A Case Study in Using Massively Parallel Simulation for Extreme-Scale Torus Network Codesign

Misbah Mubarak
Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
{mubarm,chrisc}@cs.rpi.edu

Robert B. Ross
Philip Carns
Mathematics and Computer Science Division
Argonne National Laboratory
Chicago, IL 60439
{rross,carns}@mcs.anl.gov

ABSTRACT

A high-bandwidth, low-latency interconnect will be a critical component of future exascale systems. The torus network topology, which uses multidimensional network links to improve path diversity and exploit locality between nodes, is a potential candidate for exascale interconnects.

The communication behavior of large-scale scientific applications running on future exascale networks is particularly important and analytical/algorithms models alone cannot deduce it. Therefore, before building systems, it is important to explore the design space and performance of candidate exascale interconnects by using simulation. We improve upon previous work in this area and present a methodology for modeling and simulating a high-fidelity, validated, and scalable torus network topology at a packet-chunk level detail using the Rensselaer Optimistic Simulation System (ROSS). We execute various configurations of a 1.3 million node torus network model in order to examine the effect of torus dimensionality on network performance with relevant HPC traffic patterns. To the best of our knowledge, these are the largest torus network simulations that are carried out at such a detailed fidelity. In terms of simulation performance, a 1.3 million node, 9-D torus network model is shown to process a simulated exascale-class workload of nearest-neighbor traffic with 100 million message injections per second per node using 65,536 Blue Gene/Q cores in a simulation run-time of only 25 seconds. We also demonstrate that massive-scale simulations are a critical tool in exascale system design since small-scale torus simulations are not always indicative of the network behavior at an exascale size. The take-away message from this case study is that massively parallel simulation is a key enabler for effective extreme-scale network codesign.

Keywords

High-performance computing, Parallel discrete-event simulation, Massively parallel architectures, Torus networks, Exascale systems.

Categories and Subject Descriptors

I.6.4 [Computing methodologies]: Modeling and simulation

1. INTRODUCTION

A key factor that largely determines the effectiveness of massively parallel systems is its interconnect network. With future exascale systems having up to one million compute nodes [1, 2], considerable research is in progress to determine a suitable network topology that maximizes the bandwidth of a network under different traffic patterns. One viable option is to continue using k-ary, n-dimensional torus networks that have been extensively used in modern supercomputers such as the IBM Blue Gene (BG) series [3, 4], Cray XT, and Cray XE series networks [5]. Another option is to use low-latency, low-diameter interconnect topologies (such as the dragonfly used by Cray XC30 system [6, 7], flattened butterfly [8], or PERCS [9]) that will enable fast communication at large node counts while keeping power usage and physical construction parameters in check.

While a number of network topologies have been proposed for exascale systems, the search for a topology that yields high performance for most scientific applications is still under way. The torus networks are attractive for local communication patterns because they exploit physical locality between compute nodes. Torus networks also have good path diversity, since they offer multiple minimal paths for transporting packets between the source and destination [10]. One disadvantage of torus networks is that they have a high hop count when communication involves far ends of the network.

Simulation and modeling are important tools to answer “what if” questions and guide network topology design decisions. While analytical modeling helps to make initial design decisions about the networks, an accurate and high-fidelity simulation is required to predict the network performance [10]. Such a simulation will not only help explore the design space of the interconnect topologies but also help find a high-performance network configuration by enabling parameter tuning of these topologies. Additionally, modeling exascale systems requires a simulation that can efficiently...
model an interconnect having millions of compute nodes in a reasonable amount of time. The Rensselaer Optimistic Simulation System (ROSS) is a high-performance, low-memory massively parallel discrete-event simulator [11] that can process billions of events in a second.

The potential system architecture for exascale systems is predicted to have $O(100,000)$ or $O(1M)$ compute nodes and a node interconnect bandwidth in the range of 200-400 GB/sec [12, 2]. In this paper, we use ROSS to develop a high-fidelity torus model to simulate 1.3 million compute nodes/endpoints torus network. Our torus model routes packets similar to the Blue Gene architecture, where variable-sized packets are routed in ‘packet-chunks’ of $n \times 32$ bytes each, where $n = 1$ to 16.

The main contributions of this paper are as follows: (1) we developed the methodology for simulating exascale size torus network topology at a packet-chunk level granularity using the optimistic event scheduling and reverse computation capabilities of ROSS [13], in a time frame allowing for parameter exploration of these networks on HPC systems, (2) by using simulation to investigate the network performance at a detailed fidelity, we identify the pros and cons of certain torus configurations with relevant HPC traffic patterns including local and far-end network communication, prior to building them, (3) we demonstrate the need for executing models at scale and show how small node configurations mis-predict network performance trends compared with large-scale node configurations, (4) we have built on our previous work on torus modeling to make our torus network simulation capable of modeling the BG/P and BG/Q class of networks and (5) by using large-scale torus network simulations, we can also instrument our ROSS torus network model to gain an insight into exactly why are we seeing these performance results. Taken together, these contributions underscore the need for a massively parallel simulation capability to enable effective extreme-scale network codesign.

Our work extends previous work in extreme-scale torus network models by Liu et. al [14]. Specifically, we emphasize the following key improvements:

- [14] uses packet-based deterministic routing for a uniform random traffic pattern. According to Dally and Towles [10], reporting simulation results using uniform random traffic only is a common simulation measurement pitfall since uniform random is a benign traffic pattern which naturally balances load across the network. To accurately measure network performance, one has to consider both benign and adversarial traffic patterns.

- [14] uses infinite buffering for packets without any flow-control methodology. Moreover, the packets are buffered and transported as full-sized packets whereas the BG architecture divides the packets further into chunks of 32 bytes for transportation and buffering. Our new model uses finite buffering with token-based flow control at packet-chunk level which closely approximates the BG architecture’s implementation.

- [14] focuses on the implementation of the torus network model and the simulation performance of the ROSS torus model. In this paper, our focus is to use ROSS torus network simulation as a tool to predict and closely approximate the behavior of different synthetic communication patterns on real torus networks.

- [14] performed a preliminary validation of a 1x32x32, 3-D torus on the BG/L using MPI ping-pong test that involved two MPI processes only. In our work here, we validate our updated torus model against the 3-D torus of the BG/P and 5-D torus of BG/Q machines by testing against both near and far-end network communication patterns. In particular, we validate MPI messaging for 1 hop to 11 hops on BG/Q and up to 12 hops on BG/P with variable message lengths (4 bytes - 131K bytes). The mppptest performance benchmark is used to record the latency of MPI point-to-point messages on the BG/P and BG/Q networks and we force the BG eager message protocol for all MPI communication. All MPI processes participate in the point-to-point messaging.

The road-map for this research begins with an overview of torus networks, ROSS optimistic event scheduling, and the experimental platform for our massively parallel simulations (Section 2). We then discuss the design, validation, and a 1.3 million node scalability study of the torus network simulation (Section 3). We also identify the configurations in which the torus network model yields high or limited performance under different traffic patterns. We then present the simulation performance results of our massively parallel torus network model (Section 4). We then present a literature review on torus network simulations (Section 5), followed by conclusions and future work (Section 6).

2. BACKGROUND

In this section, we first introduce the layout and properties of the torus network topology and then explain the optimistic event scheduling that enables ROSS to schedule billions of events per second on HPC systems. We also discuss the massively parallel experimental platform on which we have executed our million node torus simulation.

2.1 Torus Network

A torus is a $k$-ary $n$-cube network with $N = k^n$ nodes arranged in an $N$-dimensional grid with $k$ nodes in each dimension [10]. Each node of a torus network is connected to $2 \times n$ other nodes typically via short electrical cables. Each torus node can be identified with a unique $n$-digit, radix $k$ address. Torus networks have been extensively used for the Blue Gene [3, 4], Cray XT and Cray XE [5, 16] series of supercomputers. Since each torus node is connected to its neighbor via a dedicated link, torus networks can yield high throughput for nearest-neighbor communication patterns. Therefore, they are well suited for applications such as anisotropic mesh adaptation in CFD solvers [17]. Torus networks also offer good path diversity because several distinct paths exist between any given source and destination node. This property can be leveraged to balance load across network channels. Unlike other hierarchical networks like the butterfly and dragonfly [7, 8], torus networks have a higher average number of hop counts traversed.
Network designers determine the properties of a torus network primarily by torus dimensionality and link bandwidth. Physical channel bandwidth is an important factor in the design of torus networks and is a costly resource of an interconnection network [18]. Another key factor is its dimensionality: at the low extreme of dimension, latency is dominated by the high hop count; at the high extreme of dimension, serialization latency dominates due to narrow channel width, according to Dally and Towles [10]. Therefore, while designing a torus network, one has to balance the number of dimensions and channel bandwidth to get high performance.

2.2 Optimistic Event Scheduling with ROSS

Massively parallel ROSS models are made up of thousands to millions of logical processes (LPs), where each LP models a distinct state of the system. The LPs interact with each other via events in the form of time-stamped messages. MPI tasks in ROSS are abstracted as a processor element (PE), where a PE can have multiple LPs. Events destined for an LP on another PE are called remote messages. Using ROSS, we have developed a detailed simulation that models the behavior of the torus network topology for massively parallel architectures with millions of nodes.

ROSS uses a Time Warp synchronization protocol [19] to process events. If the synchronization mechanism detects an out-of-order event, the events are rolled back and re-executed in the correct order. Optimistic scheduling in ROSS has been shown to dramatically improve the parallel performance and reduce the amount of state-saving overhead [11, 20]. The reverse computation improves the scalability of our parallel simulations especially in instances where the model exhibits a low level of look-ahead [21].

In terms of performance, ROSS can process billions of events per second as indicated in a number of recent performance studies [21, 22, 23]. In [22], ROSS has been shown to scale on 7.86 million MPI tasks with an event rate of over 500 million events per second for the PHOLD benchmark model.

2.3 Experimental Platform

The largest network simulations presented in this work were executed on 65,536 MPI tasks (1 rack) of the Mira Blue Gene/Q system operated by the Argonne Leadership Computing Facility. Mira consists of 48 racks with each rack containing 1024 nodes. Each node has 16, 1.6 GHz compute cores. Each core in turn supports four hardware threads, bringing the maximum number of compute threads per node to 64. The Blue Gene/Q also has a five-dimensional torus network that provides both point-to-point and collective communication facilities [24]. By utilizing the threading capability of Blue Gene/Q, we scheduled 64 MPI tasks per node, which provides the best performance for ROSS given its memory pointer-intensive structure. A parallel performance evaluation of our ROSS torus network simulation on a 1.3 million node test-case is presented in Section 4.

3. TORUS NETWORK SIMULATION

To design the ROSS torus model, we closely followed the design features and configuration parameters of the existing torus network of the BG series supercomputers so that (1) we can validate our simulation results of torus network model against the existing BG architecture and (2) our model uses realistic design parameters of a torus network. The

BG/Q torus network uses a 5-D torus network with packets having a maximum size of 512 bytes, where each packet is broken into chunks of 32 bytes each for transportation over the network [4]. The torus network uses multiple virtual channels (VCs) to avoid head-of-line blocking for packets [3]. A bubble-escape virtual channel is used for deterministic routing and to avoid deadlocks [25]. Each VC has buffers for storing packets to be processed. A token-flow control mechanism is used to prevent a VC buffer from over-flowing.

To align the ROSS torus network model with the BG configuration, we model two LP types in the ROSS torus model: The MPI process LP and the torus node LP. The MPI process LP generates MPI messages that are forwarded to the torus node LP in the form of network packets. Each torus node LP is connected to its neighbors via channels having a fixed buffer capacity. The ROSS events in the torus model are of three types:

1. MPI messages: these messages are sent to/from MPI process LP.
2. Torus network packets: MPI messages are passed on to the torus node LPs by dividing them into network packets where the torus node LPs send and receive network packets. A full-sized network packet can be up to 512 bytes in our torus model.
3. Network packet chunks: similar to the BG network, when communicating between torus node LPs, the network packets are further divided into packet chunks of 32 bytes each.

Figure 1 shows the event flow of the ROSS torus network model. Each torus node LP has an incoming and outgoing virtual channel with VC buffers. The VC buffers are also allocated in the form of 32-bytes packet-chunks, thereby incorporating virtual channel flow control. The MPI process LP generates MPI messages that are divided into packets and sent to the torus node LP. The torus node LP divides the packets into chunks and injects them into the network if the VC buffer space is available. Before sending the packet-chunks over a VC, the torus node checks for the available

![Figure 1: Basic discrete-event flow of the ROSS torus model.](image-url)
buffer space. If the buffer space is not available for the next torus node, the packet-chunk keeps waiting for the next buffer slot in its current buffer location. Whenever a packet-chunk arrives at a torus node LP, it sends a credit event back to the sender torus node LP indicating that the buffer can be reused for the next packet-chunk in the queue. We model a simple input-queued router in which whenever a packet-chunk arrives at the router, a hop delay based on the router speed and port bandwidth, is added to simulate the processing time of the packet-chunk. The router can forward multiple packet-chunks at a time as long as the path from the input to output VC is available. If the path from the input VC to output VC is not available, the packet-chunk is queued and begins only when the earlier packet-chunks have processed.

The ROSS torus network model uses dimension order routing to route packets. With such routing, the radix-k digits of the destination are used to direct network packets, with one dimension at a time [26]. Although the BG architecture supports both dynamic and dimension-order routing, the dynamic routing for BG is strictly vendor specific, and details of this routing algorithm are not available [3, 4]. Therefore, in this paper, the results are presented with dimension order routing only.

### 3.1 Validation of Torus Model with Blue Gene Architectures

To verify the accuracy of our torus network model, we validated the latency results of our ROSS torus model against the mpptest benchmark for measuring MPI performance [27]. Mpptest measures the performance of MPI message passing routines with many participating MPI processes, it can isolate sudden changes in system performance by choosing various message sizes. In the mpptest bisection test, 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 CCI’s BG/Q ‘Amos’ system using mpptest. We ran the mpptest performance benchmark on 512 compute nodes on Argonne Intrepid BG/P and 1,024 compute nodes on CCI BG/Q using bisection traffic pattern with 1 MPI rank per compute node. We used only 1 MPI rank per compute node as we were interested in observing the network behavior, not how the nodes internally manage the contention for the network. The MPI eager protocol,
which uses deterministic routing, was used to measure MPI performance on the BG systems since the BG allows one to force the eager or rendezvous protocol on messages [24]. 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). We ran our torus network simulator in the following four configurations simulating an MPI job with one MPI rank per node:

1. BG/P configuration with 8 x 8 x 8 (1 mid-plane) nodes with messages traversing 8 hops between the source and destination.

2. BG/Q configuration with messages traversing exactly 8 hops between source and destination.

3. BG/P mid-plane nearest-neighbor configuration with messages traversing exactly 1 hop between source and destination nodes.

4. BG/Q farthest-node communication with messages traversing 11 hops.

The channel bandwidth of the ROSS torus model was configured according to the BG systems: 2 GB/sec (1.8 GB/s available to user) per torus link on BG/Q and 425 MB/sec (374 MB/sec available to user) per torus link on BG/P.

Figure 2 presents a latency comparison of the mpptest benchmark on a BG/P mid-plane and a BG/Q rack vs. the ROSS 3-D and 5-D tori models with 512 and 1,024 nodes respectively. The distance between the communicating MPI processes is 8 hops for both mpptest and the ROSS torus models (configurations 1 and 2). One can see close latency agreement between the MPI performance prediction of the ROSS torus model and the mpptest benchmark for message size ranging from 4 bytes to 130 Kilobytes.

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

Figure 4 presents the latency comparison of the ROSS 5-D torus model with the mpptest performance benchmark on BG/Q using farthest-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 a MPI message can traverse on a 1,024 nodes torus configuration.

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

3.2 A Million-Node Torus Network Model

After validating our ROSS torus model for the BG/P and BG/Q architectures, we set out to scale our torus model to anticipated exascale configurations. Exascale systems are expected to have up to 1 million compute nodes [1]. In this section, we evaluate the network performance trends for a 1.3 million node torus network model for a 5-D (32 x 32 x 32 x 20 x 2 configuration), 7-D (16 x 16 x 16 x 10 x 4 x 4 x 2 configuration), and 9-D (10 x 8 x 8 x 8 x 4 x 4 x 2 x 2 configuration) tori.

While designing a torus network, one has to balance the number of dimensions and channel bandwidth to get high performance for expected applications. In this section, we take a large-scale torus network with a fixed peak bandwidth (i.e. sum of the link bandwidths for all network links is the same) and experiment with the key properties of the torus networks (i.e. torus dimensions and link bandwidths) in order to identify the configuration that yields better network performance. We assume that the overall cost of the network stays relatively the same. To evaluate the network performance, we choose well-known synthetic communication patterns, including nearest-neighbor traffic, to determine performance of local communication patterns [10], and diagonal traffic, to determine the bisection bandwidth of the torus network [15]. We keep the channel bandwidth to a maximum of 2 GB/sec out of which 1.8 GB/s of the bandwidth is available for user data in ROSS, which is the default bandwidth configuration of the BG/Q. We keep the buffer capacity of a channel to 128 full sized packets per channel. Through out the simulation results presented in this section, we inject packets at a specific injection rate during an injection interval of 2 microseconds, followed by which we stop injecting more packets and observe how long the network takes to completely process the injected data. We also collect various network performance statistics such as link utilization, aggregate buffer queue utilization and the average number of hops traversed by packets to monitor the progress of injected data over the network.

3.2.1 Nearest Neighbor Traffic Pattern

The torus networks yield high throughput for applications that frequently communicate within a set of nearest-neighbors. In our synthetic nearest-neighbor traffic pattern, each torus node sends 512-byte MPI messages to all its torus neighbors. Torus networks tend to achieve a large fraction of the peak network bandwidth when communication involves neighboring torus nodes only [24].

A. Link utilization of a 9-D million-node torus: First we observe the performance and link utilization of the 9-D torus model using the 9-D nearest-neighbor traffic pattern. In this case, each torus node sends messages to its immediate neighboring node, which makes a message traverse only 1 hop. The simulation sends messages at a link bandwidth of 1.8 GB/sec which is the maximum bandwidth available

Figure 6: Average (left) and maximum (right) latency of a 1,024 node 5-D, 7-D, and 9-D torus model with 9-D nearest-neighbor traffic (the 9-D nearest-neighbor traffic is executed on 5-D and 7-D tori).
for user data in the BG/Q (1.8 GB/s of 2.0 GB/s is available to user data). The ROSS torus model simulation injects packets to the neighboring torus nodes to the maximum capacity of channel buffers. During this interval, the torus buffer queues grow quickly, and the maximum fraction of the link bandwidth is utilized (roughly 1.73 GB/sec out of 1.8 GB/sec of available link bandwidth). As shown in Figure 5, given a very high peak injection rate of $2 \times 10^8$ injections per second, the 9-D torus network has maximum link utilization with nearest-neighbor traffic and it takes 120 microseconds of simulation time for all packets in the torus buffer queues to reach their destinations (the channels hold a large number of torus packets in their buffer queues). As the channel queue size starts shrinking, the link bandwidth starts decreasing until all packets have arrived at their destinations. This experiment is an example of how we can instrument our ROSS torus network model to gain insight into the network behavior by collecting simulation performance statistics such as link utilization over simulation time.

**B. Behavior of torus network with 9-D nearest-neighbor traffic:** To compare the effectiveness of various torus configurations on a traffic pattern that communicates with a wide set of neighboring torus nodes, we take a fixed nearest-neighbor traffic pattern for a 9-D torus network and execute the 9-D nearest-neighbor traffic pattern on a 9-D, 7-D, and 5-D tori. In turn, we also test the three tori configurations by executing a 5-D nearest-neighbor traffic on a 9-D, 7-D, and 5-D tori, in order to evaluate the effectiveness of the torus configurations on a traffic pattern that communicates with relatively modest number of neighboring torus nodes. Choosing a fixed traffic configuration and mapping it on different torus dimensions reflects the fact that the communication patterns of applications are based on their own physics and they are not dependent on the system architecture. Therefore, in the next set of experiments, we keep a constant sum of link bandwidths per node and change the dimensions of the torus network. The bandwidth per link for a 5-D, 7-D, and 9-D tori is fixed at 2.0 GB/sec, 1.428 GB/sec, and 1.111 GB/sec respectively; 90% of the bandwidth in each case is available for the user data.

First, to see the impact of scale, we execute a small 1,024 node torus model configured as 5-D ($4 \times 4 \times 4 \times 4 \times 4$), 7-D ($4 \times 4 \times 4 \times 2 \times 2 \times 2$) and 9-D ($4 \times 4 \times 2 \times 2 \times 2 \times 2$) topologies. The average and maximum nearest-neighbor traffic latency is shown in Figure 6 with a constant on-node bandwidth using the 9-D nearest-neighbor traffic pattern. Next, we take a 5-D, 7-D, and 9-D configuration of a 1.3 million node with a constant on-node bandwidth and execute the 9-D nearest-neighbor traffic pattern on these configurations. Figure 7 shows the performance results of a 1.3 million nodes 5-D, 7-D, and 9-D tori networks. In both 1,024 node and 1.3 million node cases, packets on a 5-D and 7-D tori traverse more than one hop, since we have configured a 9-D torus traffic pattern on a 5-D and 7-D tori networks and the 9-D torus traffic nearest neighbors may not be immediate neighbors on a 5-D and 7-D tori networks.

Looking at the performance results of the above 1,024 node configuration, we observe a dramatically different performance picture from which is shown for the notion of exascale network configuration of 1.3 million nodes. In particular, we see that for the 1,024 node case in Figure 6, the 5-D network outperforms the 7-D network by yielding lower average and maximum latencies. Moreover, for lower injection rates, the 5-D network also outperforms the 9-D torus in terms of average latencies at a small-scale of 1,024 nodes (Figure 6 average latency on the left). We attribute this phenomena to the 5-D torus being symmetric for 1,024 nodes while the 7-D has become asymmetric which lowers overall bisection bandwidth of the network in comparison to the 5-D network. In this case, the 9-D network provides a lower latency at high injection rates because it’s neighbors align exactly with the 9-D nearest-neighbor traffic, thus avoiding the degree of congestion encountered in the 5-D network. On the other hand, for the 1.3 million node case in Figure 7, one can see that when communication involves a large number of MPI processes including nearest-neighbors, despite of their wider channel bandwidth, 5-D and 7-D tori yield 3 to 5 times higher latency than does a 9-D torus network with relatively narrower channel bandwidth. Overall, this experiment demonstrates the importance of modeling this class of network at scale since one could not make reliable predictions about network latency performance at larger scales based on smaller scale configurations for the same number of torus dimensions.

**C. Behavior of torus network with 5-D nearest-neighbor traffic:** As our third test case, we tested the three tori configurations by using the nearest-neighbor traffic pattern of a 5-D torus for 1.3 million node configuration. Figure 8 shows the performance results of this test case, in terms of average and maximum latency, for the 5-D, 7-D, and 9-D torus configurations. With 5-D nearest-neighbor traffic, packets on a 5-D torus network traverse one hop only as they are being sent to direct neighbors. However, packets on a 7-D and 9-D tori may traverse more than one hop, since the first five dimensions of a 5-D nearest neighbor traffic are different on a 7-D and 9-D tori than a 5-D torus. We note that the 9-D nearest-neighbor traffic pattern gives a lower latency when executed on a 5-D torus as opposed to the 5-D nearest neighbor traffic when executed on a 9-D torus (Figures 7 and 8 respectively). In case of a 5-D nearest neighbor traffic pattern executed on a 9-D torus, some of the 9-D torus links are left un-utilized, whereas the remaining links become congested because of the lower link bandwidth (1.11 GB/sec) with an average of 5 to 6 torus hops traversed by the packets. When the 9-D nearest-neighbor pattern is executed on a 5-D torus, all the 5-D torus links are utilized.
at a higher link bandwidth (2.0 GB/sec) with packets following a diverse set of torus paths (on average 10 to 12 torus hops are being traversed in this case). Therefore, one can see that when communication involves a limited set of MPI processes, because of the narrower channel bandwidth, the serialization latency of the 9-D torus dominates its high dimensionality, and the 5-D torus network yields better performance because of its wider channel bandwidth.

D. Findings for nearest neighbor traffic: In the case of nearest-neighbor traffic, we have observed that the network performance depends on the nature of the traffic pattern. Overall, given a choice between moderate number of torus dimensions with wide channel bandwidth and a high number of torus dimensions with narrower channel bandwidth, one should carefully align the number of torus dimensions and bandwidth with the neighborhood communication requirements of the applications. Having more torus dimensions than the application’s communication requirements can lead to load imbalance, because the dimension order routing uses selected neighboring links on a node for communication, leaving the remaining neighboring links under utilized. We also observe that the performance impact of the nearest-neighbor traffic pattern is significantly different on a small-scale torus network than the 1.3 million node torus. At a small-scale, we see the 5-D torus network giving improved performance in terms of average latencies than a 9-D torus at lower injection rates. We also see a 5-D torus performing better than a 7-D torus network at 1,024 nodes. However, these performance trends are entirely different at the 1.3 million node case where the 9-D torus always outperforms the 5-D and 7-D tori. We attribute this performance difference to the symmetric nature of the small-scale 5-D torus and emphasize that reliable predictions about performance of exascale interconnects cannot be done through small-scale simulations.

3.2.2 Diagonal Pairing Traffic

Diagonal pairing in torus networks is an effective communication pattern for measuring the bisection bandwidth of a network. The bisection bandwidth of a torus network is calculated using the following equation \( B_{iw} \) is the bisection bandwidth of the network, \( N \) is the number of nodes in the network, \( M \) is the maximum length of a torus dimension, \( B \) is the link bandwidth:

\[
B_{iw} = N * \frac{4}{M} * B
\]

In the diagonal pairing method, each torus node communicates with exactly one other torus node which is a reflection across the mid-point of each dimension, according to [15]. As shown in Figure 9, each torus node is assigned an index \( k \) for each dimension. Each node with index \( k \) in the \( n^{th} \) dimension communicates with another torus node having index \( L - k - 1 \), where \( L \) is length of the \( n^{th} \) torus dimension. The packets sent from nodes in the mid-point of the network traverse fewer hops than do the packets that are sent from the far end of the torus dimension.

A. Network performance statistics of a million node 9-D torus: Figure 10 shows the bandwidth of a 1.3 million 9-D ROSS torus network model recorded over 1 microsecond intervals with a 9-D diagonal traffic pattern. Unlike the nearest-neighbor traffic pattern, diagonal pairing can quickly fill up the buffer space of the network as packets traverse multiple network hops between the source and destination. Additionally, since the packets cross the notional mid-point of the network and may wait in the buffer queues at intermediate hops, it takes more time for the packets to arrive at their destinations. We model the diagonal pairing traffic scenario in ROSS by injecting the network packets at a very high injection rate. In our ROSS torus model, we stop injecting packets in the simulation right before the network buffers start overflowing. In Figure 10, the torus network configuration is \( 10 \times 8 \times 8 \times 8 \times 4 \times 4 \times 4 \times 2 \times 2 \), which produces a maximum hop count of 25. Due to the nature of this traffic pattern, most of the packets traverse \( h/2 \) hops (\( h \) is the number of hops traversed by a packet), while a few packets traverse more or less than \( h/2 \) hops. Figure 10 also shows the average number of hops traversed by the packets and the aggregate buffer utilization at corresponding points in the simulation. The majority of the packets issued have an average hop count of 13, and the network reaches its peak bisection bandwidth during simulation intervals of 30 to 50 microseconds, while packets traversing an average of 13 hops
are arriving at their destination nodes.

Figure 10: Aggregate bandwidth (GB/sec), average hop count and buffer queue utilization of a 9-D 1.3 million node torus model over 1-microsecond intervals with diagonal pairing traffic at injection rate $3 \times 10^7$. Packets were stopped injecting after 2 microseconds of simulation time.

Figure 11 also shows that the aggregate buffer utilization starts declining after 58 microseconds of simulation time, since majority of the total packets injected reach their destinations by this time. Towards the end of the simulation, only packets with a hop count of 14 and more are left in the queue. This experiment is another example of how we can instrument our ROSS torus network model to understand why we are seeing the performance results by collecting a number of additional simulation statistics like aggregate buffer queue utilization with simulation time, number of average hops traversed with simulation time etc.

Figure 12: Average (right) and maximum (left) latency of a 1,024 node 5-D, 7-D, and 9-D torus model with 9-D diagonal traffic (the 9-D diagonal traffic is executed on 5-D and 7-D tori).

Overall, our simulation results show that the benefits of having higher dimensions for far-end network traffic are not evident at a small-scale of 1,024 nodes since increasing torus dimensions at a small-scale may cause little or no improvement over the bisection bandwidth of the network. However, increasing torus dimensions significantly improves performance for a large network of 1.3 million nodes since we can get a higher bisection bandwidth by carefully configuring the maximum lengths of the torus dimensions even if the bandwidth available per link is narrower.

4. MASSIVELY PARALLEL SIMULATION PERFORMANCE

In this section, we explore the simulation performance of the 1.3 million node torus model. The objective of this study is to demonstrate that with ROSS we are able to simulate
viously demonstrated for this class of network models. To the best of our knowledge, this capability has not been previously demonstrated for this class of network models.

executes efficiently on 65,536 MPI tasks on Mira. To the best of our knowledge, this capability has not been previously demonstrated for this class of network models.

Our torus network model uses reverse computation to support the undo operation as part of optimistic event scheduling within the ROSS model execution framework. The torus model node and MPI process LPs have reverse event handlers such that if any of the MPI message/packet/chunk generation, packet-chunk sending, and MPI message/packet/chunk arrival events are out of order, the incorrect events are rolled back, and the corresponding LP state variables are correctly undone (see Figure 1). To write the reverse event handlers, one has to develop a "state rollback mindset" since every possible update to the state of a LP has to be reversible. However, the size of the reverse event handler code for the torus models is only 1/5 than that of its corresponding forward event handlers. Therefore, with a small amount of effort for writing reverse handlers, the optimistic event scheduling and reverse computation capability in our models enables us to avoid the look-ahead limitations of the simulators that use conservative event processing [28, 29], while at the same time eliminating the traditional time warp overheads associated with LP state saving. As a result, this network model executes efficiently on 65,536 MPI tasks on Mira. To the best of our knowledge, this capability has not been previously demonstrated for this class of network models.

Figure 13: Event rate, run time and efficiency for a 1.3 million node 9-D torus model with nearest-neighbor traffic using 64 MPI tasks per node on 1 Mira rack (65K MPI tasks).

4.1 Reverse Computation

The simulation performance in ROSS is measured by using three parameters: ROSS simulation efficiency, committed event rate, and the time to complete the simulation. These metrics provide a picture of how ROSS performs with various network models and configurations. ROSS event efficiency determines the amount of useful work performed by the simulation. It is defined in Equation 2 [23]:

\[ \text{event\_efficiency} = 1 - \frac{\text{rolled\_back\_events}}{\text{total\_committed\_events}} \]  

(2)

To reduce state-saving overheads, ROSS employs an event rollback mechanism by developing event computations such that they can be reverse processed [30]. The simulator efficiency is inversely proportional to the number of rollbacks. With no rollbacks, the simulator yields 100% efficiency.

Figures 13 and 14 show the ROSS committed event rate, simulation run time, and ROSS event efficiency of a 1.3 million node 9-D torus network model under different traffic workloads. The tests were executed on 1 rack of Mira utilizing all 4 threads per core, with each compute node have 64 MPI processes. As seen in the figures, as the injection rates of the torus model simulation keeps on increasing, the ROSS committed event rate also increases. With maximum injection rates of 100 million packets per second in Figure 13, the ROSS event rate increases to 600 million events per seconds, whereas the simulation run time does not exceed 25 seconds. Also, the ROSS event efficiency of the torus model simulation increases with increasing traffic work loads. As the injection rates increase, ROSS simulation has more events to process than it can roll-back. With more events to process, the ratio of rolled-back events to the total committed events becomes less, which in turn increases ROSS event efficiency (Equation 2). Since the injection rates of the torus

Figure 14: Event rate, run time and efficiency for a 1.3 million 9-D torus model with bisection traffic pattern using 64 MPI tasks per node on 1 Mira rack (65K MPI tasks).

4.2 Parallel Simulation Performance of Torus Network Models

Efficiency (Equation 2). Since the injection rates of the torus

Event rate (million events/sec) Simulation run time (seconds) Simulation Efficiency (% age)

Injections per second (per node)

Injections per second (per node)

Injections per second (per node)

Event rate (million events/sec) Simulation run time (seconds) Simulation Efficiency (% age)

Injections per second (per node)

Injections per second (per node)

Event rate (million events/sec) Simulation run time (seconds) Simulation Efficiency (% age)

Injections per second (per node)
network with bisection traffic are at a smaller scale than that of nearest-neighbor traffic (Figure 14), the variance in ROSS event rate and efficiency is less for bisection traffic. The simulation run time of the torus bisection traffic (Figure 14) is more than the run time of the nearest-neighbor traffic (Figure 13) because the bisection traffic traverses multiple hops (up to 25 hops) whereas the nearest-neighbor traffic traverses only a single network hop.

The ROSS simulation performance measurements show that the run times and event rate of the torus network model are scalable with high packet injection rates. The efficiency of the models increases when we increase the work loads, since more events are processed than the events being rolled back. Therefore the model can efficiently process high injection rates as the simulation is being assigned more work.

5. RELATED WORK

A number of efforts have been done to analytically model and/or simulate torus network and related network topologies.

We first start with the analytic models. Agarwal [31] improves upon the latency analytic results of Dally [32] by considering the node or switch delays in addition to wire delays in the model. These results countered Dally’s at the time by showing that a 3-D network has a lower latency as opposed to a 2-D network when switch delays are four times the wire delay. The results here are concerned with symmetric torus networks where $K$ is the same for all dimensions.

For asymmetric torus networks, Ciciani [33] presents an average latency delay model for an asymmetric 3-D torus network that considers both uni-directional and bi-directional links. Here, configurations are considered up to 1000 nodes (e.g., $10 \times 10 \times 10$) with uniform random traffic flows with Poisson arrival times.

Additionally, Min et al. [34] developed an analytic model for computing communication delays in torus networks with circuit switching under multiple time-scale bursty and correlated traffic. The probability of message blocking in practical torus topologies was calculated using the model. Probabilistic methods were also used to determine traffic characteristics on network channels. The accuracy of the analytical model was validated by comparing analytical results with simulation experiments of the real system.

For serial simulation, Dally et al. developed Booksim which is a cycle-accurate network simulation framework, to study the performance of the torus network and others topologies in comparison to a dragonfly network topology [7, 35]. While Booksim provides the support for a number of network topologies, the maximum size of the network is limited to only 1024 nodes.

Further, Wang et al. presented Orion [36], a power and performance interconnection simulator aimed at providing network designers with a framework for exploring interconnected microprocessor systems. It also provides fast design-level power estimation to enable research in power-efficient hardware. Orion uses a component-based approach by letting the users plug-in routers and links in order to investigate their impact on network performance. As a case study, Orion uses a 16-node 4x4 torus network and explores different router configurations such as wormhole and virtual channel routers under collective and uniform workloads.

Related parallel simulation results include Adiga et al. [3]. Here, the BG/L torus network was simulated by using a cycle-accurate simulator in which the simulation time was defined as the time it takes to transfer one byte. This simulator, driven by application pseudo-codes, runs on a 16-way shared memory machine. Each processor thread simulates a BG 512-node mid-plane and is synchronized using the YAWN (yet another windowing network simulator) protocol [37].

Additionally, Abhinav et al. [38] use a parallel simulation framework called BigSim, to study the performance of the PERCS network. This is a two-level direct network which connects the compute nodes into groups using high-bandwidth links at the lowest level and at the next level connects the groups using another type of link. They 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 carried out using these traces. The PERCS network topology is simulated for up to 307,200 cores at the packet-level detail.

Finally, the Structural Simulation Toolkit (SST) [29] uses a component-based parallel discrete-event model built on top of MPI. SST uses a conservative distance-based optimization without support for rollbacks. SST models a variety of hardware components including processors, memory, and networks under different accuracy and details. When the simulation starts, a system topology graph is loaded, which is then load-balanced across multiple processes. SST supports generic router models that uses wormhole routing for messages, though there is no flow-control in place and the router links have infinite buffer capacity. The network topologies currently supported are two- and three-dimensional meshes, binary tree, binary fat tree, hypercubes, flattened 2-D butterfly and a fully connected graph.

In summary, while a number of simulations accurately model the torus network topology, none of these simulations have been shown to scale to exascale network levels for a torus topology. We do note that the BigSim model of the IBM PERCS network has been shown to scale to modeling 300K cores, but the PERCS topology layout is much different from the torus network.

6. CONCLUSION AND FUTURE WORK

As we get closer to the exascale era, the search is under way for an interconnect topology that yields high bandwidth with relevant HPC traffic patterns. Simulation is an increasingly important tool for exploring the design space of massively parallel architectures yet most of the current simulation infrastructures have been shown to scale on a modest network size only. In this paper, we have applied massively parallel discrete-event simulation to efficiently model high-fidelity torus interconnects at a size of future exascale systems. We have used relevant HPC traffic patterns to explore the behavior of these network topologies under different configurations. We have shown that with the help of these network simulations, we can not only see realistic performance results, but we can instrument our ROSS network models to provide useful network performance statistics that help us gain an insight into exactly why we are seeing these performance results. We show that large-scale simulations are critical in the design of exascale systems, because trends at a modest scale are do not necessarily reflect network behavior
at exascale size. We have also demonstrated that we have the ability to simulate these large-scale network topologies in a reasonable amount of time which strongly suggests that massively parallel discrete-event simulation can be a key enabler for effective, efficient exascale-scale network codesign.

As part of future work, we plan to use the torus network model as the underlying interconnect in exascale storage architectures as part of the CODES simulation toolkit [39]. In terms of exploring other candidate interconnect topologies for exascale systems, we are also working to simulate a million node high-fidelity dragonfly network topology using ROSS [40]. We also plan to explore the behavior of the torus and dragonfly interconnects using real application network workloads in addition to the currently used, synthetic traffic patterns.

7. ACKNOWLEDGEMENT

The 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. This research used resources of Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

We gratefully acknowledge Ning Liu, Ph.D. student at the Illinois Institute of Technology, for his help with the torus network model development.

8. REFERENCES


