Model and Simulation of Exascale Communication Networks

Ning Liu[1], Christopher Carothers[1], Jason Cope[2], Philip Carns[2], Robert Ross[2]

[1]*Rensselaer Polytechnic Institute, Troy, NY, USA; and* [2]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA*

Exascale supercomputers will have millions or even hundreds of millions of processing cores and the potential for nearly billion-way parallelism. Exascale compute and data storage architectures will be critically dependent on the interconnection network. The most popular interconnection network for current and future supercomputer systems is the torus (e.g., *k*-ary, *n*-cube). This paper focuses on the modeling and simulation of ultra-large-scale torus networks using Rensselaer's Optimistic Simulator System (ROSS). We compare real communication delays between our model and the actual torus network from Blue Gene/L using 2,048 processors. Our performance experiments demonstrate the ability to simulate million-node to billion-node torus networks.[a] The torus network model for a 16-million-node configuration shows a high degree of strong scaling when going from 1,024 cores to 32,768 cores on Blue Gene/L, with a peak event-rate of nearly 5 billion events per second. We also demonstrate the performance of our torus network model configured with 1 billion nodes on both Blue Gene/L and Blue Gene/P systems. The observed best event rate at 128K cores is 12.36 billion per second on Blue Gene/P. processors.

Keywords: Parallel Discrete-Event Simulation; Torus Network; Exascale; Discrete-Event Model

[a]In this paper, we use "node" to denote a logical node in our simulation model. In contrast, "processor" and "cores" are used to denote the actual simulation platform except for the Blue Gene/L introduction subsection.

1 Introduction

We are moving closer to the era of exascale (i.e., 10^{18} FLOPs) supercomputing. As machines grow larger and larger, the internal communication between millions of cores is one of the key factors that will determine future computation and storage performance for massively parallel applications. Torus networks have been widely employed as the underlying network topology for many supercomputing systems, such as the Blue Gene [2] and Cray XT [6] families. A torus is chosen because it yields low latency for nearest neighbor processors and scalable bisection bandwidth and provides an easy physical wiring plan for upgrading a system with additional nodes without having to update the entire core torus network [2].

As part of the exascale co-design process, there is significant interest in understanding how parallel systems software like MPI/MPI-IO and the associated supercomputing applications will scale on future architectures. To this end, researchers have turned to massively parallel discrete-event simulation. For example, Perumalla's $\mu\pi$ system will allow MPI programs to be transparently executed on top of the MPI modeling layer and simulate the MPI messages. Here, each MPI task is realized as a thread in the underlying μsilk simulator. Thus, $\mu\pi$ captures the true direct execution behavior across millions of MPI tasks that are part of a whole supercomputing application. In particular, $\mu\pi$ has executed on 216,000 Cray XT5 cores an MPI job that contained over 27 million tasks. This is the largest direct execution simulation of any parallel program we are aware of to date. Similar systems, such as BigSim [29], have not achieved such a high level of scaling. To the best of our knowledge, however, neither of these systems performs packet-level simulations of the underlying network at scale. Instead the focus of their research is application computation performance with the network abstracted away.

The focus of our research is to create a "good"-fidelity, flexible, large-scale torus network model for understanding the performance implications of different exascale storage architectures that unlike today's current supercomputer systems, could be more closely woven into the compute-side architecture (i.e., bring storage closer to computation in order to improve performance and fault tolerance). Consequently, we need the ability to model an exascale compute network as part of our exascale storage co-design research. One use case scenario of the torus network model is determining the speed by which failure information is distributed throughout the storage system across different failure detection algorithms, such as a gossip algorithm [11]. In this case, we need a good model of network congestion, but a cycle-accurate network simulation is too costly to process at the network scale we are investigating, especially for longer running I/O workloads that span many massively parallel jobs and can last 12 to 24 hours, if not days [17].

The central contribution of this paper is that we present the capability to execute extremely large-scale torus network models at a "good" level of fidelity using both Blue Gene/L and Blue Gene/P supercomputers. We want to determine how big a torus network model can be executed using relatively modest supercomputing hardware (i.e., Top 500 rank as of June 2011 is less than 100 and dropping fast). We observe that the 70 teraFLOPs Blue Gene/L resource used here will be a very small slice of the upcoming "Mira" (nearly 800,000 cores) and "Sequoia" (1.6 million cores) Blue Gene/Q systems [9, 13], which will be available in late 2011/early 2012. These systems will provide a to-

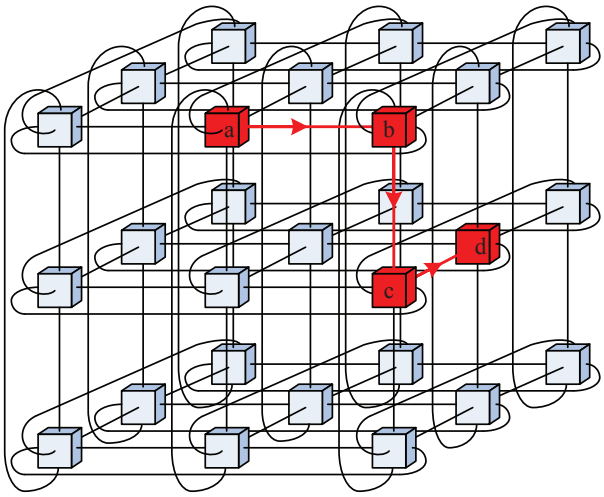

Figure 1: 3-ary 3-cube torus.

tal of 10 and 20 petaFLOPs of compute power, respectively, and more than 200 teraFLOPs of per rack. Thus, in the near future, parallel computations consisting of 32,000 MPI tasks will be commonplace.

The remainder of the paper is organized as follows: Section 2 presents the details of torus network developed for massively parallel execution. Section 3 introduces our parallel simulator and simulation platform. Performance experiment setup, results and analysis are presented in Section 4. Related work is discussed in Section 5 with closing remarks and a brief mention of future work in Section 6.

2 Torus Network Model

In this study, the simulation model consists of several models, including the network hardware, network routing model and network traffic model which drives the simulation.

2.1 Torus Structure

A torus network topology is a k-ary n-cube, where k is referred to as the radix and n as the dimension [3]. This grid structure has n dimensions, and each dimension has k nodes. A 3-ary 3-cube torus network is shown in Figure 1. Each of the $N(N = k^n)$ nodes can be identified by an n-digit radix k address $(a_1, a_2, \ldots, a_n)$. The ith digit of the address vector, a_i, represents the node position in the ith dimension. Nodes with address $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ are connected if and only if there exists i, $(1 \leq i \leq n)$, such that $a_i = (b_i \pm 1) \bmod k$ and $a_j = b_j$ for $a \leq j \leq n; i \neq j$.

The distance between two nodes in the torus is measured by the number of hops. A "hop" is the process of moving data from one node to its direct neighbor. Different from a k-ary n-cube mesh, whose maximum distance between any Two nodes is kn, the actual maximum distance in the torus is cut by half, $kn/2$. The maximum number of hops in a torus network determines the maximum latency of the communications.

Blue Gene and Cray XT supercomputer families adopt a 3-D torus. Typically, the 3-D torus network will fix the number of nodes in two dimensions so the system grows only in the 3rd dimension as racks of new systems are added. This configuration leads to a linear increase in the maximum latency. However, one solution is to create a torus network with a larger number of dimensions, as is the case with the upcoming Blue Gene/Q, which will have a 5-D torus network [9, 13].

2.2 Torus Routing

By design, a torus network will provide low latencies and high bandwidth at a moderate construction cost. We observe that a lot of research efforts have concentrated on the design and optimizations of switching fabric and routing algorithms [20, 1, 24, 16] for torus networks.

As shown in Figure 1, if node a needs to send a packet to node d, it has to choose among many possible paths. One of the possible paths is labeled in the graph. Routing in a torus can be carried out by either a specific routing algorithm, such as the deterministic e-cube algorithm, or a generic routing algorithm, such as the well-known up/down routing algorithm or the flexible routing algorithm [23]. In this paper, we use the static deterministic routing instead of dynamic routing or a mixture of static and dynamic which is used in Blue Gene system. Static, deterministic routing resembles e-cube routing in many ways. The well-known e-cube algorithm routes packets in decreasing dimension order to avoid deadlocks. When applied to a torus, the e-cube algorithm requires two virtual channels to remove any cyclic channel dependencies introduced by the wrap around links [23, 26]. Additionally, the use of bidirectional channels within a link between two nodes allows packets to be routed through minimal paths.

To illustrate how the e-cube algorithm works in a torus, we assume that each physical channel is split into two virtual lanes (VL0 and VL1). A packet arriving at a node n_c and destined to node n_d, with coordinates $n_{c_{n-1}}, n_{c_{n-2}}, \ldots, n_{c_1}, n_{c_0}$ and $n_{d_{n-1}}, n_{d_{n-2}}, \ldots, n_{d_1}, n_{d_0}$, respectively, will be routed through the physical channel belonging to the dimension i, where i is the position of the most significant digit in which addresses n_c and n_d differ. In each dimension i, packets will be routed through VL1 if the ith digit of the destination address is greater than the ith digit of the current node address. Otherwise, the packet will be routed through VL0.

2.3 Torus Traffic

Markovian models have been a popular approach to understanding interconnection network performance [3]. For the purpose of this performance study, we capture the time independent nature of the packet stream and simply let it follow the Poisson process with a mean arrival rate of λ. In our

model, each node continuously generates a Poisson stream of packets, and each generated packet randomly chooses a destination node, which follows a uniform distribution. This yields a pathological traffic pattern with little to no locality and consequently presents a model that is challenging to obtain good parallel performance.
According to Little's Law, if the average service time is fixed, the packet arrival rate determines the number of packets alive in the system at a steady state. For the torus network model considered here, each node has an infinite queue subject to the memory limitations of the parallel computer system executing the discrete-event model such that a high packet arrival rate can saturate the system.
Based on the above discussions, we have the following assumptions for our torus model:

1. Each node will participate in generating, processing, and relaying packets throughout the entire simulation with no failures.
2. The connection between any nodes in each direction is in good condition throughout the entire simulation with no failures.
3. Routing between nodes is static and deterministic. Note: our model is deadlock-free torus given the above two assumptions.
4. Bandwidth is constant on each connection between the nodes for a given message size.
5. Each node continuously generates a packet stream with an exponential interarrival time. The destination node is selected at random.

2.4 Model Configuration

In Figure 2, the torus network flow chart illustrates the event driven approach on a per node basis. At the initial state, each node, modeled as a logical process (LP), will schedule a `packet_generate_event` shown below in Algorithm 1. When processed, the `packet_generate_event` will then schedule the `packet_send_event` immediately and another `packet_generate_event` with an exponential time delay. This event generation delay forms the Poisson packets stream on each node. Prior to scheduling the `packet_send_event`, the destination node ID is placed into a packet header for routing at each hop within the torus network. Also, the packet generation time is stored in the header, enabling the model to capture the end-to-end latency. We note that the current model assumes that the "application-level" message is the same size as a packet. For real network workloads, such as the storage system, this would not be the case. Instead, a single application-level message could span multiple packets. This feature will be added at a later time when the torus model is integrated with other proposed exascale storage architecture model components. Because of the flexibility of our model, each node can generate different packet streams and destinations in the torus model. We observe that the torus model is a submodel that can be part of a greater application's communication model that is able to generate a traffic model in response to higher-level computation such as a file system in a proposed exascale storage architecture.

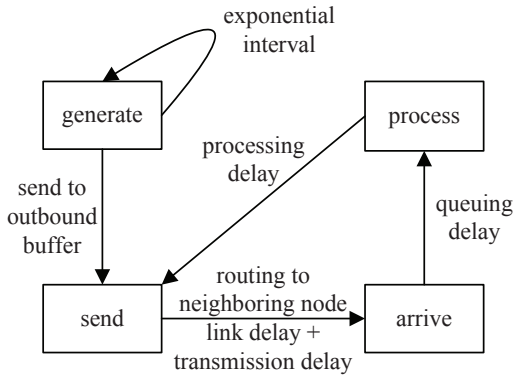

Figure 2: Discrete-event torus network model.

Algorithm 1 packet_generate_event.

1: generate a random time delay T_D (exponential distribution);
2: schedule next packet_generate_event after T_D on current node;
3: select random destination (uniform distribution) for current packet;
4: record packet destination in packet header;
5: record packet generating time in packet header;
6: schedule a packet_send_event on current node with delay T_D;

In the `packet_send_event` shown in Algorithm 2, the packet header is parsed, and the next-hop node/LP is chosen. In the torus network, the next-hop node is one of the direct neighbors among the various dimensions supported in the torus based on the previously described e-cube routing algorithm. Next, the `packet_arrive_event` is scheduled to the target neighboring node with a properly computed time delay. The time delay includes the link delay for virtual channel setup and transmission time for the packet based on the bandwidth capacity of the link. We note that the packet size on Blue Gene/L is usually a multiple of 32 bytes, with a maximum size of 256 bytes [7] including the header. In our model, the packet size is initially set as 256 bytes. Packet size and bandwidth of the link determine the transmission delay.

Algorithm 2 packet_send_event.

1: compute link delay and transmission delay T_D;
2: **if** (link available)
3: schedule a `packet_arrival_event` on next-hop node after T_D;
4: **else**
5: acquire link next_available_time;
6: schedule a `packet_arrival_event` on next-hop node at next_available_time;
7: update link next_available_time;

The `packet_arrive_event` shown in Algorithm 3 processes the arriving packet on the current node. Whenever a packet arrives at a node and cannot be immediately routed to the next hop node, it is placed in an incoming buffer along that dimension. In this model, each torus node has two buffers for each dimension. Here, one buffer is used for the $+$ direction and the other for the $-$ direction for each dimension of the torus. The queuing delay depends largely on the number of packets waiting in the buffer. Currently, the waiting queue simply follows a first-come-first-serve policy. The `packet_arrive_event` will then schedule a `packet_process_event` for the next available time on the current node.

Algorithm 3 packet_arrival_event.

1: **if** (processing unit available)
2: schedule a packet_process_event on the current node with delay T_D;
3: **else**
4: acquire processing unit next_available_time;
5: schedule a `packet_process_event` on current node at next_available_time;
6: update processing unit next_available_time;

To simplify the switching fabric, we use a constant processing time for each packet. This processing time corresponds to the time used for parsing the packet header and acquiring next-hop information. The `packet_process_event`, shown in Algorithm 4, invokes the routing routine and determines to which neighboring node the packet will be routed. The next-hop information is packed into the packet header so that `packet_send_event` can efficiently parse it. The hopping corresponds to an activation of `packet_send_event` with a processing time delay. If the destination node is the current node, the arriving packet will no longer activate events. Its life cycle has ended and we begin to collect the statistics such as the latency and trace. Packet processing event is illustrated in Algorithm 4.

Algorithm 4 packet_process_event.

1: **if** (packet arrived)
2: collect statistics;
3: **else**
4: analyze packet header and acquire destination node;
5: call routing routine and compute next hop node;
6: compute packet processing time T_P;
7: schedule a `packet_send_event` after T_P on current node;

3 Blue Gene, ROSS and Reverse Computation

Rensselaer's Optimistic Simulation System (ROSS) is geared for running large scale, parallel, discrete-event simulation models using Jefferson's Time Warp event scheduling mechanism [15]. Here, the parallel simulator consists of a collection of logical processes, each modeling a node in the torus network. LPs communicate by exchanging timestamped event messages. In this study, this event message is modeled as a packet. Like most existing parallel and distributed simulation protocols, we assume LPs may not share state variables that are modified during the simulation. The synchronization mechanism must ensure that each LP processes events in timestamp order to prevent events in the simulated future from affecting those in the past. The Time Warp [15] mechanism uses a detection-and-recovery protocol to synchronize the computation. For the recovery, we employ reverse computation, which is described below.

3.1 Blue Gene

The philosophy of the Blue Gene series is that more powerful processors are not the answer to winning the massively parallel scaling war. Instead, the Blue Gene architecture balances the computing power of the processor against the data delivery speed of the network. This philosophy led designers to create smaller, lower-power compute nodes comprising two 32-bit IBM PowerPCs running at only 700 MHz with a peak memory per node of 1 GB for Blue Gene/L. A Blue Gene/L rack is composed of 1,024 dual-processor "node" cards and is divided into 32 drawers of 32 nodes per drawer. Additionally, specialized I/O nodes perform all file I/O. Normally there is one I/O node for every 32 compute nodes. The Blue Gene/L system used in this performance study is a 16-rack, 32,768-processor system located at Rensselaer's Computational Center for Nanotechnology Innovations.

The IBM Blue Gene/P (BG/P) system is the second in a series of supercomputers designed by IBM to provide extreme-scale performance along with high reliability and best-inclass power consumption. Each rack of BG/P has 1,024 quad-core nodes, with a total of 2 terabytes of memory and a peak performance of 13.9 teraFLOPs. In this study, we also used Intrepid, the Argonne Leadership Computing Facility BG/P system. Intrepid has 40 such racks; in aggregate, the system has 80 terabytes of memory, a peak performance of 557.06 teraFLOPs, and 163,840 compute cores.

3.2 ROSS Implementation

The ROSS API [10, 5, 14, 27] is kept simple and lean. Developed in ANSI C, the API is based on a LP model. Services are provided to allocate and schedule messages between LPs. A random number generator library is provided based on a combined linear congruential generator (CLCG). Each LP by default is given a single seed set. All memory is directly managed by the simulation engine. Fossil collection is driven by the availability of free event memory, and its frequency is controlled with tuning parameters. The event-list priority queue can be configured to be either a Calendar Queue [8] or Splay Tree [25]. For this study, the Splay Tree is used in all performance experiments.

3.3 Reverse Computation

Reverse computation [10] is used to undo incorrect event computations. Shown in Figure 3 is the reverse handler code for all torus model event types.

As can be seen from the reverse code handler, the `packet_generate_event`, `packet_arrive_event`, and `packet_send_event` routines all have straightforward state reversal operations without any control flow changes. The `packet_process_event` needs to know whether this event reached its final destination node so the corresponding LP state variables and random numbers are correctly undone.

4 Simulation Results

The performance study examines the impact of processor/core count on four primary metrics: committed event-rate, percentage of remote events, efficiency and secondary rollbacks. Efficiency is defined as one minus the ratio of rolled back events divided by the number of total committed events. This is a more restrictive form of event efficiency used in [5] and a better predictor of event rate and overall speedup. These metrics, taken together provide a more comprehensive performance picture of how the torus network model is affecting overall parallel simulator performance. Specifically, we are using the committed event-rate and not the packet rate so we can observe how different torus configurations effect simulator performance since the packet rate can be affected by changes in the torus configuration.

Using these performance metrics, we focused the strong scaling study on a 16-million-node torus and then the 1-billion-node torus performance experiments. Our scaling study is based on the Blue Gene/L architecture using up to 32,768 cores; the validation study used an SMP system with 12 cores (Intel Xeon x5650, 2.67 GHz) to save supercomputer usage, where each core supports two threads of execution (24 threads total). We begin with our validation study.

```
void
rc_event_handler(nodes_state * s, tw_bf * bf,
                 nodes_message * msg, tw_lp * lp)
{
  int index =
    floor(N_COLLECT_POINTS*(tw_now(lp)/g_tw_ts_end));
  switch(msg->type)
    {
    case GENERATE:
      N_generated_storage[index]--;
      s->packet_counter--;
      tw_rand_reverse_unif(lp->rng);
      tw_rand_reverse_unif(lp->rng);
      break;
    case ARRIVAL:
      s->next_available_time =
         msg->saved_available_time;
      tw_rand_reverse_unif(lp->rng);
      msg->my_N_hop--;
      s->N_wait_to_be_processed--;
      msg->queueing_times -= s->N_wait_to_be_processed;
      s->node_queue_length[msg->source_direction]
                          [msg->source_dim]--;
      msg->my_N_queue -=
         s->node_queue_length[msg->source_direction]
                             [msg->source_dim];
      break;
    case SEND:
      s->source_dim = msg->saved_source_dim;
      s->direction = msg->saved_direction;
      s->next_link_available_time[s->direction]
                                 [s->source_dim]=
        msg->saved_link_available_time[s->direction]
                                       [s->source_dim];
      tw_rand_reverse_unif(lp->rng);
      break;
    case PROCESS:
     if ( bf->c3 == 0 )
        {
          N_finished--;
          N_finished_storage[index]--;
          total_time -= tw_now(lp) -
            msg->travel_start_time;
          total_hops -= msg->my_N_hop;
          total_queue_length -=
            msg->my_N_queue;
          queueing_times_sum -=
            msg->queueing_times;
        }
      s->node_queue_length
        [msg->source_direction]
        [msg->source_dim]++;
      s->N_wait_to_be_processed++;
      break;
    }
}
```

Figure 3: Reverse code event handler for torus metwork model.

4.1 Validation study

In this paper, there are two parts to the validation study. First, the torus network model agrees with Little's law under a variety of torus configurations and packets arrival rates. Second, we compared MPI_Send()/MPI_Recv() latency times of the actual Blue Gene/L network using 2K pro-

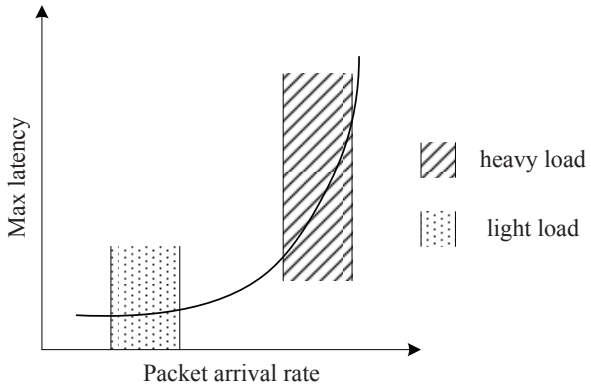

Figure 4: Latency prediction curve, with predicted max latency as a function of packet arrival rate.

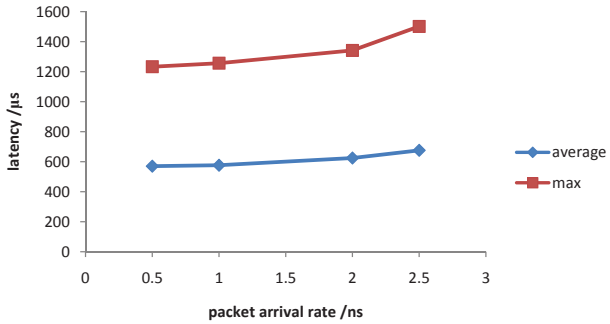

Figure 5: 512-ary 2-cube torus latency.

cessors (1,024 node torus, 1x32x32) and experimental runs of our torus network model for the same torus configuration. Starting from 256K nodes, we built three different torus networks 8-ary 6-cube, 64-ary 3-cube and 512-ary 2-cube for the Little law validation. The maximum distance decreases with an increase in the number of dimensions. The maximum distances for the above three network topologies are 24, 96, and 512, respectively. As we probe the saturation points of different torus network, the latency curves are shown in Figures 5, 6, and 7. The latency is measured in microseconds and the packet arrival rate is measured to that of per nanosecond. These curves have a similar shape as the predicted latency curve shown in Figure 4, indicating that the queuing behavior of the model is operating as expected.

For each of the torus configurations, we captured the average and maximum latency for all packets generated during the entire simulation. The total number of packets alive during steady state was also recorded. Under a small packet arrival rate, the system arrives at a steady state quickly. The torus network is far from saturation under this light load, as there is little packet queuing on each node. The latency does not increase while the packet arrival rate increases initially. Also, average latency is in proportion with the maximum distance in different torus configurations. As the packet arrival rate continues to increase, the total number of packets alive in the system increases as well. The queuing effect leads to a dramatic increase in the maximum latency. For each configuration, there is a saturation point for a certain packet arrival rate.

For the 16-million-node torus, we varied the configuration from an 8-ary 8-cube to a 16-ary 6-cube and finally to a 64-ary 4-cube. The Little's Law results are shown in Table 1. For the 1-billion-node torus the Little's law results are shown in Table 2, we varied the network configuration from 8-ary 10-cube to 32-ary 6-cube and finally 64-ary 5-cube. The experiment quickly reaches steady state with a light load of 0.5-2 pkt/ns. We note that for both scenarios the computed latencies via Little's law and the simulated latencies are in close agreement. Little's law latency is computed by using

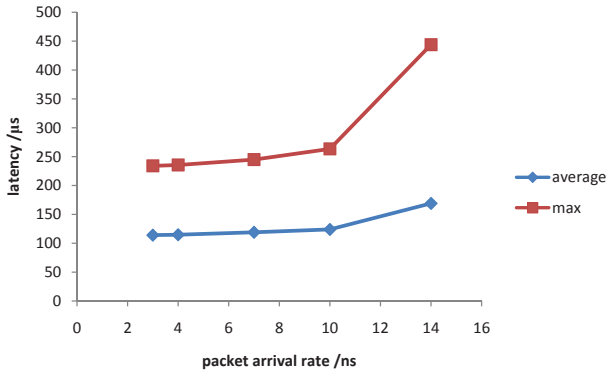

Figure 6: 64-ary 3-cube torus latency.

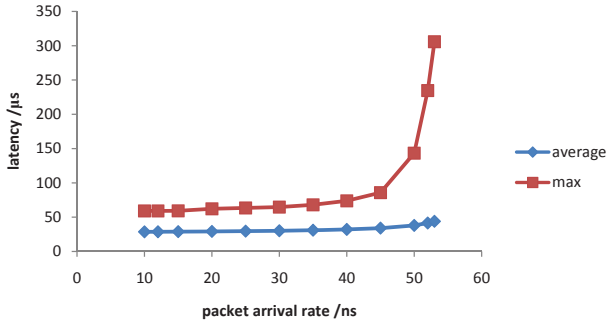

Figure 7: 8-ary 6-cube torus latency.

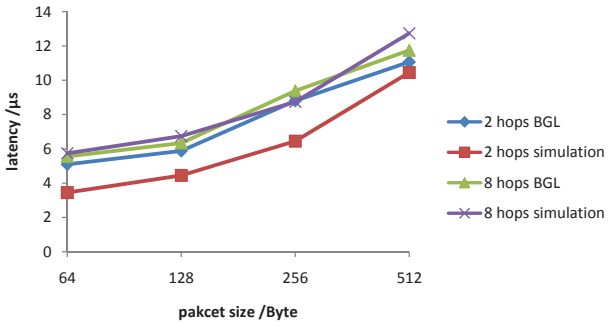

Figure 8: Simulated and observed torus network latency.

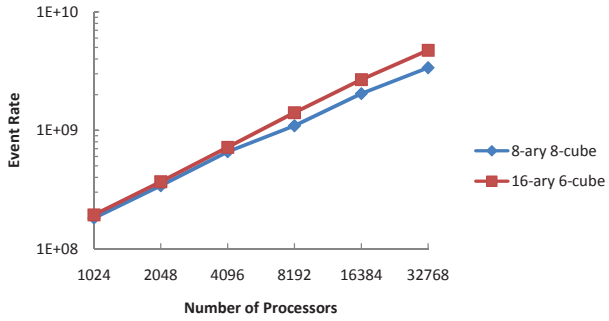

Figure 9: Event-Rate Scalability: The event-rate is shown as a function of the number of Blue Gene/L processors.

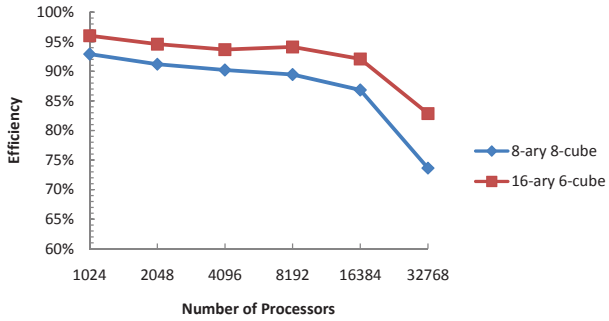

Figure 10: Event Efficiency: Model efficiency is shown as a function of the number of Blue Gene/L processors.

the number of packets waiting in the torus network and the packet arrival rate. These network models are operating in the light load zone of the latency curve shown in Figure 4.

To further validate the torus model, we conducted MPI_Send()/MPI_Recv() tests on Blue Gene/L and compared them to the simulation using our model as shown in Figure 8. Specifically, we measure communications times using an MPI "ping-pong" program and compare these latency measures with those predicted by our torus network model, since MPI_Send()/MPI_Recv() communication operations traverse over the Blue Gene's 3-D torus network. For this comparison, we configured the torus message size to be 250 bytes, which ensures use of the eager protocol being employed on the Blue Gene's real torus network and deterministically routes messages. We also varied the distances between the two communicating nodes in the torus from 2 to 8 hops. We observe the torus network model delays are matched to that of the Blue Gene/L machine with the 2-hop case and yielding more error than the 8-hop case. We believe the difference lies in some routing optimizations being made by the Blue Gene network enabling it to exceed the network performance of what our model predicts. Further investigation is required to more fully understand this phenomenon. We note that because the model does perform congestion queuing correctly, we believe it will be sufficient for predicting storage architecture performance. For example, the time to commit data to RAM disk, SSD, or disk ranges from hundreds of microseconds to milliseconds vs the torus network model error of 2 microseconds for a 2-hop torus network message. So, the current network model latency error is negligible for the exascale storage modeling purpose.

4.2 Million-Node Torus Scaling Study

For the scaling study, we configured two torus models with 16 million nodes. The first torus was configured as an 8-ary 8-cube and the second was a 16-ary 6-cube. The packet arrival rate on each node was 10 pkt/ms. To reach a steady state, we set the simulation time to 100 ms. In fact, we observed the system reaching a steady state in less than 10 ms. On Blue Gene/L, the strong scaling experiments start by using 1,024 cores and run up to 32K cores. The committed event-rate is shown in Figure 9. We observe a near-linear speedup using 1,024 cores as the base case, with a peak event-rate of 4.78 G/s on 32K cores. The torus model creates a difficult scenario for parallel event scheduling because each packet randomly selects its destination, leading to a high number of remote/off-processor scheduled events. As shown in Figures 10, 11, and 12, with an increase in the number of processors, the event efficiency decreases, and the remote event-rates and secondary rollback event rates increase. At 32K cores, the event efficiency actually drops below 90% for both configurations. We also observe that the efficiency drops significantly as we decrease the system packet arrival rate. This phenomenon is attributed to a reduction in the available work per unit of simulation time, which increases the likelihood of incurring a rollback.

The best event efficiency is achieved at near the saturation point of the torus network model. We speculate on the maximum latency curve shown in Figure 4 based on our experimental findings. The shaded area corresponds to the near saturation of the system or heavy load, while the dotted area represents a light load. For the two torus configurations, the 8-ary 8-cube has larger bisection bandwidth and therefore smaller latency. The 16-ary 6-cube torus tends to be more saturated under the same packet arrival rate and yields a higher event efficiency and committed event-rate.

Table 1: 16-million-node torus: simulation VS. theory.

Torus Configuration	No.Packets Waiting	Packet Arrival Rate	Simulated Latency	Computed Latency
8^8	168,201	4	42,022	40,700
	210,240	5	42,043	42,048
16^6	2,531,687	40	63,260	63,292
	3,166,931	50	63,306	63,339
64^4	5,095,405	30	169,908	169,847
	6,805,740	40	169,571	170,144

Table 2: 1-billion-node torus: simulation VS. theory.

torus configuration	No.Packets Waiting	Packet Arrival Rate	Simulated Latency	Computed Latency
8^{10}	105,505	2	52,688	52,753
32^6	64,109	0.5	126,710	128,218
64^5	107,135	0.5	209,686	2,142,709

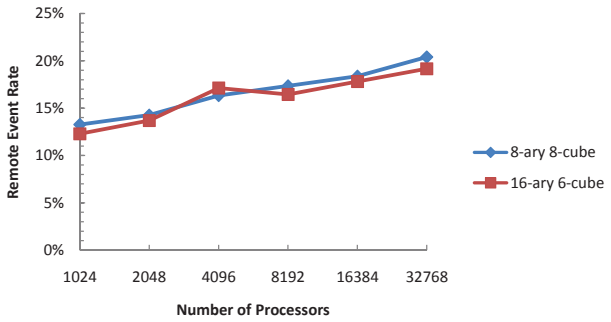

Figure 11: Remote events: The percentage of remote/off-processor events is shown as a function of the number of Blue Gene/L processors.

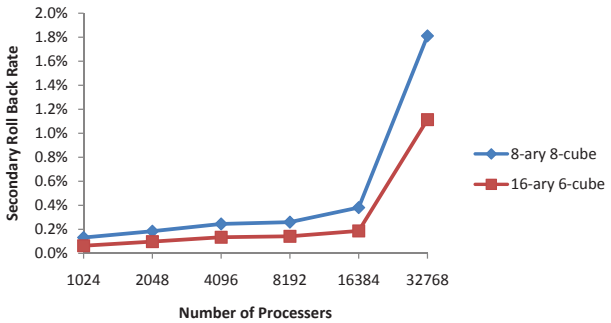

Figure 12: Secondary rollback rate: The percentage of secondary rollbacks is shown as a function of the number of Blue Gene/L processors.

Table 3: Strong scaling performance of 1-billion-node model at configuration 32^6 w ith packet arrival rate of $200\,pkt/ns$ on Blue Gene/L.

	Number of Processors		
	4,096	8,192	16,384
Number of packets (G)	40	40	40
Efficiency	97.05%	96.00%	81.90%
Event-rate (M/s)	639	1,066	1,681
Remote event percentage	11.72%	12.41%	13.79%
Secondary rollback rate	0.0286%	0.0347%	0.220%
Number of event (G)	5,644	5,644	5,644

Table 4: Strong scaling performance of 1-billion-node model at configuration 32^6 with packet arrival rate of $400\,pkt/ns$ on Blue Gene/L.

	Number of Processors		
	4,096	8,192	16,384
Number of packets (G)	80	80	80
Efficiency	97.33%	96.81%	96.42%
Event-rate (M/s)	638	1,241	1,966
Remote event percentage	11.72%	12.41%	13.79%
Secondary rollback rate	0.0268%	0.0312%	0.0245%
Number of event (G)	11,442	11,442	11,442

4.3 Billion-Node Torus Network Scaling Study

As our model grows to 1 billion nodes, the simulation becomes more memory demanding. Consequently, the required base case (i.e., minimum number of processors that are able to execute this model) begins at 4,096 processors on Blue Gene/L with a total memory of 2TB and goes to 8,192 and 16,384 processors. The results are shown in Tables 3 and 4. On Intrepid, our scaling study runs to 131,072 cores and the performance results are shown in Table 5.

In these studies, the total number of generated packets

is $O(10^{11})$, and the total number of events scheduled is $O(10^{13})$. This extreme-scale torus model can sustain a continuous packet stream of 10^{11} pkt/s. Compared with our 16-million-node torus model experiments, the efficiency appears lower at 16,384 processors for the 200 pkt/ns scenario. This phenomenon is attributed to the 1-billion-node torus model being underloaded with packets relative to the 16-million-node torus model. Thus, in the absence of queuing effects, events are scheduled more closely together in simulated time in the 1-billion-node torus model, leading to a higher rollback probability. The overall loss in event-rate performance is attributed to the larger event population leading to increases in queuing overheads as well as larger cache-memory overheads because of the increased amount of RAM required to execute the 1-billion-node torus model. The strong scaling study on Blue Gene/P systems has shown better performance compared to the results on Blue Gene/L. We observed a sublinear strong-scaling performance from 4,096 cores to 131,072 cores. The peak event rate achieved is 12.359 billion per second, which is the best performance ever reported from ROSS. Note that the Blue Gene/P node is quad-core instead of dual-core on Blue Gene/L and thus we attribute the increased performance to better locality. One obvious observation is that the secondary rollback rate is much lower on Blue Gene/P than on Blue Gene/L.

5 Related Work

A large body of previous research focused on the modeling and simulation of torus interconnection networks. Min and Ould-Khaoua [19] proposed a torus network model based on circuit switching. In this paper, the traffic generated on each node followed an independent stochastic process with a given arrival rate. We followed a similar traffic pattern, but our model was based on packet switching. In circuit switching, a handshaking protocol between the source and destination node was used to initiate the communication operation. Instead of establishing a virtual channel first between source and destination, we considered incoming and outgoing buffers on each node. Min and Ould-Khaoua also investigated long-range dependence and traffic correlation and are able to validate their model on a small-scale torus network.
Sancho et al. [23] focused on a comparison of routing algorithms over torus networks. Their InfiniBand network model was composed of a set of switches and hosts interconnected by a single link. The model provided the e-cube, up/down, and flexible torus routing algorithms. The authors demonstrated that the flexible routing algorithm was the most effective on small-scale 3D and 2D torus networks. In our model, we showed the effectiveness of static deterministic routing algorithm up to 10 dimensions and extremely large-scale torus network. In Sancho et al's work, the traffic model was simplified to switches. The packet latency was determined by changes in the switch traffic.

In the context of network simulation for supercomputer systems, Adiga et al's use of the YAWNS protocol [2] to model the Blue Gene/L torus network on a per cycle basis appeared to be one of the most accurate models created to date. This work focused on the design and implementation of the 3D Blue Gene/L torus network. Details of the physical system included variable packet size, packet header size, different virtual channels and usage, per link bandwidth, and routing strategies. In contrast, we assumed a fixed packet size, and, for simplicity, adopt static routing instead of a mixed static/dynamic routing in our simulation. In their simulation, hot region traffic and all-to-all traffic were studied on torus networks with 4K nodes and 32K nodes. We noted that our goals and intended use of this model were to understand the trade-offs associated with different exascale storage architectures and filesystems capabilities. To this end, we believed a packet-level network model such as the one discussed here was sufficient.

Guirguis et al. [12] examined dynamic routing over content-addressable networks and demonstrate that routing could be improved while recording fewer states of the neighbors. A detailed analysis of the congestion behavior on Blue Gene/P interconnection networks was provided by Balaji et al. [4]. Their experiments were executed on over 128K cores, providing insight into the network performance characteristics of MPI.

Rahman et al. [21, 18] examined the performance of the Hierarchical Torus Network (HTN) under the traffic pattern generated by the matrix transpose computation. They designed a deadlock-free routing algorithm for the HTN using virtual channels and evaluated the performance of the HTN incontrast to the performance of conventional mesh and other networks. There research results demonstrated the high throughput and low-latency capabilities of HTN over other networks.

Safaei et al. [22] presented a new mathematical model to

Table 5: Strong scaling performance of 1-billion-node model at configuration 32^6 with packet arrival rate of $160 pkt/ns$ on Blue Gene/P.

	Number of Processors					
	4,096	8,192	16,384	32,768	65,536	131,072
Efficiency	99.83%	99.90%	99.83%	99.55%	98.89%	97.51%
Event-rate (M/s)	609	1,192	2,260	4,002	7,307	12,359
Remote event percentage	11.71%	12.39%	13.77%	16.53%	16.88%	17.22 %
Secondary rollback rate ($\times 10^{-5}$)	1.06	0.254	0.0429	0.51	3.87	21.7

capture the mean message latency in the torus interconnect network with circuit switching in the presence of a hot-spot. Simulation experiments demonstrated close agreement between the observed network behavior and those obtained by the analytical model.

Additional recent large-scale, parallel, discrete-event performance studies included the work of Bauer et al. [5], who conducted an extensive study of the PHOLD model and an electromagnetic wave propagation model. On Blue Gene/P, they reported a peak event-rate of 12.3 G/s using 65,536 cores. They observed near-linear scaling on Blue Gene/L from 1,024 cores to 32K cores. Yaun et al. [27, 28] demonstrated that optimistic protocols were able to efficiently simulate large-scale TCP scenarios for realistic, network topologies using an inexpensive hyper-threaded computing system.

6 Conclusion

This paper focuses on the simulation of large-scale torus networks. Based on the Rensselaer's Optimistic Simulation System (ROSS), we are able to achieve a near-linear speedup for our torus model. On Blue Gene/L, the peak event-rate on 32K cores is 4.78 G/s. On Blue Gene/P, the best event-rate observed is 12.359G/s on 128K cores. We further demonstrated the ability to model and simulate a million-node and a billion-node torus network on both Blue Gene/L and Blue Gene/P platforms. The experiment results are validated by Little's law for different torus configurations under varying packet arrival rate. We also conducted comparison tests between actual Blue Gene torus network and our model using MPI_Send()/MPI_Recv(). The latencies captured are comparable. We believe the difference is due to subtle complexities in the Blue Gene's packet routing that are not accounted for in our model. We plan to work on improving the accuracy of the model, for example, by introducing different torus routing algorithms and modeling the details of virtual channels.

References

[1] H. Abu-Libdeh, P. Costa, and A. Rowstron. Symbiotic routing in future data centers. In *ACM Conference on Special Interest Group on Data Communication (SIGCOMM'10)*, pages 51–62, New Delhi, India, August 2010.

[2] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue Gene/L torus interconnection network. *IBM J. RES. & DEV.*, 49(2/3):265–276, 2005.

[3] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, 1991.

[4] P. Balaji, H. Naik, and N. Desai. Understanding network saturation behavior on Large-Scale Blue Gene/P systems. In *The Fifteenth International Conference on Parallel and Distributed Systems (ICPADS'09)*, Shenzhen, China, 2009.

[5] D. W. Bauer, C. D. Carothers, and A. Holder. Scalable time warp on Blue Gene supercomputers. In *Proc. ACM/IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS'09)*, Lake Placid, New York, 2009.

[6] A. S. Bland, R. A. Kendall, D. B. Kothe, J. H. Rogers, and G. M. Shipman. Jaguar: The world's most powerful computer. In *Compute the Future, CUG 2009 Proceedings*, Atlanta, Geogia, May 2009.

[7] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burow, T. Takken, and P. Vrannas. Design and analysis of the BlueGene/L torus interconnection network. Technical Report RC23025(W0312-022), IBM Thomas J. Watson Research Center, New York, December 2003.

[8] R. Brown. Calendar queues: A fast 0(1) priority queue implementation for the simulation event set problem. *Commun. ACM*, 31:1220–1227, October 1988.

[9] T. Budnik, B. Knudson, M. Megerian, S. Miller, M. Mundy, and W. Stockdell. Blue Gene/Q resource management architecture. In *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS10), co-located with IEEE/ACM Supercomputing*, volume 2010, 2010.

[10] 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, 1999.

[11] A. Das, I. Gupta, and A. Motivala. Swim: Scalable weaklyconsistent infection-style process group membership protocol. In *Proc. Intnl. Conf. Dependable Systems and Networks (DNS)*, pages 303–312, 2002.

[12] M. Guirguis, A. Bestavros, and I. Matta. Routing tradeoffs inside d-dimensional torus with applicability to CAN. In *Proc. of the First International Computer Engineering Conference (ICENCO2004)*, Cairo, Egypt, December 2004.

[13] P. Heidelberger. The IBM Blue Gene/Q interconnection network and message unit. In *19th Symposium on High-Performance Interconnects*, volume 19, 2011.

[14] A. Holder and C. D. Carothers. Analysis of time warp on a 32,768 processor IBM Blue Gene/L supercomputer. In *2008 Proceedings European Modeling and Simulation Symposium (EMSS)*, 2008.

[15] D. R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, 1985.

[16] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta. Microarchitecture of a high-radix router. In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)*, Madison, Wisconsin, June 2005.

[17] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 213–226, Berkeley, CA, USA, 2008. USENIX Association.

[18] M. Ghoshtt M. M. Hafizur Rahman and S. Horiguchi. Inter-processor communication performance of a hierarchical torus network under bit-flip traffic patterns. In *International Conference on Electrical and Computer Engineering (ICECE'06)*, Dhaka, Bangladesh, December 2006.

[19] G. Min and M. Ould-Khaoua. Prediction of communication delay in torus networks under multiple time-scale correlated traffic. *Performance Evaluation*, 60:255–273, 2005.

[20] G. Mora, J. Flich, J. Duato, P. Lopez, and E. Baydal. Towards and efficient switch architecture for high-radix switches. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'06)*, pages 11–20, San Jose, California, USA, December 2006.

[21] M. M. Hafizur Rahman and S. Horiguchi. High performance hierarchical torus network under matrix transpose traffic patterns. In *Proc. 7th International Symposium on Parallel Architectures, Algorithms and Netowrks (ISPAN'04)*, Hong Kong, China, May 2004.

[22] F. Safaei, A. Khonsari, M. Fathy, and M. Ould-Khaoua. Analysis of circuit switching for the torus interconnect networks with hot-spot traffic. In *Proc. 2006 International Conference on Processing Workshops (ICPPW'06)*, Columbus, Ohio, August 2006.

[23] J. C. Sancho, A. Robles, P. Lopez, J. Flich, and J. Duato. Routing in infiniband(tm) torus network topologies. In *Proc. IEEE International Conference on Parallel Processing (ICPP'03)*, Valencia, Spain, 2003.

[24] J. Shalf, S. Kamil, L. Oliker, and D. Skinner. Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect. In *Proceedings of the 2005 ACM/IEEE Super Computing (SC'05)*, Seattle, Washington, November 2005.

[25] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32:652–686, July 1985.

[26] G. M. Thorson and S. L. Scott. Adaptive routing mechanism for torus interconnection network, December 1997.

[27] G. Yaun, C. D. Carothers, and S. Kalyanaraman. Large-scale tcp models using optimistic parallel simulation. In *Proc. 7th workshop on Parallel and Distributed Simulation (PADS'03)*, San Diego, California, June 2003.

[28] G. R. Yaun, D. W. Bauer, H. L. Bhutada, C. D. Carothers, M. Yuksel, and S. Kalyanaraman. Large-scale network simulation techniques: Examples of TCP and OSPF models. *SIGCOMM Computer Comunications Review Special Issue on Tools and Technologies for Research and Eduction*, 33(5), 2004.

[29] G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale. Simulating Large Scale Parallel Applications using Statistical Models for Sequential Execution Blocks. In *Proc. 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, number 10-15, Shanghai, China, December 2010.

Acknowledgments

This work was supported in part by the Office of Advanced Scientific Computer Research, Office of Science, U.S. Dept. of Energy, under Contracts DE-AC02-06CH11357 and DE-FC02-10ER25989/DE-SC0004875, and in part by the NSF CNS NeTS Program, Contract #0435259. This research used resources of the 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. Computing time on Intrepid was provided by a U.S. Department of Energy INCITE award. Rensselaer's Computational Center for Nanotechnology Innovations (CCNI) provided the Blue Gene/L computing resources.