model with empirical measurements from the Blue Gene architecture. We compared the latency measurements from our CODES torus model with the `mpptest` benchmark on the Blue Gene/P and Blue Gene/Q architectures [36]. The `mpptest` benchmark measures the performance of MPI message-passing routines with many participating MPI processes, and it can isolate sudden changes in system performance by varying the message size. We used the `mpptest` bisection test, in which each MPI process communicates with exactly one other process such that half of the processes in the communicator are communicating with the other half. One can configure the distance between the MPI processes such that two processes communicating with each other can exchange MPI messages that traverse a fixed number of hops. This strategy also helps measure the bisection bandwidth of a network where packets traversing through a fixed number of hops cross the mid-point of a network.

We measured the MPI performance on Argonne's BG/P system Intrepid and RPI Computational Center for Innovation (CCI) BG/Q AMOS system using `mpptest`. The `mpptest` performance benchmark was executed on 512 compute nodes on the Argonne Intrepid BG/P and 1,024 compute nodes on the CCI BG/Q using bisection traffic pattern with 1 MPI rank per compute node. We used 1 MPI rank per compute node since we were interested in observing the network behavior, not how the nodes internally manage the network traffic. The MPI eager protocol, which uses deterministic routing, was used to measure MPI performance on the BG systems [37]. The torus configuration on the BG/P is 8 x 8 x 8 (1 mid-plane) and on the BG/Q is 8 x 4 x 4 x 4 x 2 (1 rack). The channel bandwidth of the ROSS torus model was configured according to the BG systems: 2 GiB/s (1.8 GiB/s available to user) per torus link on BG/Q and 425 MB/s (374 MB/s available to

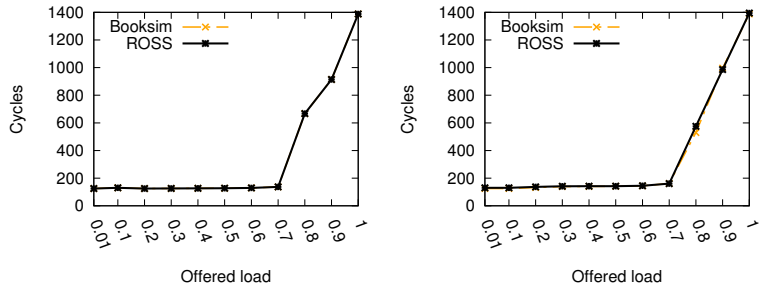

Fig. 4. Average network latency with minimal routing uniform random traffic (left) and adaptive routing uniform random traffic (right).

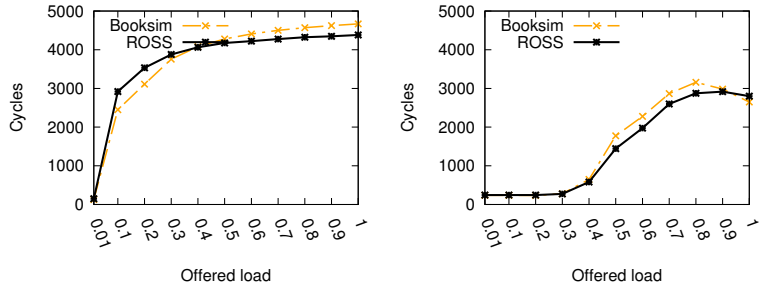

Fig. 5. Average network latency with minimal routing, worst-case traffic (left), and adaptive routing worst-case traffic (right).

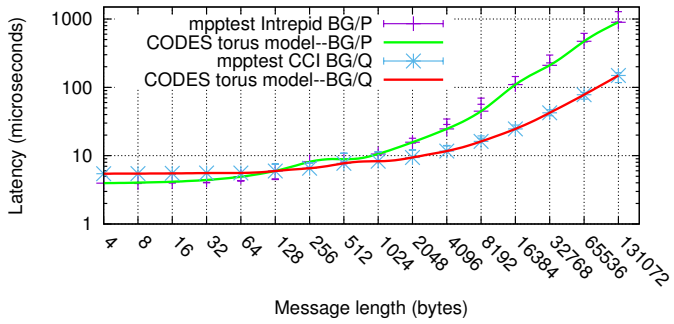

Fig. 6. Latency comparison of `mpptest` on Intrepid BG/P and CCI BG/Q with CODES 3D and 5D torus models. Each packet traverses 8 hops between the source and destination (512 nodes on Intrepid and 1,024 nodes on the CCI BG/Q with 1 MPI rank per node).

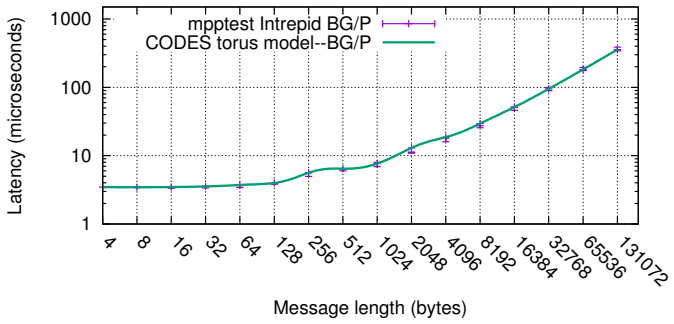

Fig. 7. Latency comparison of `mpptest` on Intrepid BG/P with CODES 3D torus model using nearest-neighbor communication (512 nodes of intrepid BG/P with 1 MPI rank per node).

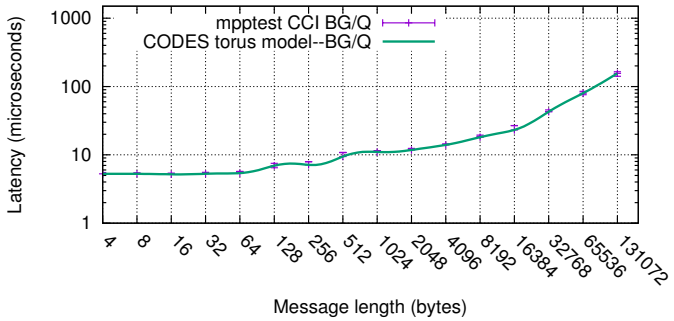

Fig. 8. Latency comparison of `mpptest` on CCI BG/Q with CODES 5D torus model using farthest node communication (hop count set to 11 hops, 1,024 nodes of CCI BG/Q with 1 MPI process per node).

user) per torus link on BG/P. We ran our torus network simulator in the following four configurations simulating an MPI job with one MPI rank per node: (1) 512-node (1 mid-plane) BG/P configuration with messages traversing 8 hops between the source and destination; (2) 1,024-node (1 rack) BG/Q configuration with messages traversing exactly 8 hops between source and destination; (3) 512-node (1 mid-plane) BG/P configuration with messages traversing exactly 1 hop between source and destination nodes; and (4) 1,024-node (1 rack) BG/Q communication with messages traversing 11 hops, the maximum number of hops a message can travel on 1 rack. The number of virtual channels was set to 1 in the torus model, and dimension-order routing was used.

Figure 6 presents a latency comparison of the `mpptest` benchmark on a BG/P midplane and a BG/Q rack vs. the CODES 3D and 5D torus models with 512 and 1,024 nodes, respectively. The distance between the communicating MPI processes is 8 hops for both `mpptest` and the CODES torus models (configurations 1 and 2). One can see close latency agreement between the MPI performance prediction

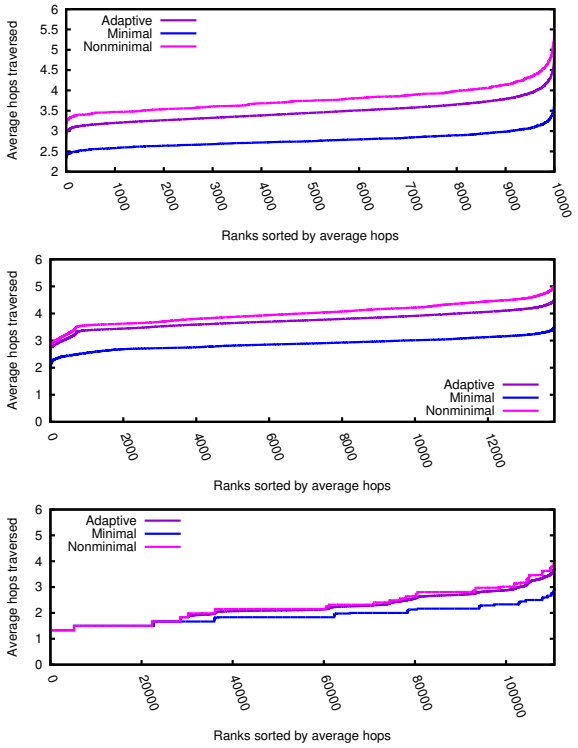

Fig. 9. Average hops traversed on dragonfly network for a 10,000 MPI ranks Crystal Router trace (top), 13K AMG trace (middle), and 110K Multigrid trace (bottom).

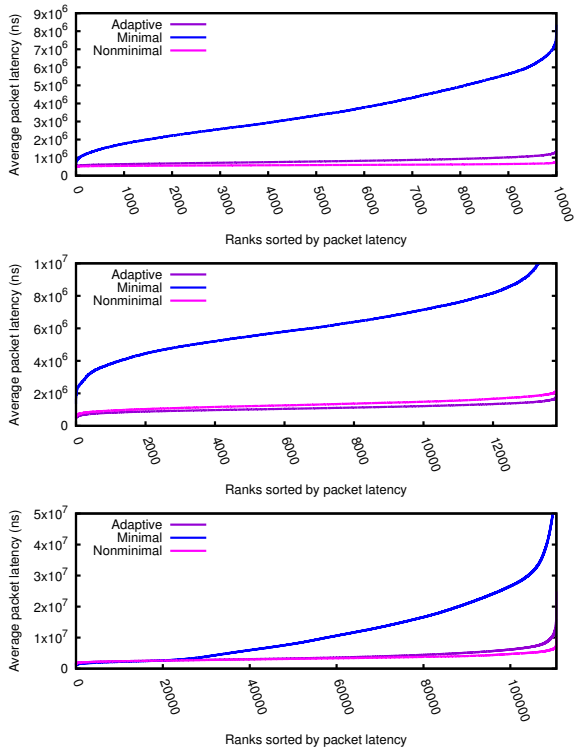

Fig. 10. Average packet latency on dragonfly network for a 10,000 MPI ranks Crystal Router trace (top), 13K AMG trace (middle) and 110K Multigrid trace (bottom).

of the CODES torus model and the `mpptest` benchmark for message sizes ranging from 4 bytes to 130 kilobytes.

Figure 7 provides a latency comparison of the CODES 3D torus model and `mpptest` performance benchmark on the Argonne BG/P Intrepid for 512 compute nodes with nearest-neighbor traffic (configuration 3). The distance between the communicating MPI processes is 1 hop for both `mpptest` and CODES torus models.

Figure 8 presents a latency comparison of the CODES 5D torus model with the `mpptest` performance benchmark on BG/Q using furthest- node communications (configuration 4). The distance between communicating MPI processes is 11 intermediate hops on BG/Q, which is the maximum number of hops that an MPI message can traverse on a 1,024-node torus configuration.

From these statistics, one can see close latency agreement between the MPI performance prediction of the CODES torus model and the `mpptest` performance benchmark for message sizes ranging between 4 bytes and 130 kilobytes.

6 Network Performance Evaluation

In this section, we present two case studies of how the trace-based communication capability can be used to explore the design space of torus and dragonfly network models for representative application workloads. As described in Section 3.2, we use network traces from a Geometric Multigrid application executed on 110K MPI processes, AMG application with 13,824 MPI processes and Crystal Router from Nek5000 with 10,000 MPI processes. For the dragonfly network model, we explore the effect of using minimal, nonminimal, and adaptive routing algorithms on the network performance of these communication patterns. For the torus network model, we examine the effect of varying torus dimensionality on the performance of these communication patterns. For both network simulations, we measure detailed statistics for each simulated MPI rank to study the network performance (average packet latency, number of hops traversed, link busy time, and communication time of the applications). Link busy time is the total simulated time that the buffers stay full for that particular link, which indicates the degree of congestion over the link. Communication time is the maximum simulated time that the simulated MPI ranks spend in communication, including the time to complete blocking MPI operations such as MPI_Wait, MPI_WaitAll, MPI_Send, and MPI_Recvs. For both network models, we have used a buffer depth of 8 KiB (16 KiB for dragonfly global channels) and a packet size of 512 bytes.

All experiments were executed on up to 8 nodes of Argonne's leadership facility (ALCF's) Cooley cluster and RPI Computational Center for Innovation (CCI)'s RSA cluster. ALCF cooley system has 126 compute nodes with each node having 12, 2.4 GHz Intel Haswell processors [38]. The system has a total of 47 Terabytes of RAM with each node having 384GB RAM. CCI's RSA cluster has a total of 34 nodes connected via 56 Gb InfiniBand network. Each node has two four-core 3.3 GHz Intel Xeon processors and 256 GB of system memory [39].

6.1 Dragonfly network model

We use the dragonfly network model to study the impact of minimal, randomized nonminimal, and adaptive routing

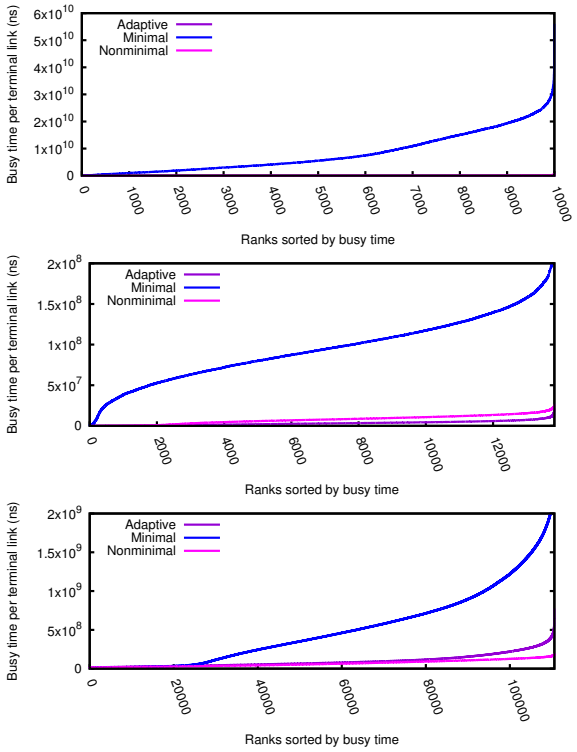

Fig. 11. Busy time of terminal-router links for a 10,000 MPI ranks Crystal Router trace (top), 13K AMG trace (middle) and 110K Multigrid trace (bottom).

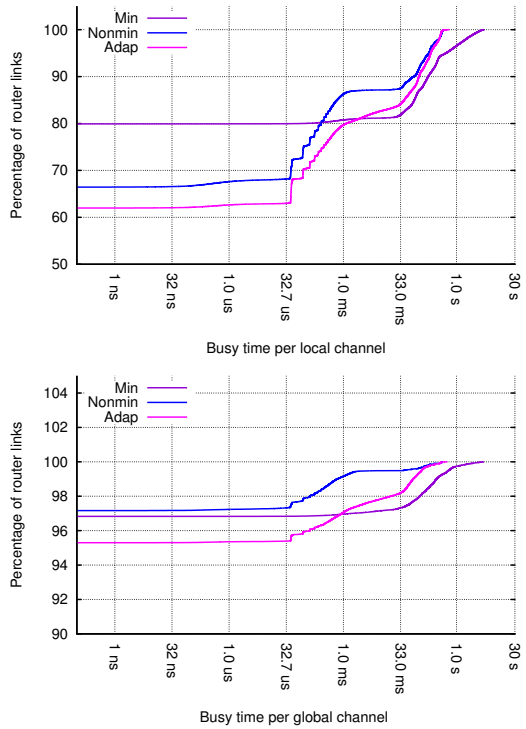

Fig. 14. Busy time of local (left) and global (right) links of the router in dragonfly model for a 110,592 MPI ranks Multigrid trace.

algorithms on the network performance of the AMG, Multigrid and Crystal Router application traces. The dragonfly bandwidth configuration in our experiments is 5.25 GiB/s for local and terminal-router ports, 4.7 GiB/s for global ports, close to Edison, the Cray XC30 system [16]. Figure 9 shows the average number of hops traversed per rank for the application traces with different routing algorithms. Figure 10 shows the average packet latency over the dragonfly network with the application traces using different routing algorithms. Figure 11 shows the busy time of the terminal to router links of the dragonfly network with different routing algorithms. One can see that even though minimal routing traverses the least number of hops as compared with randomized nonminimal and adaptive, the average packet latency and link busy times are lower for adaptive and nonminimal routing algorithms. In all three cases, minimal routing has a high packet latency and busy times. The reason is that the performance of minimal routing depends largely on the traffic pattern. With load-balanced traffic patterns that involved extensive nonlocal communication, minimal routing tends to give better performance than other form of routings do. However, nearest-neighbor traffic patterns can cause extensive communication between groups with minimal routing because it sends all traffic over the same minimally connected route. Among the three application's traffic patterns, AMG has a 3D nearest-neighbor communication pattern whereas Multigrid and Crystal Router also involve some degree of neighborhood communications as opposed to nonlocal communication. When communication is with a closed subset or neighboring group, the single channel between any two groups leads to congestion with minimal routing. Randomized nonminimal routing balances load over the local and global channels by diverting flits to a random intermediate group, which distributes the traffic over the underutilized channels and reduces congestion. As defined in Section 4.3.1, adaptive routing selects a minimal or nonminimal traffic route depending on the queue length for minimal and nonminimal ports. In all these cases, adaptive routing mimics nonminimal routing as it senses congestion over the minimal route. Therefore, it takes the nonminimal path for the majority of the packets.

Figure 12 shows the maximum time taken by a simulated MPI rank over the dragonfly network with all three routing algorithms and application traces. One can see again that nonminimal routing yields a lower communication time than minimal routing. Adaptive routing has a slight overhead in this case because it uses a minimal path for the packets until the channels get congested and it senses congestion over the network. Once it finds congestion over the channels, it starts sending packets nonminimally. These experiments demonstrate that the choice of routing algorithms can have a significant impact on the application performance. Such simulation experiments are an example of how one can choose the appropriate routing algorithm specific to a particular application. One can also use such experiments to explore additional routing options in the dragonfly network such as piggyback routing and progressive adaptive routing [18].

Using our network simulations, we can also capture the statistics on individual network links. With dragonfly networks, some of the local and global links can become hotspots and congest the entire network. Figure 14 shows the cumulative distribution function of the busy times of

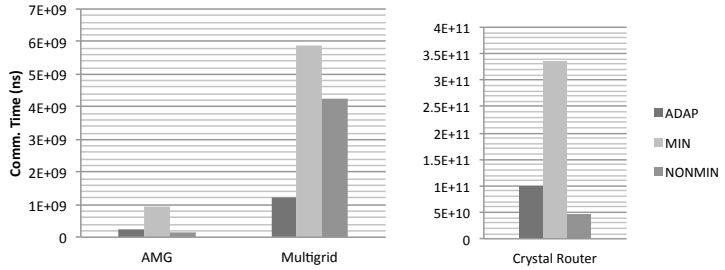

Fig. 12. Communication time of application traces over the dragonfly network model.

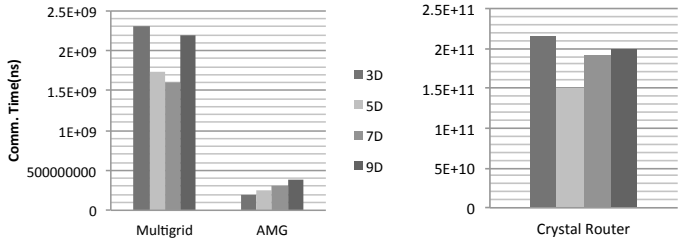

Fig. 13. Communication time of application traces over the torus network model.

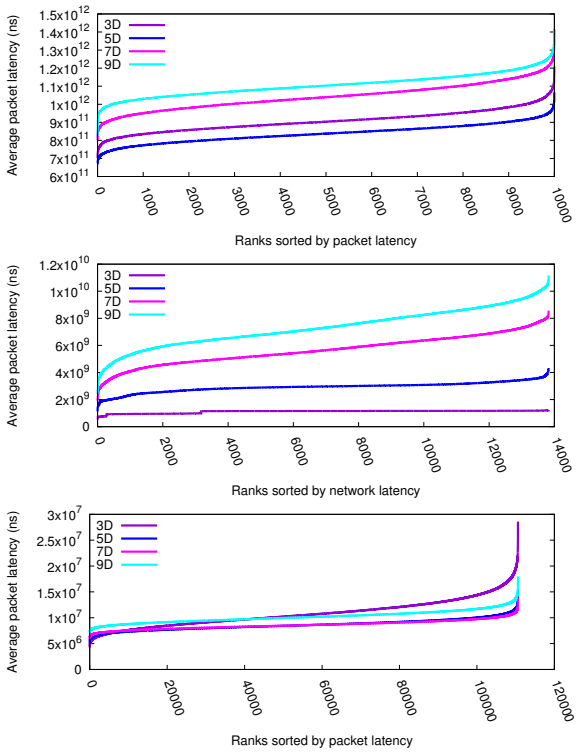

Fig. 15. Average packet latency on torus network for a 10,000 MPI ranks Crystal Router trace, 13K AMG trace and 110K Multigrid trace.

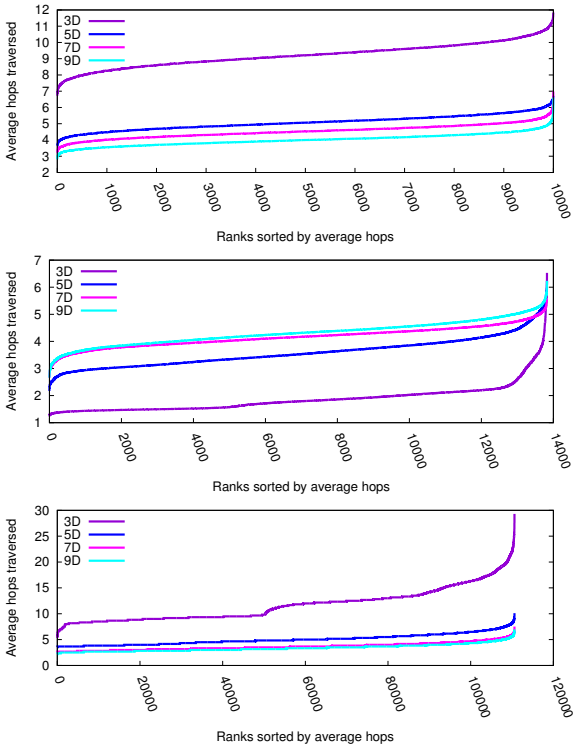

Fig. 16. Average hops traversed on torus network for a 10,000 MPI ranks Crystal Router trace, 13K AMG trace and 110K Multigrid trace.

the local and global channels of the router for the three routing algorithms with the 110K Multigrid application trace. Figure 14 left shows the busy times of local channels of a router whereas the right graph shows the busy times of global channels of a router for a 110K MPI process Multigrid trace. One can see that with minimal routing, roughly 20% of local channels get their buffers full for a high duration of simulated time. In case of nonminimal and adaptive routings, 34% and 39% of the local channels get their buffers full (respectively) but for a relatively shorter duration of simulated time. The busy time of global channels is also higher for minimal routing than adaptive and nonminimal routings. This demonstrates the fact that with minimal routing, a selected group of local and global channels are

excessively congested whereas several other routes remain unutilized. Nonminimal and adaptive routings, on the other hand, make use of a broader group of local and global channels without introducing excessive load over the same group of channels. This experiment is an example of how one can use simulation to identify network hotspots with different application traces and routing algorithms.

6.2 Torus network model

Dimensionality and link bandwidth are the two key properties of a torus network. In this section, we explore the effect of dimensionality and link bandwidth of a torus network on the application performance. We choose four different torus dimensions: 3D, 5D, 7D, and 9D, with each node having a fixed bandwidth of 20 GB/s. The link bandwidth varies with torus dimension where each node has a fixed aggregate bandwidth for all links. For a 3D torus, the link bandwidth is 3.33 GB/s; for a 5D torus, the link bandwidth is 2.0 GB/s; for a 7D torus, the link bandwidth is 1.42 GB/s; and for a 9D torus, the link bandwidth is 1.11 GB/s. Figure 15 shows the average packet latency, and Figure 16 shows the average number of hops traversed on a 3D, 5D, 7D, and 9D torus networks with Crystal Router, AMG and Multigrid application traces. The first case of Crystal Router involves a many-to-many communication pattern, and it involves intense data transfers between a selected number of ranks (usually 5). For this reason, we see the 5D torus giving the lowest average latency, because the communication pattern aligns well with the network and there is sufficient link bandwidth (2 GB/s) available. Even though the 7D and 9D torus networks traverse relatively fewer hops than does the 5D torus, they have a lower link bandwidth that yields a higher packet latency.

The AMG traffic pattern involves 3D nearest-neighbor communication. For this reason, we see a 3D torus network giving the lowest packet latency and traversing the fewest number of hops. The packet latency for 5D, 7D, and 9D torus networks is higher than for the 3D torus in this case because higher-dimensionality tori have lower bandwidth per link. Additionally, the mapping of the 3D nearest-neighbor pattern is not well aligned with higher torus dimensions, leading to a higher number of hops being traversed. We note that the dimension length and symmetry of the torus network play a key role in determining the overall network performance. In the case of AMG, our experiments show that a fully symmetric 3D torus 24x24x24 gives 4.6x better packet latency than an asymmetric torus 36x24x16 (not shown in the graphs).

The Multigrid traffic pattern involves neighbor communications, but in this case the neighborhood of a process is larger than with AMG. In a sense, Multigrid communications are many-to-many with a relatively smaller amount of data transferred between ranks (2–3 MB). For this reason, we see higher dimensionality tori (7D and 9D) traversing the minimum number of hops in this case. The average packet latency is also lowest in the 7D torus. The 3D torus has a poor performance in this case because the process neighborhood is larger and packets have to traverse a large number of hops.

Figure 13 shows the maximum communication time of the three application traces using different torus dimensions. With Crystal Router, the 5D torus network gives a relatively better performance than the others do. For the AMG application trace, the 3D torus has the lowest communication time relative to other torus dimensions. The 7D and 5D tori have a lower communication time than the others do in the case of a Multigrid application trace. The take-away message of these experiments is that the torus dimensionality and link bandwidth can have a significant impact on the overall application communication time. Such simulations can be used to tune the application performance according to the underlying torus network's dimension lengths and bandwidth configurations.

7 Simulation performance

To test the performance of the simulation itself, we recorded the simulation runtime to execute each of these large-scale traces on the torus and dragonfly network models. All simulations were executed on Argonne's Cooley cluster and RPI's RSA cluster on up to 64 cores (8 nodes). By recording the simulation runtime, we wanted to ensure that we are able to execute these large-scale simulations in a reasonable amount of time. Figure 17 shows the simulation runtime of the three application traces on the network models. For both the torus and dragonfly network models, we can see that the Crystal Router application trace consumes the most time. The reason is that the traffic pattern of Crystal Router transfers a substantial amount of data among the MPI ranks. In the case of the 10,000 MPI ranks trace, Crystal Router transfers a total of 2 TiB of data, which translates into large event queues throughout the simulation. Additionally, Crystal Router involves frequent MPI_WaitAll operations after every few sends or receive operations. This frequent synchronization leads to a large number of simulation rollbacks with the ROSS optimistic mode because one out-of-order event can cause a chain of rollbacks of dependent events. On the other hand, the 110K node Multigrid application trace takes an average of 80 minutes to execute on 64 cores of the RSA and Cooley clusters. The AMG application trace is executed on only 8 cores or 1 node of the cluster, but it still takes an average of 30 minutes to execute on the torus network and an average of 2 hours to execute on the dragonfly network. With the help of this experiment, we observe that CODES network simulations can execute on a modest number of compute cores in a reasonable amount of time.

To compare the performance of optimistic discrete-event simulations with traditional serial time-stepped simulations, we present a performance comparison of the CODES discrete-event dragonfly network model with the *booksim* simulation framework [35]. Since *booksim* is a single-threaded simulator, we configured CODES in its serial mode to keep the measurements comparable. By default, *booksim* has three warmup and three measurement phases each having 10,000 cycles. Since we have a single simulation phase in CODES, we configured the simulation end time to have a warmup phase of 30,000 cycles followed by a measurement phase of 30,000 cycles. In this way, we get the same overall simulation end time for both simulators. Both simulators were configured to continuously generate packets after a fixed time interval. The tests were carried

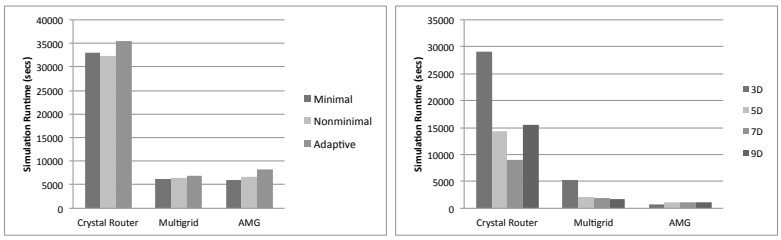

Fig. 17. Simulation runtime for dragonfly (left) and torus (right) network models on 16 to 64 cores of ALCF Cooley and CCI RSA clusters.

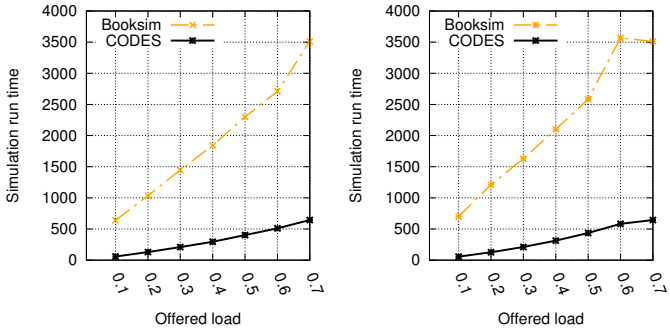

Fig. 18. Simulation runtime (cycles) of CODES vs. booksim for 1K nodes, 264 routers with uniform random traffic using minimal routing (left) and adaptive routing (right)

out on a system with dual 6-core Intel X5650s running at 2.67 GHz. The machine has an available memory of 48 GiB and 12 MB of L3 cache that is shared among all threads/cores. Figure 18 shows the performance of single-threaded ROSS and *booksim* with both UGAL and MIN routing. The CODES dragonfly attains a minimum of 5x up to a maximum of 11x speedup over *booksim* in serial mode with minimal routing and a minimum of 5.3x speedup and a maximum of 12.38x speedup with adaptive routing. These results demonstrate the performance benefits of using a discrete-event scheduling approach over traditional time-stepped series simulations.

Additionally, previous work [40] compares the performance of CODES with state-of-the-art simulation frameworks like SST and BigSim in sequential mode. The performance results demonstrate that CODES torus network model is an order of magnitude faster than the BigSim torus model. We note that the execution time of CODES torus model is 50% lower than SST torus model, even in sequential mode. As noted in [40], SST does not support packet-level fidelity for its network models in parallel builds.

8 Related Work

The BigSim simulator is a parallel discrete-event simulator that has been used to model HPC system architectures, such as Blue Gene and PERCS [6], [41]. It has two modes of execution: online and post mortem. The online mode runs the parallel simulation while executing real applications whereas the post mortem mode simulation uses network traces to feed into the parallel discrete-event simulation. It has been used to explore various intelligent topology-aware mappings and routing techniques to avoid hot spots due to multiple levels in the PERCS topology. The BigSim simulator predicts the application performance for a future machine by obtaining traces through emulation on existing architectures. The simulation for future machines is then driven by these network traces. The PERCS network topology is simulated for up to 307,200 cores at the packet-level detail. However, BigSim is based on the POSE PDES engine that imposes high overheads and limits its scalability. A performance study [40] demonstrates that BigSim is an order of magnitude slower than the CODES network models in sequential execution mode.

Dally and Kim use *booksim*, a serial, time-stepped series simulation framework that can support multiple network topologies including torus, dragonfly, and fat tree [11], [12]. *Booksim* was used to simulate the dragonfly topology on a scale of 1,024 compute nodes and 264 routers with synthetic traffic patterns. The simulation was used to explore the routing choices in the design of dragonfly networks—minimal, nonminimal, and adaptive routing—for various HPC traffic patterns including nearest-group (worst-case) and uniform random synthetic traffic. A modified version of *booksim* was also used to simulate the SlimFly network topology [42], [43] that uses the Moore bound concept to get a small network diameter and large global bandwidth. Simulation results for network sizes of up to 10K nodes are presented by using *booksim*'s minimal, valiant, and adaptive routing.

SimGrid [44] uses a single-node online simulation of MPI applications in which part of the application is executed as a simulation component. To achieve online simulation scalability and speedup, SMPI simulations use analytical models to account for network contention. Given the RAM footprint reduction techniques with online simulation, the performance of the simulation is faster than the simulated time.

Dimemas, an MPI performance analysis tool, is used with a detailed Omnest-based network simulator, Venus, to get trace-driven simulation [3]. Birke et al. use Venus

with the Dimemas and Paraver tools to study the impact of an HPC communication network using real application traces [45]. Similar to our model-net layer in CODES, Venus breaks messages into packets, injects packets into the network, and reassembles packets at the destination. When a complete simulated message arrives at the destination, its information is returned to Dimemas. The network topologies focused on in this work are the mesh, the hierarchical full mesh, and the fat tree. Initial results that show performance improvement on up to 256 LPs are also shown on the Blue Gene platform.

An extreme-scale simulator xSim is presented in [46], [47] that runs the applications on a lightweight parallel discrete-event simulation of concurrent tasks (threads). XSim collects the performance data from the application in the form of processing and network models. The network model also has a virtual message-passing layer through which the MPI application can be executed on a smaller HPC system and its performance parameters can be evaluated on a extreme-scale HPC system. As part of performance tests, the entire NAS parallel benchmark suite up to 16K nodes is executed in the simulated HPC environment.

The Structural Simulation Toolkit (SST) [48] uses a component-based parallel discrete-event model built on top of MPI. SST uses a conservative distance-based optimization to model a variety of hardware components including processors, memory, and networks under different accuracy and details. The network topologies currently supported are two- and three-dimensional meshes, binary tree, fat tree, hypercubes, and flattened 2D butterfly. Detailed SST models run at a small-scale of few simulated nodes. SST macro is a recent addition that enables running DUMPI traces at a large-scale using coarse-grained simulation approach.

In summary, while a number of simulations accurately model HPC network topologies, few of these frameworks have been shown to execute at the size of extreme-scale networks. Most simulation frameworks that can execute network simulations at an extreme scale and support trace-based simulations tend to rely on simplistic analytical network models instead of going to packet-level detail. We do note that the *BigSim* model has been shown to scale to modeling 300K cores, but the PERCS topology layout is different from the torus or dragonfly networks. Additionally, the BigSim simulation framework has a slow execution time [40]. Moreover, the majority of the cited discrete-event simulation frameworks use conservative event scheduling, which leads to limited scalability and slower simulation.

9 Conclusion

With future HPC systems having up to 100,000 nodes and an interconnect bandwidth of up to a terabyte per second, considerable research is in progress to explore network topologies that can yield high performance for a broad range of scientific applications and network communication patterns. Our work addresses the challenges faced by today's network simulation frameworks in exploring the network design space. First, our research relies on detailed flit-level models for network performance prediction instead of simplified analytical models. Second, our network simulations are highly scalable and efficient, thanks to the optimistic event scheduling capability of ROSS. Third, we use a variety of network workloads, both synthetic and trace-based network communication, instead of relying on a particular form of workload.

To the best of our knowledge, this is the first work to explore the largest application traces from the Design Forward program on top of dragonfly and torus network simulations at a flit-level detail in a reasonable amount of time. We have provided examples of how one can investigate the implications of configuration parameters on the behavior of these networks for large-scale application communication patterns. With these network simulations, not only can we see realistic network performance results, but we also can instrument our CODES network models to gain insight into detailed network statistics such as link traffic and busy times in order to figure out why are we seeing this network performance [26]. Overall, this work demonstrates that network designers can use simulation to explore design options with a variety of synthetic and real HPC application network traces.

Acknowledgments

This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computer Research (ASCR) under contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory and the Computational Center for Innovation (CCI) at Rensselaer Polytechnic Institute. We are thankful to Nikhil Jain (UIUC) for his contributions to CODES network models.

References

[1] K. L. Spafford and J. S. Vetter, "Aspen: a domain specific language for performance modeling," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 84.

[2] N. R. Adiga *et al.*, "Blue Gene/L torus interconnection network," *IBM J. of Res. and Develop.*, vol. 49, no. 2.3, pp. 265–276, Mar. 2005.

[3] C. Minkenberg and G. Rodriguez, "Trace-driven co-simulation of high-performance computing systems using OMNeT++," in *Proc. of the 2nd Int. Conf. on Simulation Tools and Techn.*, 2009, pp. 65–72.

[4] D. Jefferson, "Virtual time," *ACM Trans. on Programming Languages and Syst. (TOPLAS)*, vol. 7, no. 3, pp. 404–425, Jul. 1985.

[5] L. Carrington, A. Snavely, X. Gao, and N. Wolter, "A performance prediction framework for scientific applications," in *Computational Science (ICCS) 2003*, 2003, pp. 926–935.

[6] G. Zheng, G. Kakulapati, and L. V. Kalé, "Bigsim: A parallel simulator for performance prediction of extremely large parallel machines," in *Proc. of 18th IEEE Int. Parallel and Distributed Process. Symp.*, 2004, pp. 78–87.

[7] J. Cope *et al.*, "CODES: Enabling co-design of multilayer exascale storage architectures," in *Proc. of the Workshop on Emerging Supercomputing Technol.*, Nov. 2011.

[8] P. D. Barnes, C. D. Carothers, D. R. Jefferson, and J. M. LaPre, "Warp speed: executing time warp on 1,966,080 cores," in *Proc. of the 2013 ACM SIGSIM Conf. on Principles of Advanced Discrete Simulation (PADS)*, May 2013, pp. 327–336.

[9] Argonne Leadership Computing Facility (ALCF), "*Aurora, Argonne's next generation supercomputer*." [Online]. Available: http://aurora.alcf.anl.gov(Accessed on: Apr. 27, 2015)

[10] Department of Energy, "*Cori Cray XC at NERSC*." [Online]. Available: https://www.nersc.gov/users/computational-systems/cori/

[11] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *ACM SIGARCH Comput. Architecture News*, vol. 36, no. 3, pp. 77–88, Jun. 2008.

[12] J. Kim, W. Dally, S. Scott, and D. Abts, "Cost-efficient dragonfly topology for large-scale systems," *Micro, IEEE*, vol. 29, no. 1, pp. 33–40, Feb. 2009.
[13] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Using massively parallel simulation for MPI collective communication modeling in extreme-scale networks," in *Proc. of the 2014 Winter Simulation Conf.*, 2014, pp. 3107–3118.
[14] Department of Energy, "*Design Forward - Exascale Initiative*." [Online]. Available: http://www.exascaleinitiative.org/design-forward (Accessed on: Dec. 31, 2014)
[15] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low-memory, modular Time Warp system," *J. of Parallel and Distributed Comput.*, vol. 62, no. 11, pp. 1648–1669, Nov. 2002.
[16] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc series network," *Cray Inc., White Paper WP-Aries01-1112*, 2012.
[17] M. García *et al.*, "On-the-Fly Adaptive Routing in High-Radix Hierarchical Networks," in *41st Int. Conf. on Parallel Process. (ICPP)*, 2012.
[18] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott, "Overcoming far-end congestion in large-scale networks," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 415–427.
[19] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. Burlington, MA, USA: Morgan Kaufmann, 2004.
[20] D. Chen *et al.*, "The IBM Blue Gene/Q interconnection network and message unit," in *Int. Conf. for High Performance Comput., Networking, Storage and Anal. (SC)*, 2011, pp. 1–10.
[21] S. R. Alam *et al.*, "Cray XT4: an early evaluation for petascale scientific simulation," in *Proc. of the 2007 ACM/IEEE Conf. on Supercomputing (SC)*, Nov. 2007, pp. 1–12.
[22] C. Vaughan, M. Rajan, R. Barrett, D. Doerfler, and K. Pedretti, "Investigating the impact of the Cielo Cray XE6 architecture on scientific application codes," in *IEEE Int. Symp. on Parallel and Distributed Process. Workshops and Phd Forum (IPDPSW)*, May 2011, pp. 1831–1837.
[23] F. Cappello, A. Guermouche, and M. Snir, "On communication determinism in parallel HPC applications," in *Proc. of the 19th Int. Conf. on Comput. Commun. and Networks (ICCCN)*, 2010, pp. 1–8.
[24] S. R. Alam, R. F. Barrett, J. A. Kuehn, P. C. Roth, and J. S. Vetter, "Characterization of scientific workloads on systems with multi-core processors," in *IEEE Int. Symp. on Workload Characterization*, 2006, pp. 225–236.
[25] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Modeling a million-node dragonfly network using massively parallel discrete-event simulation," in *High Performance Comput., Networking, Storage and Anal. (SCC) SC Companion*, 2012, pp. 366–376.
[26] M. Mubarak *et al.*, "A case study in using massively parallel simulation for extreme-scale torus network codesign," in *Proc. of the 2nd ACM SIGSIM/PADS Conf. on Principles of Advanced Discrete Simulation*, 2014, pp. 27–38.
[27] D. Chen *et al.*, "Looking under the hood of the IBM Blue Gene/Q network," in *Int. Conf. on High Performance Comput., Networking, Storage and Anal. (SC)*, 2012, pp. 69–79.
[28] Department of Energy, "*Characterization of the DOE Mini-apps*." [Online]. Available: http://portal.nersc.gov/project/CAL/trace.htm (Accessed on: Dec. 29, 2014)
[29] Sandia National Labs, "*SST DUMPI trace library*." [Online]. Available: http://sst.sandia.gov/using_dumpi.html (Accessed on: Apr. 3, 2015)
[30] *Co-design at Lawrence Livermore National Laboratory*, "*Algebraic Multigrid Solver (AMG)*." [Online]. Available: https://codesign.llnl.gov/amg2013.php (Accessed on: Apr. 19, 2015)
[31] Department of Energy, "*AMR Boxlib*." [Online]. Available: https://ccse.lbl.gov/BoxLib/
[32] S. Snyder *et al.*, "Techniques for Modeling Large-scale HPC I/O workloads," in *High Performance Comput., Networking, Storage and Anal. (SCC) SC Companion*, 2015.
[33] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. on Parallel and Distributed Syst.*, vol. 3, no. 2, pp. 194–205, Mar. 1992.
[34] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distributed Comput.*, vol. 1, no. 4, pp. 187–196, Jan. 1986.
[35] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, "*Booksim 2.0 user's guide*." [Online]. Available: https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim (Accessed on: Oct. 24, 2014)
[36] W. Gropp and E. Lusk, "Reproducible measurements of MPI performance characteristics," in *Proc. of the 6th Eur. PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Mach. and Message Passing Interface*, 1999, pp. 11–18.
[37] M. Gilge, "*IBM System Blue Gene Solution: Blue Gene/Q Application Development*." [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg247948.pdf (Accessed on: Jan. 20, 2015)
[38] "Argonne leadership facility (alcf) cooley, url = https://www.alcf.anl.gov/user-guides/cooley, author=Argonne National Laboratory,."
[39] "Rpi rsa cluster, computational center for innovations, url = https://secure.cci.rpi.edu/wiki/index.php/RSA_Cluster, author=RPI CCI,."
[40] B. Acun, N. Jain, A. Bhatele, M. Mubarak, C. D. Carothers, and L. V. Kale, "Preliminary evaluation of a parallel trace replay tool for hpc network simulations," in *Workshop on Parallel and Distributed Agent-Based Simulations*, 2015.
[41] A. Bhatele, N. Jain, W. Gropp, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *Int. Conf. for High Performance Comput., Networking, Storage and Anal. (SC)*., 2011, pp. 1–11.
[42] M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *Proc. of the Int. Conf. for High Performance Comput., Networking, Storage and Anal. (SC)*, 2014, pp. 348–359.
[43] G. Kathareios, C. Minkenberg, B. Prisacari, G. Rodriguez, and T. Hoefler, "Cost-effective diameter-two topologies: analysis and evaluation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 36.
[44] P.-N. Clauss *et al.*, "Single node on-line simulation of MPI applications with SMPI," in *Proc. of IEEE Int. Parallel & Distributed Process. Symp. (IPDPS)*, 2011, pp. 664–675.
[45] R. Birke, G. Rodriguez, and C. Minkenberg, "Towards massively parallel simulations of massively parallel high-performance computing systems," in *Proc. of the 5th Int. ICST Conf. on Simulation Tools and Techn.*, 2012, pp. 291–298.
[46] T. Naughton, C. Engelmann, G. Vallée, and S. Bohm, "Supporting the development of resilient message passing applications using simulation," in *Proc. of 22nd Euromicro Int. Conf. on Parallel, Distributed and Network-Based Process. (PDP)*, 2014, pp. 271–278.
[47] S. Bohm and C. Engelmann, "xSim: The extreme-scale simulator," in *Int. Conf. on High Performance Comput. and Simulation (HPCS)*, 2011, pp. 280–286.
[48] A. F. Rodrigues *et al.*, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 38, no. 4, pp. 37–42, Mar. 2011.

Misbah Mubarak is a postdoctoral researcher in the Mathematics and Computer Science Division of Argonne National Laboratory. She received her Ph.D. and M.S. from Rensselaer Polytechnic Institute in 2015 and 2011 respectively. She also has experience in developing multitier enterprise applications at CERN, Switzerland, and Teradata corporation.
Christopher D. Carothers is a professor in the Computer Science Department at Rensselaer Polytechnic Institute. He received his Ph.D., M.S., and B.S. from Georgia Institute of Technology in 1997, 1996, and 1991, respectively. He is also the director of Rensselaer's supercomputing facility. His massively parallel discrete-event simulation system ROSS has efficiently executed using nearly 2,000,000 processors on the Sequoia IBM Blue Gene/Q supercomputer.
Robert B. Ross is a senior computer scientist in the Mathematics and Computer Science Division of Argonne National Laboratory and a senior fellow in the Northwestern-Argonne Institute for Science and Engineering. He currently holds several leadership positions at Argonne and in the U.S. DOE computing community, including serving as deputy director of the Scientific Data Management, Analysis, and Visualization Institute and as co-lead of the Data Management component of the DOE Office of Science Exascale Computing activity.
Philip Carns is a principal software development specialist in the Mathematics and Computer Science Division of Argonne National Laboratory and a fellow of the Northwestern-Argonne Institute of Science and Engineering. He has served as the lead developer on notable projects such as PVFS, Darshan, and BMI.

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.