Modeling a Million-Node Slim Fly Network Using Parallel Discrete-Event Simulation

Noah Wolfe, Christopher Carothers
Rensselaer Polytechnic Institute
110 8th St
Troy, NY
wolfen,chrisc@rpi.edu

Misbah Mubarak, Robert Ross, Philip Carns
Argonne National Laboratory
9700 South Cass Avenue
Lemont, IL 60439
mmubarak,rross,carns@mcs.anl.gov

ABSTRACT

As supercomputers close in on exascale performance, the increased number of processors and processing power translates to an increased demand on the underlying network interconnect. The Slim Fly network topology, a new low-diameter and low-latency interconnection network, is gaining interest as one possible solution for next-generation supercomputing interconnect systems. In this paper, we present a high-fidelity Slim Fly flit-level model leveraging the Rensselaer Optimistic Simulation System (ROSS) and Co-Design of Exascale Storage (CODES) frameworks. We validate our Slim Fly model with the Kathareios et al. Slim Fly model results provided at moderately sized network scales. We further scale the model size up to n unprecedented 1 million compute nodes; and through visualization of network simulation metrics such as link bandwidth, packet latency, and port occupancy, we get an insight into the network behavior at the million-node scale. We also show linear strong scaling of the Slim Fly model on an Intel cluster achieving a peak event rate of 36 million events per second using 128 MPI tasks to process 7 billion events. Detailed analysis of the underlying discrete-event simulation performance shows that a million-node Slim Fly model simulation can execute in 198 seconds on the Intel cluster.

Categories and Subject Descriptors

I.6.4 [**Computing methodologies**]: Modelling and simulation

General Terms

Keywords

Slim Fly, Network topologies, Parallel discrete event simulation, interconnection networks

1. INTRODUCTION

Performance of interconnection networks is integral to large-scale computing systems. Current HPC systems have thousands of compute nodes; for example, the Mira Blue Gene/Q system at Argonne has 49,152 compute nodes [23]. Some of the future pre-exascale machines, such as Aurora to be deployed at Argonne National Laboratory, will have over 50,000 compute nodes [13]. The ability of the interconnection network to transfer data efficiently is essential to the successful implementation and deployment of such large-scale HPC systems. There is a trade-off of latency, cost, and diameter among the potential network topologies that currently exist. One topology that meets all three metrics is Slim Fly, as proposed by Besta and Hoefler [4]. High bandwidth, low latency, low cost, and a low network diameter are all properties of the Slim Fly network that make it a solid option as an interconnection network for large-scale computing systems.

In this paper, we present a highly efficient and detailed model of the Slim Fly network topology using massively parallel discrete-event simulation. Validating against the Slim Fly simulator by Besta and Hoefler [4], our Slim Fly model is capable of performing minimal, nonminimal, and adaptive routing under uniform random and worst-case traffic workloads. Our model is also capable of large-scale network modeling; and in this paper, we execute a million-node SlimFly network on the RSA Intel cluster at Rensselaer Polytechnic Institute (RPI) Center for Computation Innovations. In addition, our model has been implemented to execute under optimistic event scheduling using reverse computation and achieves 36 million events per second while maintaining 99% efficiency. This level of performance establishes our Slim Fly model as a useful tool that will give network designers the capability to analyze different design options of Slim Fly networks.

The main contributions of this paper are as follows.

- A Rensselaer Optimistic Simulation System (ROSS) parallel discrete event Slim Fly network model that can simulate large-scale Slim Fly networks at a detailed fidelity and provide insight into network behavior by recording detailed metrics at this scale. The Slim Fly model is also shown to be in close agreement with the Kathareios et al. Slim Fly network simulator [14].
- We simulate and provide a detailed visual analysis of

a Slim Fly network at a scale of 74,000 nodes, inspired by the Argonne Aurora supercomputer.

- This paper also models the largest discrete-event Slim Fly network to date at just over 1 million nodes and crossing the 7 billion committed events mark.
- In terms of the simulation performance itself, a strong-scaling study of our simulation demonstrates that our Slim Fly model is highly scalable and can achieve an event rate of 43 million events per second on 16 nodes, 128 processes of the Intel cluster at RPI [8]

The remainder of the paper is organized as follows. Section 2 provides the network simulation design in terms of the topology, routing algorithms and flow control. We also describe the details of the discrete event simulation implementation. Section 3 presents the validation experiments. Section 4 describes the network and discrete-event simulation performance results. Section 5 discusses related work, and Section 6 summarizes our conclusions and briefly discusses future work.

2. SLIM FLY NETWORK MODEL

In this section, we describe the simulation design of the Slim Fly topology as well as its implementation in the form of a discrete event simulation.

Table 1: Descriptions of symbols used

Topic	Symbol	Description
SF	p	Nodes connected to a router
	N_r	Total routers in network ($N_r = 2q^2$)
	N_n	Total nodes in network ($N_n = N_r * p$)
	k'	Router network radix
	k	Router radix ($k = k' + p$)
	q	Prime power
CODES/ ROSS	LP	Logical Process (simulated entity)
	PE	Processing element (MPI rank)

2.1 Slim Fly Topology

Introduced by Besta and Hoefler [4], the Slim Fly consists of groups of routers with direct connections to other routers in the network similar in nature to the dragonfly interconnect topology. Each router has a degree of local connectivity to other routers in its local group and a global degree of connectivity to routers in other groups. Unlike the dragonfly topology, however, the Slim Fly does not have fully connected router groups. Within each group, each router has only a subset of intragroup connections governed by one of two specific equations based on the router's subgraph membership. Furthermore, all router groups are split into two subgraphs. Each router possesses global intergroup connections only to routers within the opposite subgraph, forming a bipartite graph between the two subgraphs. These global connections are also constructed according to a third equation [4]. Figure 1. shows a simple example of the described structure and layout of the Slim Fly topology.

An important feature of the Slim Fly topology is that its graphs are constructed to guarantee a given maximum diameter. One example set of graphs, which we use in this paper,

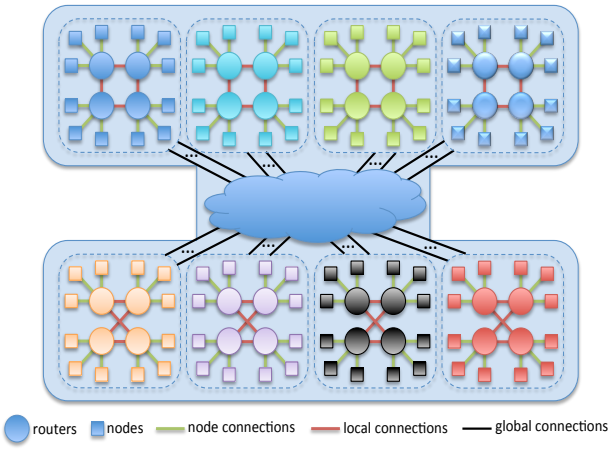


Figure 1: General structure and layout of MMS Slim Fly graphs. Global connections between subgraphs have been generalized for clarity. There are no intergroup connections within the same subgraph. Each router contains one global connection to one router in each of the q-many router groups in the opposing subgraph.

is the collection of diameter 2 graphs, called MMS graphs, introduced by MCKay et al. [16]. MMS graphs guarantee a maximum of 2 hops when traversing the network layer; and because they approach the Moore bound [18], these graphs constitute some of the largest possible graphs that maintain full network bandwidth while maintaining a degree of 2. The 2-hop property holds true while scaling to larger node graphs because the router radix grows as well. For example, routers in a 3K node network require a 28 radix router, while a much larger 1M node network needs a 367 radix router.

Following the methods derived in [12] and summarized and applied to the Slim Fly topology in [4], we developed a separate application to create the nontrivial MMS network topology graphs that govern the interconnection layout of nodes and routers in Slim Fly networks. The process requires (1) finding a prime power $q = 4w + \delta$ that yields a desired number of routers $N_r = 2q^2$; (2) constructing the Galois field and, more important, the primitive element ξ that generates the Galois field; (3) using ξ, computing generator sets X and X' [12] and using them in conjunction with equations 1–3 to construct the interconnection of routers; and (4) sequentially connecting compute nodes to routers.

An example MMS graph is provided in Figure 2. As shown, all routers have three coordinates (s, x, y) indicating the location of the router in the network. The $s \in \{0, 1\}$ coordinate indicates the subgraph, while the $x \in \{0, ..., q-1\}$ and $y \in \{0, ..., q-1\}$ coordinates indicate the router's group and position within the group, respectively. Following the coordinate system, Equation 1 is used to compute the intragroup connections for all groups of subgraph 0 shown in Figure 2. Equation 2 performs the same computationfor all groups in subgraph 1, shown in red. Equation 3 determines the connections between the two subgraphs, shown in blue. For simplicity, Equation 3 connections are displayed only for router$(1, 0, 0)$.

router$(0, x, y)$ is connected to $(0, x, y')$ iff $y - y' \in X$; (1)

router$(1, m, c)$ is connected to $(1, m, c')$ iff $c - c' \in X$; (2)

router$(0, x, y)$ is connected to $(1, m, c)$ iff $y = mx + c$; (3)

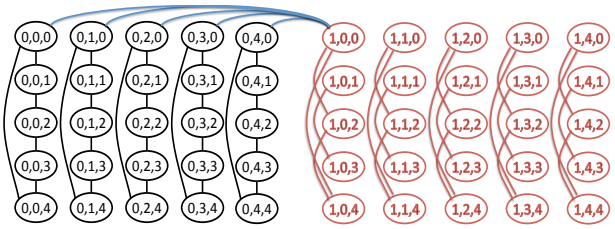


Figure 2: Example MMS graph with $q = 5$ illustrating the connection of routers within groups and between subgraphs.

2.2 Routing Algorithms

Our Slim Fly model currently supports three routing algorithms for studying network performance: minimal, nonminimal, and adaptive routing.

2.2.1 *Minimal Routing*

The minimal, or direct, routing algorithm routes all network packets from source to destination using a maximum of two hops between routers (property of MMS graphs guarantees router graph diameter of two regardless of the size of the graph). If the source router and destination router are directly connected, then the minimal path consists of only one hop between routers. If the source compute node is connected to the same router as the destination compute node, then there are zero hops between routers. In the third case, an intermediate router must exist that shares a connection to both source and destination router so the packet traverses a maximum of two hops.

2.2.2 *Nonminimal Routing*

Nonminimal routing for the Slim Fly topology follows the traditional Valiant randomized routing algorithm [25]. This approach selects a random intermediate router that is different from the source or destination router and routes minimally from source router to the randomly selected intermediate router. The packet is then routed minimally again from the intermediate router to the destination router. The number of hops traversed with valiant routing would be double that of minimal routing. In the optimal case when all three routers are directly connected, the path will be two hops. On the other end of the spectrum each minimal path to and from the intermediate router can have two hops, bringing the maximum number of possible hops to four.

2.2.3 *Adaptive Routing*

Adaptive routing mixes both minimal and nonminimal approaches by adaptively selecting between the minimal path and several valiant paths. To make direct comparisons for validating our model, we follow a slightly modified version of the Universal Globally-Adaptive Load-balanced (UGAL) algorithm [26] shown in [14]. First, the minimal path and several nonminimal paths (n_I) are generated, and their corresponding path lengths L_M and L_I^i, $i \in 1, 2, ...n_I$ are computed. Next, we compute the penalty $c = L_I^i / L_M * c_{SF}$, where c_{SF} is a constant chosen to balance the ratio. Next, the final cost of each nonminimal route $C_I^i = c * q_I^i$ is computed, where q_I^i is the occupancy of the first router's output port corresponding to the path of route i. The cost of the minimal path is simply the occupancy of the first router's port along the path q_M. Then, the route with the lowest cost is selected, and the packet is routed accordingly. With this method, each packet has a chance of getting routed with anywhere from one to four hops.

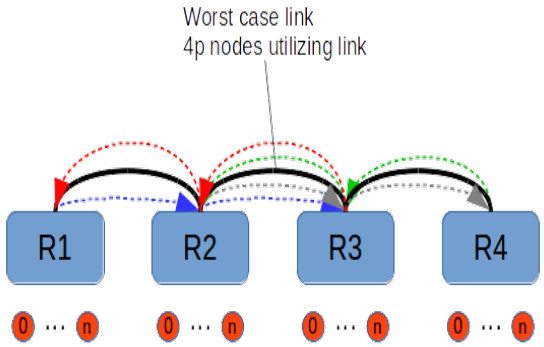


Figure 3: Worst-case traffic layout for the Slim Fly topology.

2.3 Traffic Workloads

To accurately simulate and analyze the network communication using a Slim Fly interconnection topology, we implemented two traffic workloads. The first workload is the uniform random (UR) traffic pattern that selects a random destination compute node anywhere in the network that is different from the source and destination computes nodes. The second workload is a worst-case (WC) traffic pattern that simulates an application that is communicating in a manner that fully saturates links in the network and thus creates a bottleneck for minimal routing. In this workload, each compute node in a router, R1, will communicate to a node within a paired router that is the maximum two hops away. Another pair of routers that share the same middle link with the previous pair of routers will be established to fully saturate that center link. As shown in the example in Figure 3, all compute nodes connected to R1 communicate with nodes connected to R3 along the blue path. Also, the reverse communication is true, because all nodes connected to R3 communicate with nodes connected to R1 along the red path. The router pair R2 and R4 are set up in the same manner communicating along the gray and green paths, respectively. This setup of network communication puts a worst-case burden on the link between routers 2 and 3 as $4p$ nodes are creating $2p$ data flows. With all nodes paired in this configuration, congestion quickly builds up for all nodes in the system and limits maximum throughput to $1/2p$.

2.4 Congestion Control

Both virtual channel [10] and credit based flow control [11] are used in our Slim Fly model to prevent congestion. Following the approach in [4], we discretize our selection of vir-

tual channels to the number of hops a message packet has taken. In other words, for every hop i that a message packet takes, when leaving a router, that packet uses the ith virtual channel. Packets that take a local route and have only one hop will always use $VC0$. Packets that take a global path (assuming minimal routing) will use $VC0$ for the first hop and then $VC1$ for the second hop. Clearly, the optimal number of VCs to use in minimal routing is two. In the case of nonminimal routing such as valiant and adaptive routing, the number of virtual channels used is four, because the maximum possible number of hops in a packet's route is four.

In terms of the implementation, an *output_vc* variable is added to the compute node message state structure and initialized to 0 when a message is created. Each time a router sends a message, it sends the message on the *output_vc* virtual channel and increments *output_vc* so that the next router on the path will use the next corresponding VC.

2.5 Discrete-Event Simulation

Capturing performance measurements of extreme-scale networks having millions of nodes requires a simulation that can efficiently decompose the large problem domain. One such approach, used in this paper, is parallel discrete-event simulation (PDES). PDES decomposes the problem into distinct components called logical processes (LPs), each with its own self maintained state in the system. These LPs model the specific computing components in the simulation such as routers, nodes, and workload processes. LPs interact and capture the system dynamics by passing timestamped event messages to one another. These LPs are further mapped to physical MPI rank processing elements (PEs), which compute their corresponding LPs' events in timestamped order.

We have implemented our Slim Fly model using ROSS [6], a discrete event simulator with support for both conservative and optimistic parallel execution. Conservative execution uses the YAWNS protocol [21] to keep all LPs from computing events out of order. The optimistic event scheduler allows each LP to keep its own local time and therefore compute events out of order with respect to other LPs. Optimistic event scheduling is faster than conservative scheduling. However, this speedup comes at the cost of out-of-order event execution, which is handled by reverse computation [7]. When a temporal anomaly occurs and an event is processed out of timestamp order, all events must be incrementally rolled back to restore the state of the LP to just before the incorrect event occurred.

The rollback process uses a reverse event handler to undo the events. The reverse handlers for the model must be provided by the model programmers. The reverse event handler is a negation of the forward event handler performing inverse operations on all state changing actions. For example, in the Slim Fly model using nonminimal routing, when a message packet arrives at its first router from a node, that router LP performs forward operations in the router-receive forward event handler. The router LP (1) increments the number of received packets, (2) sends a credit event to the sending node LP, (3) computes the next destination by sampling a random number for the random intermediate destination, and (4) creates a new router-send event to relay the packet to the next hop router LP. The reverse event handler needs to undo these operations by (1) decrementing the received packets state variable, (2) sending an anti-message to the sending node to reverse the credit sent, (3) unrolling the random number generator by one, and (4) creating an antimessage (a message indicating an event was issued out of timestamp order and needs to be rolled back in optimistic execution) to cancel the router-send event.

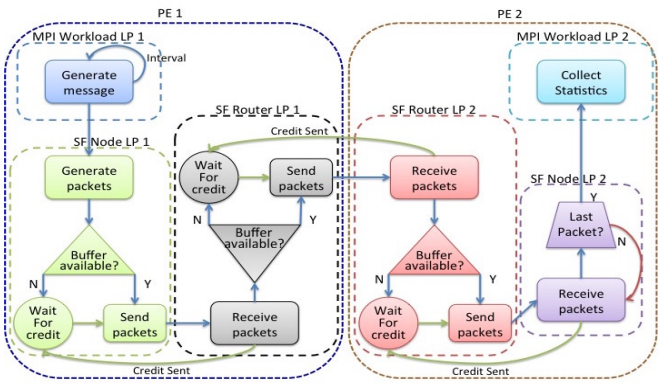


Figure 4: Diagram showing the general execution path of events in the Slim Fly–specific ROSS parallel discrete-event simulation.

We also implemented our Slim Fly model with CODES (Co-Design of Exascale Storage) [9]. This is a simulation toolkit built on top of ROSS that can be used to simulate storage [24] and HPC network systems. CODES helps facilitate the use of HPC network workloads and simulating network communication in the context of discrete event simulations. CODES also provides a range of network models including dragonfly [19], torus [20], and analytical LogGP [2]. Our Slim Fly network model is an addition to the CODES network models. Using the CODES simulation framework, dedicated MPI workload LPs (representing MPI processes attached to compute nodes) generate and receive messages. Messages generated by workload LPs are sent immediately to their corresponding attached compute node LPs with minimal delay in simulation time. Subsequent message-generated events are created at a given interval to approximate the desired network injection rate, incorporating network latencies.

In our Slim Fly model, each LP represents one router, compute node, or simulated MPI workload process in the network. Also, each timestamped event represents a network packet transferring through the network. Figure 4 shows the general structure and event-driven procedure for the Slim Fly network simulation. In this figure, we are running the simulation on two physical cores with one MPI rank per core, resulting in two PEs. The LPs are distributed equally among the two PEs. Events/messages, represented by the arrows between LPs, are transferred between the LPs. For simplicity, only the LPs involved in the example are illustrated.

Upon receiving a message event, the compute node LP decomposes the message into packets and extracts the message destination. The compute node LP computes the next hop

and corresponding output port for each packet using the selected routing algorithm. Prior to sending a packet, the sending node LP checks the occupancy of the selected port and virtual channel. If space exists, the packet is allocated, and a receive event is scheduled on the destination router with a time delay. This time delay incorporates the bandwidth and latency of the corresponding network link. If the buffer is full, the node LP follows credit-based flow control and must wait for a credit from the destination router to open up a space on the corresponding link.

In order to accurately analyze the Slim Fly network, various parameters and statistics are collected and stored in both the LPs and the event messages. These statistics include start and end times of packets on the network, average hops traversed by the packets, and the virtual channels being used.

Once a packet arrives at the router LP, a credit event is sent back to the sending LP to free up space in the sending LP's output buffer. The LP then extracts the destination node ID. The router LP determines the next hop and corresponding output port, once again using the routing algorithm specified. The router also follows the same credit-based flow control scheme as the compute node LP.

After the packet reaches its destination node LP, the node waits for all packets belonging to that message to arrive before issuing a message arrival event on the destination workload LP. At this point, we can collect the statistics stored in the messages, for example, packet latency and number of hops traversed.

3. SLIM FLY MODEL VALIDATION

In this section, we present a comparison with published Slim Fly network results by Kathareios et al. [14] to validate the implementation of our model. The specifics of the IBM-ETH-SF simulator are not provided, but the authors do mention that it is based on the Omnest simulator, which also employs parallel discrete-event simulation. This IBM-ETH collaborative work presents throughput results for a Slim Fly network with the configuration below. The configuration is of particular interest because it yields a total number of compute nodes that is similar to the number of nodes in the future Summit supercomputer [22].

- $q = 13$, $p = 9$, $N_n = 3042$, $N_r = 338$, $k = 28$.

Further network parameters include a 100 Gbps link bandwidth for all links with a latency of 50 ns. The routers utilize virtual channels, a buffer space of 100 KB per port (equally divided among the VCs), and a 100 ns traversal delay. Flow control is done with the use of credits and messages are 256-byte packets. Simulation time for the IBM-ETH-SF was 200 μs with a 20 μs warmup. In our simulation, we include the warmup time in the total execution and therefore run the simulation for 220 μs. The results include minimal, nonminimal, and adaptive routing for uniform random and worst-case traffic workloads. Our simulation results in comparison with the IBM-ETH-SF results are presented in Figures 5, 6, and 7. The metric comparison is throughput percentage and

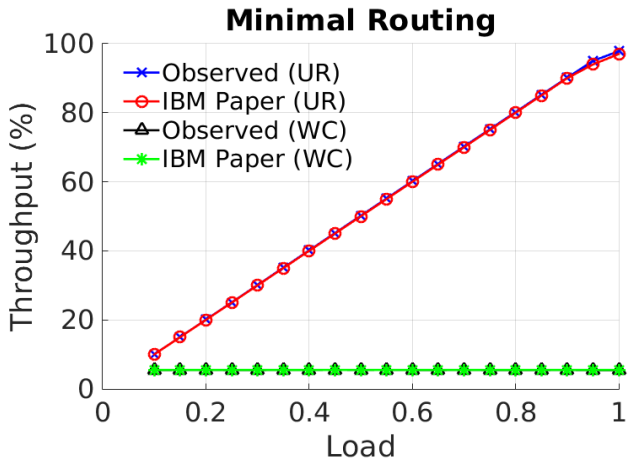


Figure 5: Throughput comparison of minimal routing for uniform random (UR) and worst-case (WC) traffic workloads.

is computed according to Equation 4. Besta and Hoefler [4] approximate the upper bound for bisection bandwidth for the Slim Fly topology to be 71% link bandwidth per node when $p = \lfloor \frac{k'}{2} \rfloor$. Therefore, our simulated 100 Gbps link bandwidth translates to a maximum throughput of 71 Gbps per node. The *observed_throughput* is gained from our Slim Fly model by performing a sum reduction to get the total number of packets transferred by all compute nodes, multiplying by the 256 byte packet size and dividing by the total number of compute nodes.

$$throughput\_percent = \frac{observed\_throughput\_Gbps}{0.71 * 100Gbps} * 100 \quad (4)$$

3.1 Minimal Routing Comparison

Figure 5 presents the throughput analysis for the minimal routing algorithm under input loads varying from 10% to 100% link bandwidth. Focusing on the uniform random workload case, our Slim Fly model closely matches that of the IBM-ETH-SF. As expected, the minimal routing algorithm excels under uniform random workloads. Both simulations show the Slim Fly network throughput matching the injection load from 10% load to about 95% load, at which point the throughput trails off to roughly 98% throughput at 100% load. In the worst-case workload results, the two results are again a close match. Both show a constant 5.5% throughput utilization from 10% to 100% load. Clearly, minimal routing is a poor choice for traffic that is not distributed throughout the network, because minimal routing has no means of selecting alternate routing paths to avoid congested links.

3.2 NonMinimal Routing Comparison

The results comparing throughput analysis for nonminimal routing are shown in Figure 6. In this case, all four sets of results are close together. Under both uniform random and worst-case traffic routing, the Slim Fly network achieves a throughput equal to the injection load until 50% load is

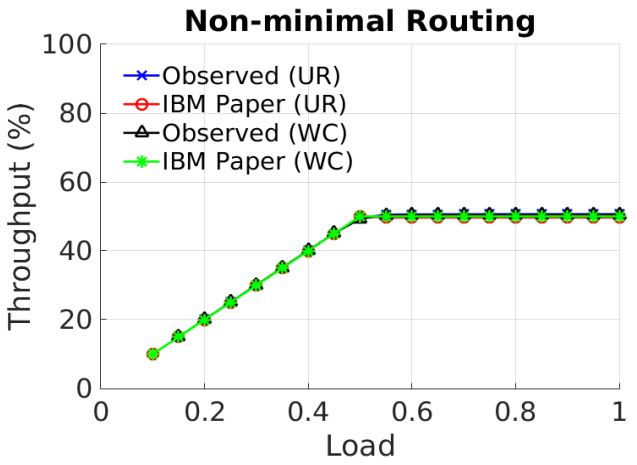


Figure 6: Throughput comparison of nonminimal routing for uniform random (UR) and worst-case (WC) traffic workloads.

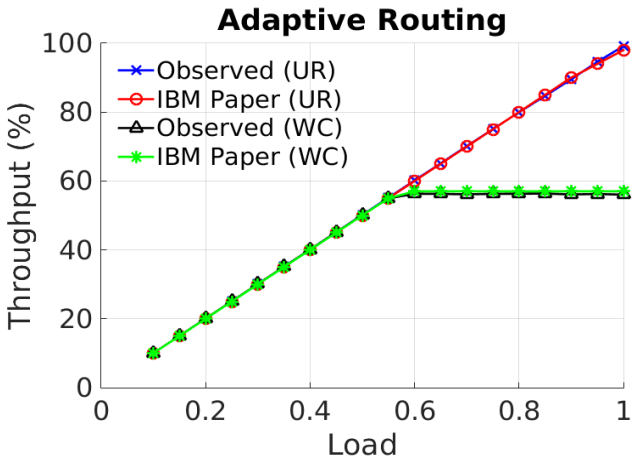


Figure 7: Throughput comparison of adaptive routing for Uniform Random (UR) and Worst Case (WC) traffic workloads.

reached. At this point, the network throughput reaches a bottleneck and maintains just under half-link bandwidth up to 100% injection load. Nonminimal routing underperforms compared with minimal routing for uniform random traffic because the maximum path length of all routes is twice as long in nonminimal routing at four hops compared with two hops in minimal routing. Therefore, nonminimal routing reaches congestion in UR traffic at roughly 50% load, roughly half the load of minimal routing. However, nonminimal routing outperforms minimal routing in worst-case traffic because of its ability to perform a uniform load balancing of traffic as it selects a random intermediate router along its path.

3.3 Adaptive Routing Comparison

The throughput comparison results for the adaptive routing algorithm are shown in Figure 7. In all cases, we set the number of indirect routes, $n_i = 3$, and $c_{SF} = 1$. Once again, the observed results for our Slim Fly model agree with those of the IBM-ETH-SF simulator. In both uniform random and worst-case traffic workloads, the network throughput matches the injection load until 55% load, at which point the worst-case traffic results reach congestion and are limited at 58%. The uniform random traffic results continue with optimal throughput and reach nearly full system throughput at 100% load. Adaptive routing is able to match the performance of minimal routing for uniform random traffic because it can continually select the minimal path for all packets. Adaptive routing outperforms both minimal and nonminimal routing for worst-case traffic because of its ability to dynamically select between the minimal and nonminimal routes.

3.4 Network Visualization

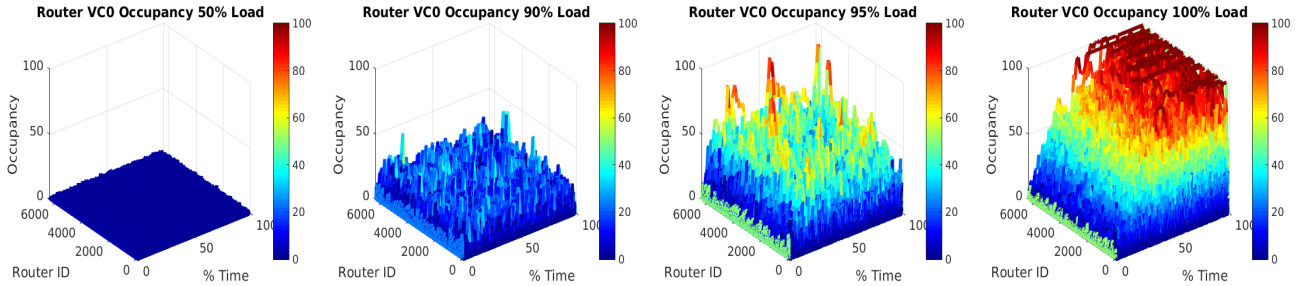
Continuing the analysis, we show visual representations of router occupancy and message sends and receives for both router LPs and compute node LPs during the above simulations. These visualizations provide insight into large, complex network simulations.

The router occupancy metric collects the number of packets sitting in queue waiting for space to open up on the necessary router output port. Since we use VCs for congestion control, the router occupancy metric can be further divided into virtual channels with 2 VCs per port per router used in the case of minimal routing and 4 VCs per port per router used in nonminimal routing. Since virtual channels help alleviate congestion, their occupancy can provide insight to help identify the source of congestion in the network.

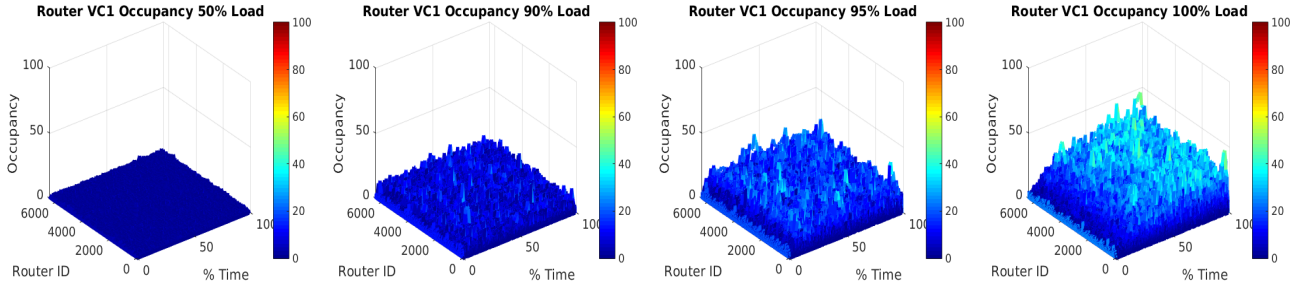
Figure 8 presents a number of graphs visualizing the occupancy of all virtual channels for all ports on all routers in the simulation. All four Slim Fly test cases are from Figure 5, which run the 3K-node Slim Fly model using minimal routing for uniform random traffic. In this case, there are 338 routers with a network radix of 19 and 2 VCs per port. The result is a total of 6,422 ports, each with 2 virtual channels. Figures 8a–8d display the occupancy of VC0 with increasing load from 50% to 100%, and Figures 8e–8h display the same for VC1.

The 3K-node Slim Fly model experiences little congestion until about 90% injection load, where VC0 sees a uniform distribution of roughly 20% congestion in the network. At 100% injection load, the network begins to reach the buffer space limit as packets enter the network at an increased rate, further explaining why we see a slight dip in throughput performance for minimal routing under uniform random traffic in Figure 5. The VC0 buffer fills up first, indicating that the compute nodes are injecting packets into the network faster than the routers can relay them.

In addition to buffer occupancy, the number of message packets sent and received by all routers and compute nodes is visualized over the simulation time. The results are collected during the same simulation in Figures 8d and 8h and are displayed in Figure 9. The first noticeable feature is the large spike in the beginning of the compute node sends (Figure 9a). Occurring at the beginning of the simulation, this spike is a result of the initial packet burst into the system, which is followed by a balancing out as the network reaches a steady state. The same phenomenon is reflected as a slow



(a) VC0 50% Load (b) VC0 90% Load (c) VC0 95% Load (d) VC0 100% Load



(e) VC1 50% Load (f) VC1 90% Load (g) VC1 95% Load (h) VC1 100% Load

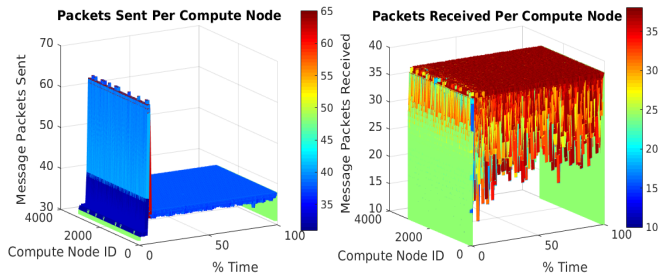
Figure 8: Router occupancy comparison for simulations using UR traffic and minimal routing with increasing injection load. Figures are best viewed in color.

start in router sends and receives plots in Figures 9c and 9d. These figures all resemble the uniform random traffic workload being simulated, except for the initial startup phase. We can also note that the steady-state section of the router sends graph has more of a yellow hue than does the router receives plot, indicating that the routers consistently receive message packets but have slightly more variance in the rate at which they are able to send packets, which can contribute to the slight drop in network throughput observed at 100% load.

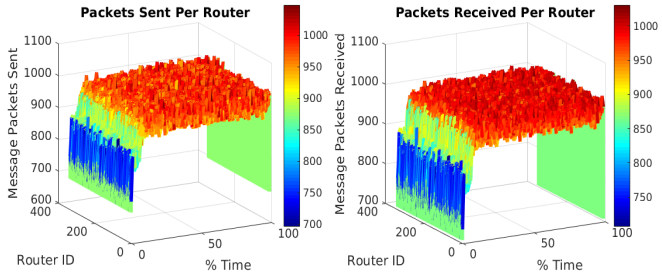
The added capability of visual analysis of these important network metrics not onl helps detect network congestion but also helps identify the time, location, and effect of congestion on the entire network simulation.

4. LARGE-SCALE PERFORMANCE

To show the full capabilities of the ROSS discrete event Slim Fly model simulator, we constructed and analyzed large-scale Slim Fly model configurations. The analysis includes discrete-event compute statistics and strong scaling on the Intel cluster at the Center for Computational Innovations at RPI to emphasize the efficiency of the new Slim Fly simulator. Following the same simulation parameters as in Section 3, we use 100 Gbps link bandwidth with a latency of 50 ns. Routers utilize virtual channels, a buffer space of 100 KB per port, and a 100 ns traversal delay. Each message consists of 256-byte packets. In all the adaptive routing cases, we set the number of indirect routes, $n_i = 3$, and $c_{SF} = 1$ μs. The increased model sizes result in much larger end-to-end runtimes (the time including the initial configuration of LPs in addition to the simulation processing time). In order to generate results in time for the paper submission, the



(a) Compute Node Sends (b) Compute Node Receives



(c) Router Sends (d) Router Receives

Figure 9: Messages sent and received over time for the simulation using UR traffic and minimal routing using 100% load. Figures 9a and 9b show the number of sends and receives sampled over the simulation run time for all the compute nodes. Figures 9c and 9d show the same for all routers in the simulation.

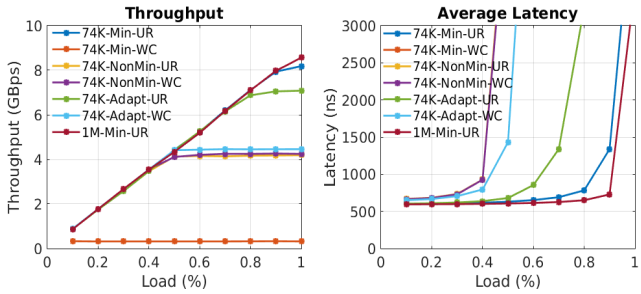


Figure 10: Million-node and 74K-node Slim Fly model performance analysis with uniform random traffic and minimal routing.

simulated time is decreased from 220 μs, as in the previous section, to 100 μs.

4.1 74K-Node Slim Fly Model

In this section, we simulate the Slim Fly model at the scale of Aurora, the future supercomputer to be deployed at Argonne National Laboratory. Aurora is stated to have more than 50,000 nodes, which is significantly larger than Summit [13]. Assuming that the Knights Hill version of the Intel Xeon Phi, which will be the compute architecture for the system, is released with 3 TFLOPs, the future Aurora supercomputer will need to have 60,000 nodes in order to reach the quoted 180 PFlOPS of system performance. A network the size of the future Aurora supercomputing system results in a Slim Fly topology with the following configuration:

- $q = 37$, $p = 27$, $N_n = 73,926$, $N_r = 2738$, $k = 82$.

This 73,926-node model is the smallest configuration that can obtain at least 60,000 nodes without exceeding the $p = \lfloor \frac{k'}{2} \rfloor$ restriction for obtaining optimal system throughput.

We tested the model using all three routing algorithms—minimal, nonminimal, and adaptive—using both uniform random and worst-case traffic workloads. Both system throughput utilization and average packet latency are shown in Figure 10. The results follow the same general trend as was observed for the 3K-node Slim Fly validation model in Section 3. Minimal routing performs nearly full bandwidth under uniform random traffic. Simulating worst-case traffic, minimal routing maintains roughly half the throughput achieved with 10% load of uniform random traffic. Nonminimal routing hits congestion at 50% load under both uniform random and worst-case traffic. Again, this is the result of the nonminimal routing algorithm forcing path lengths to be twice as long as minimal routing. Adaptive routing achieves better throughput over minimal and nonminimal routing for worst-case traffic because it has the added benefit of selecting between the minimal or nonminimal route. The low result at higher loads using adaptive routing may be a result of the 100 μs simulation time not allowing such a large network simulation to reach steady state. This prevents the nonminimal output buffers from fully saturating, thus preventing the adaptive algorithm from selecting the minimal path at high loads.

4.2 Million-Node Slim Fly Model

Further showcasing the scalability of the Slim Fly model, we scale the topology over 1 million nodes. To the best of our knowledge, this is the largest simulated Slim Fly network model. The million-node Slim Fly uses the following configuration:

- $q = 163$, $p = 19$, $N_n = 1,009,622$, $N_r = 53,138$, $k = 255$.

The feasibility of such a large Slim Fly topology must take into account the requirement of a router/switch containing at least 264 ports. Also, utilizing only 19 node connections per router leaves a significant amount of bandwidth on the network side of the router and provides the ability to scale the system up to 6.4 million nodes with up to p=122 nodes per router. This number of nodes reaches the desired $p = \lfloor \frac{k'}{2} \rfloor$ where we still achieve full link bandwidth throughout the system [4]. Unfortunately, this also raises the necessary router radix k' to 367.

Figure 9 presents the throughput and average packet latency results for the million-node simulation using minimal routing and uniform random traffic. Unlike the 74K-node Slim Fly performance, which trails off to a maximum throughput of 8.2 GBps, the million-node model achieves a maximum of 8.6 GBps. As mentioned before, the million-node configuration has only 19 nodes per router, well below the suggested $p = \lfloor \frac{k'}{2} \rfloor$ nodes per router. Therefore, the million-node Slim Fly model will not experience any congestion under uniform random traffic with minimal routing because there is ample network bandwidth to satisfy the much smaller injection load bandwidth.

4.3 Scaling Analysis

In this section, we present the strong-scaling performance of the Slim Fly network model on the RSA Intel cluster at RPI. The system has 34 nodes, each node consisting of two 4-core Intel Xeon E5-2643 3.3 GHz processors and 256 GB of RAM. The million-node Slim Fly model is memory intensive and therefore limits the number of MPI ranks that can be executed per node to 4. The 74K-node Slim Fly model has a much smaller memory footprint, allowing 8 MPI ranks per node. All simulations are executed by using minimal routing, uniform random traffic, and an injection load of 10%.

Additionally, ROSS uses simulation-specific parameters that can be used to tune the simulation performance by controlling the frequency of global virtual time (GVT) calculation [6]. These parameters are the "batch" and "gvt-interval." The batch size is the number of events the ROSS event scheduler will process before checking for the arrival of remote events (events issued from other MPI ranks) and antimessages. The GVT interval is the number of times through the main scheduler loop before a GVT computation will be started. The default values "batch=16" and "gvt-interval=16" are used in the optimistic event scheduling simulations, and the default lookahead value of 1 is used in the conservative executions.

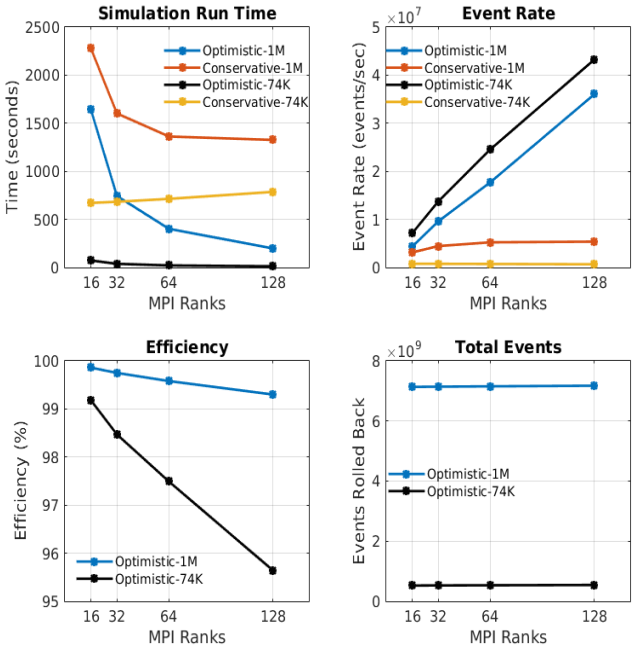


Figure 11: Million-node scaling analysis simulating 100 μs using minimal Routing, UR traffic, and 10% network load.

The scaling performance results are evaluated according to simulation run time (not including simulation configuration time), event rate, event efficiency, and total number of processed events. These measurements provide insight into how well the Slim Fly model performs as a ROSS discrete-event simulation. Event rate is a simple calculation of completed events per second, and event efficiency describes how much work is being performed in the positive direction. Instead of using traditional state saving techniques, ROSS uses reverse event handlers that undo untimely executed events. This technique saves state but requires extra compute to unroll the events. The simulation efficiency measures the amount of reverse computation using Equation 5 [3].

$$efficiency = 1 - \frac{rolled\_back\_events}{total\_events} \quad (5)$$

As shown in Fig. 7, utilizing the ROSS optimistic event scheduler results in an ideal linear speedup of both the Slim Fly million-node model and the 74K-mode model. The largest event rate is achieved running the 74K-node Slim Fly model on 128 MPI ranks, executing just over 43 million events per second and processing 543 million events. Not far behind, the million-node model achieves a rate of 36 million events per second processing 7 billion events. Mapping the 7 billion events to 128 MPI processes translates to each process staying saturated with events and leads to an event efficiency above 99%. Running smaller Slim Fly configurations on as many processes leads to negative efficiency because of less available work. At 128 MPI processes, the smaller 74K-node Slim Fly model has a 3% lower efficiency than does the million-node model but manages to execute events at a 20% faster event rate. The smaller number of events per PE in the 74K-node model translates to less overhead reordering events to maintain timestamp order.

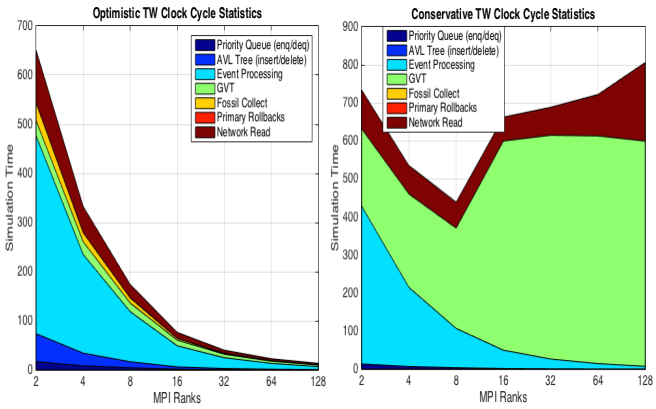
4.4 Discrete-Event Simulation Analysis

In order to understand the performance of the Slim Fly model within the context of the underlying discrete-event simulation engine, this section sheds light on which tasks the model spends the majority of its clock cycles. Figure 12 presents two area plots showing the distribution of time the 74K-node Slim Fly model simulation spends in each phase of the ROSS discrete-event computation. Each execution performs the same 220 μs simulation modeling minimal routing under uniform random traffic with an increasing number of MPI ranks. Also, each simulation uses a batch size of 16 and GVT interval of 16. The same general trends observed in these figures are consistent for all other combinations of non-minimal and adaptive routing with uniform and worst-case traffic.

Focusing first on optimistic execution in Figure 12a, we see the Slim Fly model spends the majority of the time processing events. This trend is consistent regardless of the number of MPI ranks utilized. In addition, the distribution of time spent in each aspect of the simulation stays constant for optimistic scheduling as the number of MPI processes increases. This denotes an ideal distribution of events to LPs, and even further, an ideal distribution of LPs to PEs. It allows the simulation to scale linearly because there is an equal amount of work for each processor, preventing the case where some processors have less work. Less work causes the PE to advance its local time much further than the global virtual time and has a much higher chance of processing an event out of order and forcing a large number of primary and secondary rollbacks. All in all, the Slim Fly model excels under optimistic event scheduling.

In conservative event scheduling using a lookahead value of 1 and starting with 2 MPI ranks, we see a large portion of the Slim Fly model compute cycles spent in GVT. Unlike optimistic scheduling where each PE can maintain its own local time and process events accordingly, conservative execution forces all PEs to maintain the same virtual time, essentially executing in a semi-lockstep manner. This guarantees that no messages are processed out of order, but it requires more interaction from GVT, as shown in Figure 12b. Moving from 2 to 8 MPI ranks, the execution time decreases because of a linear decrease in processing time, while the GVT computation slightly rises. After 8 MPI ranks, the amount of work available for the number of processes decreases to the point that GVT must intervene more often to keep the processes in order, so we experience a large increase in GVT cycles. As the number of MPI processes increases, so does the number of number of PEs the event scheduler must keep locked at the same virtual time. This situation inevitably leads to PEs sitting idle waiting for GVT to advance the time window.

Overall, the ROSS-CODES Slim Fly network model is an efficient tool for modeling large-scale Slim Fly configurations. Using 128 processes on the Intel cluster, the 74K-node model gives the highest event rate of 43 million events per second, while the million-node case processes the most events at 7 billion committed events. Both Slim Fly models achieve strong linear scaling using optimistic event scheduling with



(a) Optimistic Scheduling (b) Conservative Scheduling

Figure 12: Distribution of simulation time for the 74K-node Slim Fly model with minimal routing, UR traffic, and 10% load.

the million-node model performing a 100 μs minimal routing simulation with 10% load under uniform random traffic in just 198 seconds.

5. RELATED WORK

Significant research has been done in simulating large-scale network interconnects and using visualization to gain further insight and understanding. Computing systems are increasingly emphasizing low-latency and low-cost networks.

Liu et al. [15] demonstrate the effectiveness of applying the fat tree interconnect to large data centers. The work focuses on the ability of fat tree networks to perform well under data-center applications at large scale. Unlike our work that currently focuses on HPC workloads, their work focuses on workloads approximating the Hadoop MapReduce model.

Mubarak et al. [19] [20] demonstrate the performance of both the torus and dragonfly network topologies using synthetic workloads at large scale. The simulations are also implemented on top of the CODES and ROSS discrete-event simulation frameworks and run on IBM Blue Gene/P and Blue Gene/Q systems. Our work, in contrast, studies the performance and scaling using an Intel cluster.

Bhatele [5] presents new methods that include visualization for identifying congestion in dragonfly networks and developing new task mappings to allow for efficient use of resources. Still a relatively new topology, the Slim Fly model does not have any real-world implementations. Therefore, it can benefit from the same methods of simulation and visual analysis to predict and limit congestion of possible future Slim Fly systems, especially when executing multiple concurrent tasks.

Acun et al. [1] present TraceR, a tool that replays the BigSim application traces on top of CODES network models. TraceR provides the ability to test CODES network models under real-world production application workloads. In contrast, our work simulates synthetic uniform random and worst-case traffic workloads. Since the TraceR tool has been interfaced with the CODES and ROSS frameworks, it can be experimented with on the Slim Fly model.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a Slim Fly network simulator developed using CODES and the parallel discrete-event simulation framework ROSS. Having implemented minimal, nonminimal, and adaptive routing algorithms specific to the Slim Fly model, we simulated the effectiveness of those routing methods under uniform random and worst-case synthetic traffic workloads. The results of the Slim Fly model have been verified by using published results from Besta and Hoefler [4].

Furthermore, the Slim Fly network model has been shown to scale in network size from a 3,042 and 73,926 node systems, inspired by the future Summit and Aurora supercomputers, to a million-node system topology. Additionally, the Slim Fly model scales linearly in execution up to 128 MPI ranks on the CCI RSA Intel cluster, achieving a peak event rate of 43 million events per second with 543 million total events processed for the 74K-node Slim Fly model. The million-node model achieves 36 million events per second processing 7 billion events.

Through visualization of network simulation metrics like buffer utilization and message packet transfers, we get an insight into the behavior of large-scale HPC networks. These methods provide the ability to view the entire network simulation and identify the cause and effect of congestion.

Further scaling of the million-node model can also be performed to make full use of the simulated $q = 163$ MMS configuration. A Slim Fly configuration can maintain maximum link bandwidth up to $p = \lfloor \frac{k'}{2} \rfloor$ total nodes per router. The million-node model we have simulated has a network radix of $k' = 255$ and $p = 19$ nodes per router. However, this $q = 163$ MMS configuration can use up to $p = 122$ nodes per router to get a 6.4 million-node model capable of maintaining full network throughput. Adding additional realistic workloads that model real world applications is another future direction we plan to experiment with. We also plan to explore other applications for future large-scale Slim Fly network interconnects by possibly simulating a large-scale neuromorphic supercomputing system [17].

Acknowledgments

This work was supported by the Air Force Research Laboratory (AFRL), under award number FA8750-15-2-0078. This was also supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

7. REFERENCES

[1] B. Acun, N. Jain, A. Bhatele, M. Mubarak, C. Carothers, and L. Kale. Preliminary evaluation of a parallel trace replay tool for HPC network simulations. In S. Hunold, A. Costan, D. GimÃl'nez, A. Iosup, L. Ricci, M. E. GÃşmez Requena, V. Scarano, A. L. Varbanescu, S. L. Scott, S. Lankes, J. Weidendorfer, and M. Alexander, editors, *Euro-Par*

2015: Parallel Processing Workshops, volume 9523 of *Lecture Notes in Computer Science*, pages 417–429. Springer International Publishing, 2015.

[2] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. Loggp: Incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '95, pages 95–105, New York, NY, USA, 1995. ACM.

[3] P. D. Barnes, Jr., C. D. Carothers, D. R. Jefferson, and J. M. LaPre. Warp speed: Executing time warp on 1,966,080 cores. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '13, pages 327–336, New York, NY, USA, 2013. ACM.

[4] M. Besta and T. Hoefler. Slim Fly: A Cost Effective Low-Diameter Network Topology. Nov. 2014. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC14).

[5] A. Bhatele. Task mapping on complex computer network topologies for improved performance. Technical report, LDRD Final Report, Lawrence Livermore National Laboratory, Oct. 2015. LLNL-TR-678732.

[6] C. D. Carothers, D. Bauer, and S. Pearce. ROSS: A high-performance, low memory, modular time warp system. In *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation*, PADS '00, pages 53–60, Washington, DC, USA, 2000. IEEE Computer Society.

[7] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto. Efficient optimistic parallel simulations using reverse computation. *ACM Trans. Model. Comput. Simul.*, 9(3):224–253, July 1999.

[8] CCI. RSA cluster, Nov. 2014.

[9] J. Cope, L. N., L. S., C. P., C. C. D., and R. R. Codes: Enabling co-design of multilayer exascale storage architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies (WEST)*, Tuscon, AZ, USA, 2011.

[10] W. Dally. *IEEE Transactions on Parallel and Distributed S*.

[11] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[12] P. R. Hafner. Geometric realisation of the graphs of McKay-Miller-Siran. *Journal of Combinatorial Theory, Series B*, 90(2):223–232, 2004.

[13] Intel. Ushering in a new era: Argonne National Laboratory's Aurora system. Technical report, Intel Corporation, April 2015.

[14] G. Kathareios, C. Minkenberg, B. Prisacari, G. Rodriguez, and T. Hoefler. Cost-effective diameter-two topologies: Analysis and evaluation. Nov. 2015. Accepted at IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC15).

[15] N. Liu, A. Haider, X.-H. Sun, and D. Jin. Fattreesim: Modeling large-scale fat-tree networks for HPC systems and data centers using parallel and discrete event simulation. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '15, pages 199–210, New York, NY, USA, 2015. ACM.

[16] B. D. McKay, M. Miller, and J. Siran. A note on large graphs of diameter two and given maximum degree. *Journal of Combinatorial Theory, Series B*, 74(1):110 – 118, 1998.

[17] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[18] M. Miller and J. Siran. Moore graphs and beyond: a survey of the degree/diameter problem. *The Electronic Journal of Combinatorics [electronic only]*, DS14:61 p., 2005.

[19] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns. Modeling a million-node dragonfly network using massively parallel discrete-event simulation. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, SCC '12, pages 366–376, Washington, DC, USA, 2012. IEEE Computer Society.

[20] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns. A case study in using massively parallel simulation for extreme-scale torus network codesign. In *Proceedings of the 2Nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '14, pages 27–38, New York, NY, USA, 2014. ACM.

[21] D. M. Nicol. The cost of conservative synchronization in parallel discrete event simulations. *J. ACM*, 40(2):304–333, Apr. 1993.

[22] NVIDIA. Summit and Sierra supercomputers: An inside look at the u.s. department of energy's. Technical report.

[23] M. Papka, P. Messina, R. Coffey, and C. Drugan. *Argonne Leadership Computing Facility 2014 annual report*. March 2015.

[24] S. Snyder, P. Carns, J. Jenkins, K. Harms, R. Ross, M. Mubarak, and C. Carothers. A case for epidemic fault detection and group membership in HPC storage systems. In S. A. Jarvis, S. A. Wright, and S. D. Hammond, editors, *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, volume 8966 of *Lecture Notes in Computer Science*, pages 237–248. Springer International Publishing, 2015.

[25] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[26] S.-J. Wang. Load-balancing in multistage interconnection networks under multiple-pass routing. *Journal of Parallel and Distributed Computing*, 36(2):189–194, 1996.