*Proceedings of the 2018 Winter Simulation Conference*
*M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.*

# LEVERAGING SHARED MEMORY IN THE ROSS TIME WARP SIMULATOR FOR COMPLEX NETWORK SIMULATIONS

Caitlin J. Ross
Christopher D. Carothers

Computer Science Department
Rensselaer Polytechnic Institute
110 8th Street
Troy NY 12180, USA

Misbah Mubarak
Robert B. Ross

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue
Lemont, IL 60439, USA

Jianping Kelvin Li
Kwan-Liu Ma

Computer Science Department
University of California, Davis
1 Shields Avenue
Davis, CA 95616, USA

## ABSTRACT

Scalability of parallel discrete-event simulation (PDES) systems is key to their use in modeling complex networks at high fidelity. In particular, intranode scalability is important due to the prevalence of many-core systems, but MPI communication between cores on the same node is known to have drawbacks (e.g., software overheads). We have extended the ROSS optimistic PDES framework to create memory pools shared by MPI processes on the same node in order to reduce on-node MPI overhead. We perform experiments to compare the performance of shared memory ROSS with pure MPI ROSS on two different systems. For the experiments, we use several models that exhibit a variety of characteristics to understand the conditions where shared memory can benefit the simulation. In general, higher remote event rates means that simulations are more likely to benefit from using shared memory, but this may also be due in part to improved rollback behavior.

## 1 INTRODUCTION

Parallel discrete-event simulation (PDES) can be a productive tool in the codesign of high performance computing (HPC) systems and networks. For instance, IBM used PDES to study a variety of design tradeoffs in the Blue Gene/L system (Adiga et al. 2005), as well as to evaluate newly proposed networks to determine cheaper, yet effective, alternatives to expensive fat-tree networks (Chen et al. 2016). PDES has also been shown to be a highly scalable method of simulation, which has been seen, for example, in prior performance studies of Rensselaer's Optimistic Simulation System (ROSS). ROSS has been shown to scale up to 1.9 million cores when running PHOLD on the Sequoia system at Lawrence Livermore National Laboratory (Barnes et al. 2013). However, this level of scalability may not be experienced when using PDES to simulate more complex, heterogeneous models, which drives the work in improving the scalability of simulators such as ROSS.

In PDES, entities are modeled as logical processes (LPs) that interact through the exchange of timestamped messages. For parallel processing, LPs are mapped to processing elements (PEs), and synchronization is performed to ensure causal correctness of the simulation. Two main classes of synchronization protocols in PDES exist: conservative and optimistic. Conservative PDES ensures causal correctness by processing events only when it is safe to do so. Optimistic PDES protocols allow events to be processed speculatively and ensure causal correctness by providing an out-of-order event detection and recovery mechanism (Jefferson 1985; Fujimoto 1990).

ROSS is an open source PDES framework that provides conservative and optimistic parallel exectution modes. ROSS is built around the use of MPI for communication between all processing elements, making each PE an MPI rank. In modern multicore systems, using MPI communication between cores located on the same node is known to have drawbacks, such as software overheads and additional data copies (Gropp and Thakur 2006). As the number of cores on compute nodes increases, this on-node MPI communication overhead becomes worse. A common approach to reducing MPI overheads for intranode communication has been to combine the use of MPI and a multithreaded programming paradigm, such as pthreads. In PDES applications, this approach would make each PE a thread that shares a common memory space with other PEs/threads on that node. However, mixing pthreads and MPI requires that either the developer ensures that only one thread per node makes MPI calls or `MPI_THREAD_MULTIPLE` mode is used to allow pthreads to call MPI operations in a thread-safe way. The former option can cause performance bottlenecks, while the latter causes increased overhead.

In this work, we extend ROSS to use shared memory for communication between MPI ranks located on the same node. Because of the issues combining the use of MPI and pthreads discussed previously, we keep PEs realized as MPI ranks, but use shared memory between ranks on a given node to reduce on-node MPI overhead. In Section 3, we provide the implementation of shared memory to be used by MPI processes in ROSS, along with a discussion of the design tradeoffs for other approaches to improving the intranode exchange of data. In addition, we compare the performance of Shared Memory ROSS (SHMEM-ROSS) with the previous MPI only version of ROSS (MPI-ROSS), using a variety of models including PHOLD, a neuromorphic processor model, and HPC interconnect models. In particular, we examine the performance of the two ROSS versions on the Theta Cray XC supercomputer at Argonne National Laboratory (ANL), since each node contains a 64-core Intel Xeon Phi processor, as well as a Linux cluster with 16-core nodes. Further details of our experimental setup and performance evaluation are provided in Sections 4 and 5.

## 2 BACKGROUND AND RELATED WORK

A number of PDES frameworks have been developed over the years that support conservative and/or optimistic synchronization protocols, such as Georgia Tech Time Warp (GTW), Warped2, and ns-3. GTW was developed for optimistic simulation of models with relatively fine event granularity on cache-coherent, shared memory multiprocessors (Das et al. 1994). Warped2 is also an optimistic PDES framework and supports the use of both pthreads and MPI (Martin et al. 2003). Another popular discrete event simulator for modeling computer networks is ns-3; it supports sequential and conservative parallel simulation, with MPI being used for parallel simulation (Pelkey and Riley 2011).

Similarly to the work presented in this paper, other Time Warp simulators have been used in the exploration of intranode optimizations. For instance, Pellegrini and Quaglia (2015) have explored optimizations for Time Warp systems running on non-uniform memory access (NUMA) multicore systems. They developed a NUMA-aware memory management architecture for multithreaded Time Warp systems. Their implementation is application-transparent and they test it with the ROOT-Sim optimistic simulator. However, their performance studies are limited to single node systems.

### 2.1 ROSS

ROSS is an open source Time Warp simulator that implements the rollback mechanism using reverse computation (Carothers et al. 2002). In addition to the LPs and PEs discussed in Section 1, ROSS has entities, called kernel processes (KPs), that are responsible for a subset of LPs colocated on the same PE and were introduced to reduce fossil collection overheads. However, this results in rollbacks occurring on a KP basis instead of LP basis, meaning that some LPs may be rolled back unnecessarily. The number of KPs to use in a simulation is a performance tradeoff, between fossil collection overhead and false rollback overhead. For further details on the ROSS implementation, we refer the reader to the work of Carothers et al. (2002) and Bauer et al. (2009).

Some previous work has looked into converting ROSS from being MPI-based to being multithreaded, in order to take advantage of multicore systems. For example, Jagtap et al. implemented ROSS-MT using threads instead of MPI processes. They evaluated the performance of ROSS-MT against MPI-ROSS running the PHOLD benchmark on a 4-core Intel Core i7 processor, a 48-core AMD Opteron 6100 machine, and a 64-core Tilera processor (Jagtap et al. 2012a; Jagtap et al. 2012b; Wang et al. 2014). ROSS-MT experienced a speedup up to 3x on the Core i7, a 1.2x speedup on the 48-core machine, and up to 2.8x speedup on the Tilera chip. Since ROSS-MT has no support for a multi-node system (i.e., it does not use MPI to scale beyond a single node), it was then extended to ROSS-CMT, which allows the threads on a node to use MPI for communication with other nodes (Wang et al. 2012; Wang et al. 2013). In their performance studies, they found that ROSS-CMT outperformed MPI-ROSS by a factor of 4.5x, however, they allow only one thread on a node to perform MPI communication, which can potentially cause performance bottlenecks at scale. In contrast, in our implementation MPI processes on a node share event memory pools for communication with other ranks on the same node, and each process still has the ability to use MPI for internode communication. Our implementation and design choices are discussed in further detail in Section 3. To the best of our knowledge, ROSS is the only PDES framework that has explored the use of MPI and shared memory without the use of pthreads.

### 2.2 CODES

CODES is a simulation framework built on top of ROSS, leveraging its event scheduling capability to provide high-fidelity simulation of a variety of HPC network and storage models (Mubarak et al. 2017b). In the CODES HPC interconnect models, each router or switch is represented by a single LP. Each end point is represented by at least two LPs, dependent on the type of workload being simulated. All end points have one LP that handles the packet send and receive functionality and at least one LP that represents an MPI rank that generates traffic for the modeled workload.

## 3 SHARED MEMORY ROSS

The design of our shared memory pool structure for a single compute node is shown in Figure 1. This framework will enable the fast sharing of timestamped event data via shared memory pools including direct cancellation of incorrectly scheduled events across MPI ranks that reside on the same compute node. This performance improving functionality will be supported without having to change any of the LP mappings or the overall flow of how messages are processed except at the lowest event-passing levels.

### 3.1 Pthreads vs. MPI Ranks

Central to this design is preservation of the original data structure and functionality of the processing element (PE), which is realized as an MPI rank. A PE contains all the necessary data structures for a serial simulator plus additional functionality to realize a variety of optimistic and conservative event schedulers. Specifically, we did not want to introduce a mix of pthreads and MPI into this design because of the functional complexity of keeping pthreads from having to access MPI functionality or the increased

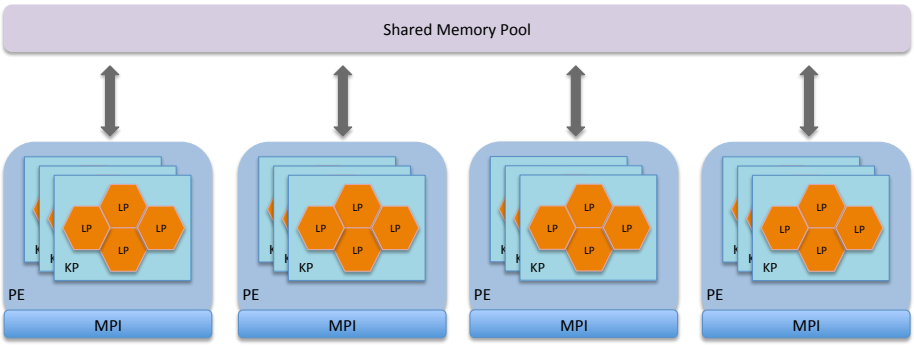



Figure 1: Diagram of ROSS entities using shared memory on a node.

overhead of the alternative `MPI_THREAD_MULTIPLE` mode, which allows pthreads to participate in MPI operations in a thread-safe way (Gropp and Thakur 2006). Another observation that led to this design choice is that pthreads and MPI ranks are both realized as full blown processes in all Linux OS distributions. The only difference is that pthreads provide a globally shared memory view by sharing the same set of virtual page tables within the Linux process. This can be seen in the Linux kernel because the `clone()` system call used by pthreads maps to the same system call handler function `do_fork()` as the `fork()` system call. The key difference between these calls is additional flags that enable mapping to the same virtual memory address space (e.g., `CLONE_VM`).

Inside each PE (i.e., MPI rank) are kernel processes (KPs) which aggregate a number of LPs together to reduce fossil collection overheads. LPs are the containers of model state and communicate exclusively by exchanging timestamped event messages. Direct sharing of model data across LPs is strictly forbidden in the current implementation of ROSS, since that would allow LPs to access incorrect versions of state in virtual time, leading to incorrect simulation results. For that, shared-state functionality, such as Space-Time Memory (Ghosh and Fujimoto 1991; Mehl and Hammes 1993), would need to be implemented.

### 3.2 Shared Memory Segment Setup

Prior to setting up the shared memory segment, each MPI rank attempts to perform a *bind* operation to a specific core on the compute node. This is done to prevent MPI ranks from migrating between cores which impacts the efficient sharing of data. In particular, we want to colocate the shared memory segment onto banks of physical memory closest to the cores that will be using that memory. Setting the core/CPU affinity helps to accomplish this task. Additionally, new MPI sub-communicators are established to enable multiple shared memory pools to be allocated within different sub-groups of MPI ranks all executing on the same node. For example, this capability supports configurations such as 8 shared pools with 8 MPI ranks each on a single shared memory node with 64 cores.

### 3.3 Shared Memory Segment Allocation

Now, to implement a shared memory pool, we allocate a shared memory segment using the `shmget` system call to which all MPI ranks executing within a compute node can be attached. However, an interesting challenge occurs when using the shared memory segment. By default, the virtual memory address where each MPI rank attaches could be different. Thus, the shared memory addresses for removing events would differ depending on which MPI rank was performing the processing. For our purposes, this is an unwanted feature of Linux shared memory segments.

To overcome this issue, we leverage the `mmap` system call to poll a virtual memory address to determine whether that block is available by mapping a "dummy" test memory block. If the `mmap` call succeeds for all the MPI ranks within a particular shared memory group communicator, then we have all MPI

ranks within the group attempt to perform an `shmat()` system call which will attach the shared memory segment at the desired virtual memory address. Should the `mmap` system call fail, we subtract the size of the desired shared memory segment again from the previously tested virtual address and try again. We observe that on most Linux systems, fewer than 100 attempts are required which execute in less than 20 seconds at simulation start up. In Linux, we start at address `0x00007fffff000000` and subtract the size of desired shared memory segment from that address which becomes the address of attachment. On the Theta supercomputer system, we find the typical attached address is `0x7ffeadfe7000` for a shared event pool size of 700 MB.

### 3.4 Sharing Remote Events and Direct Event Cancellation

Once created, the shared memory event pool is divided in equally large chucks across the MPI ranks that are attached to this segment. Then within each subpool, a free list of shared memory events is created where each event's `shared` flag is marked true. This flag will be used to determine which free list (local or shared) to place an event back into during fossil collection.

Additionally, three pthread locks for queue lists are held in each MPI rank's shared memory address space. The first list is the `event_q`, which will hold forward/positive timestamped event messages sent between LPs that are mapped to this group of MPI ranks. Next, the `cancel_q` will hold anti-message events sent between MPI ranks within this sharing group. The last list is the `return_q` and is used to hold a list of remote shared events that originated from this MPI rank's shared memory segment. Pthread locks on all of these list heads are used to ensure atomic access to these list data structures.

During regular event processing, if a new event is to be scheduled between LPs that are part of the same shared memory group, an event from the shared free list is allocated. After the event message data is populated in the model event handler and the event is "sent" into the ROSS engine for delivery, ROSS first determines the shared status of the event by checking its `shared` flag and then determines which MPI rank's shared memory `event_q` is its destination. Once the right pool area is determined, the correct `event_q` is locked and the event is threaded into the event list. A pointer to any sent event is kept in the parent event's cancellation list, which can be used during rollback processing. Direct event cancellation will be described below in more detail.

To receive shared events, each MPI ranks polls its own local `event_q` list. This is done prior to any MPI message polling for events that have been scheduled for LPs that are not on the same compute node or within the same shared memory segment group. If there are events, the lock is obtained on the `event_q`, and the whole list of events is inserted into the MPI rank's event priority queue (e.g., a Splay Tree).

To cancel a shared event due to its parent event being rolled back, it is pulled off the parent event's cancellation list and placed into the destination MPI rank's shared `cancel_q` and the event's internal flag is marked as being a cancellation event. As part of normal event processing, the destination MPI rank will also poll the `cancel_q` and pull off (using the pthread lock) any cancellation events for processing. Because the cancellation or "anti-message" event and its positive event are the same event, no special lookup is required to "find" the positive event. Instead, flags have been set to denote its location within the MPI rank's event processing data structures, for example, the priority queue or processed event list. This direct cancellation process is the same as that used in the original shared memory ROSS implementation (Carothers et al. 2000).

To return shared events to their original MPI rank owners, each MPI rank finds correctly marked shared memory events during its normal fossil collection operation and places shared memory events back into the original MPI rank's `return_q`. This queue list is locked to ensure proper atomic access. Then locally when an MPI rank has exhausted its own shared event free list, it will replenish that list by pulling off the complete list of events in the `return_q` and inserting the whole list into the empty free list of shared events.

## 4 EXPERIMENTS

This section discusses the experimental design used to compare the performance of SHMEM-ROSS with MPI-ROSS. We describe the platforms used for the simulation runs, along with brief model descriptions and parameter settings.

### 4.1 Experimental Platform

All simulation runs are performed on the Theta Cray XC system at Argonne National Laboratory or the DRP Linux cluster at Rensselaer Polytechnic Institute's Center for Computational Innovation. Theta has 4,392 nodes, with each node containing a 64-core 1.3 GHz Intel Xeon Phi processor. Each node also has 16 GB of MCDRAM, along with 192 GB of DDR4 RAM. The DRP cluster consists of 64 nodes with each node containing two 8-core 2.6 GHz Intel Xeon E5-2650 processors. Each node has 128 GB of RAM and the nodes are connected via 56 Gb FDR InfiniBand. For runs performed on Theta, we use up to 64 cores per node, while on DRP, we use up to 16 cores per node.

### 4.2 Simulation Models

#### 4.2.1 CODES Dragonfly Model

For the CODES dragonfly model, we use a network configuration to simulate the Theta Cray XC system. We use a previous Theta configuration from April 2017 with 3,456 nodes connected by a high-radix dragonfly network (Argonne Leadership Computing Facility 2017; Faanes et al. 2012). The 864 routers are represented by one LP each, while the 3,456 nodes are each represented by two LPs, resulting in 7,776 LPs. For further details regarding the CODES implementation of the dragonfly model and Theta configuration, including validation results, we refer the reader to the works of Mubarak and Ross (2017a) and Mubarak et al. (2017c). For this model, we perform our experiments with a workload consisting of synthetically generated uniform random traffic and an adaptive routing protocol. We set the message size and packet sizes to each be 2,048 bytes. Each workload generator LP is configured to generate 200 messages throughout the simulation, with an interarrival time of 20 ns. We perform the experiments with 2 to 64 PEs on both Theta and the DRP cluster.

#### 4.2.2 CODES Fat-Tree Model

For simulations performed with the CODES fat-tree network model, we use a configuration to simulate the future Summit HPC system at Oak Ridge National Laboratory. This configuration uses a pruned 3-level fat-tree topology with 36-port switches and a total of 3,564 nodes. There are thus 504 switch LPs and 7,128 LPs to represent 3,564 compute nodes (2 LPs per simulated compute node), resulting in a total of 7,632 LPs. Further details on the CODES fat-tree model implementation can be found in the work of Wolfe et al. (2017). Similarly to the dragonfly simulations, we use uniform random traffic, with message and packet sizes set to 2,048 bytes. The interarrival time is again set to 20 ns and the packet injection rate is set to 90% of the terminal link bandwidth. All of the fat-tree experiments are performed on both Theta and the DRP cluster with 2 to 64 PEs.

#### 4.2.3 NeMo Model

NeMo is a neuromorphic processor architecture simulation framework built on top of ROSS. NeMo models the neurons, axons, and synapses of neurosynaptic cores with unique LP types (Plagge et al. 2016). In our NeMo simulations, we simulate the IBM True North chip with 4,096 neurosynaptic cores, each of which is mapped to ROSS PEs. Each neurosynaptic core contains 256 axon LPs, 256 neuron LPs, and 1 synapse LP that represents all 65,536 synapses present in the core. This results in a total of 2,101,248 LPs for our NeMo configuration. We perform the experiments on both Theta and DRP with 16 to 512 total PEs

incrementing by powers of 2. These runs are performed on 1 to 8 nodes of Theta and 1 to 32 nodes on the DRP cluster, depending on the number of PEs used. Simulated time in NeMo is in number of ticks, and all of our NeMo configurations run for a total of 1,000 ticks.

#### 4.2.4 PHOLD Model

For these experiments, we use a modified PHOLD model. Our modification ensures that remote events are exchanged between PEs located in the same shared memory group as opposed to a uniform random traffic pattern that is typical in PHOLD. This allows us to determine the performance gains ROSS experiences when fully exploiting the use of shared memory. For the PHOLD runs, we run with 128 LPs per PE and 8 KPs per PE. The mean used for timestamp distribution of events is set to 1 in all configurations, and we use three settings for the remote event rate: 0.1, 0.5, and 0.9. We use the PHOLD model to test SHMEM-ROSS at scale, so we perform these runs only on Theta, using 64 to 2,048 nodes of the system.

## 5 EVALUATION

We first discuss the runtime results for all of the simulated models described in Section 4. For all configurations of both CODES models and NeMo, we ran each simulation 10 times. The graphs for these results show the average runtime, and error bars denote the standard deviation for that configuration. Each model has different levels of remote events in the simulation. The percentage of remote events for these three models are shown in Table 1. For these models, the number of remote events increases as the number of PEs increase.

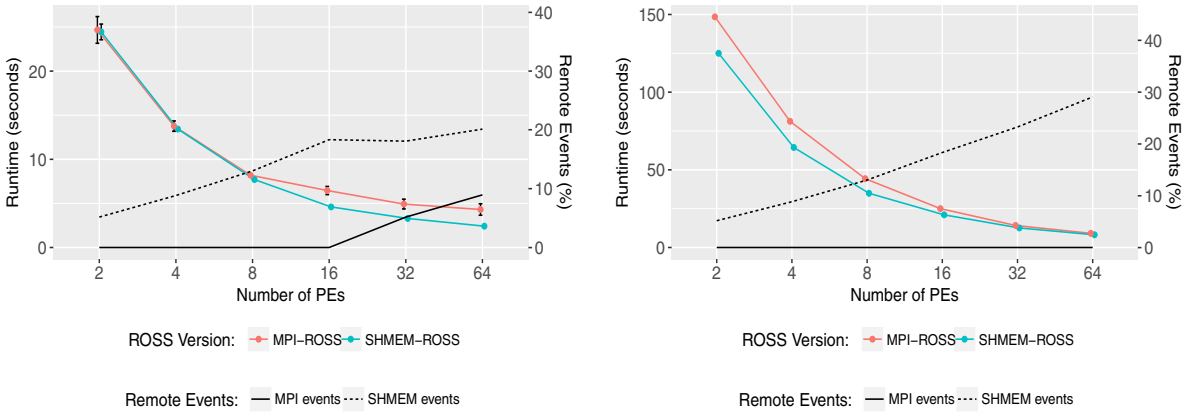
The runtimes for the simulations performed with the CODES dragonfly model are shown in Figure 2, with DRP results on the left and Theta results on the right. For DRP, we note that both versions of ROSS perform similarly for 8 PEs or less, while SHMEM-ROSS starts to outperform MPI-ROSS starting at 16 PEs. At this point, the percentage of remote events transferred through shared memory approaches 20%, while the percentage of MPI events stays below 10% when running on multiple nodes, allowing for the simulation to benefit from the use of shared memory. The dragonfly results on Theta appear to have a reverse trend from the DRP results. Here, SHMEM-ROSS experiences more runtime improvement at smaller PE counts than larger, despite the fact that remote events are now exchanged only through shared memory at all scales, due to Theta having more cores per node. However, Theta's processors have a slower clock rate (1.3 GHz) than the processors on DRP (2.6 GHz), contributing to increased runtimes over the DRP runs.

We examined the rollback behavior of the dragonfly model running on both machines (not pictured) and found that SHMEM-ROSS appears to help control the amount of rollbacks experienced for the DRP runs when increasing the number of PEs, while on Theta, the rollback behavior of this model is almost the same for both ROSS versions. It appears that a side effect of Theta's slower processors is more controlled rollback behavior upon which SHMEM-ROSS is unable to improve, whereas for the faster processors on the DRP cluster, the use of shared memory appears to help to control rollbacks.

Figure 3 shows the runtimes for the CODES fat-tree model, with DRP results on the left and Theta on the right. In the case of simulations performed on DRP, the largest runtime improvement for SHMEM-ROSS over MPI-ROSS is seen at 16 PEs, where the percentage of remote events peaks at approximately 30%. Unlike the dragonfly model, when continuing to increase the PEs (and thus, node count), the percentage of shared memory remote events starts to decrease and the percentage of MPI events increases at a faster rate. This may explain why the runtime difference between MPI-ROSS and SHMEM-ROSS starts to decrease at 32 and 64 PEs for the fat-tree model. With the fat-tree runs on Theta, we see a similar trend to the dragonfly Theta runs, namely increased runtimes in general, as well as the largest differences in runtime

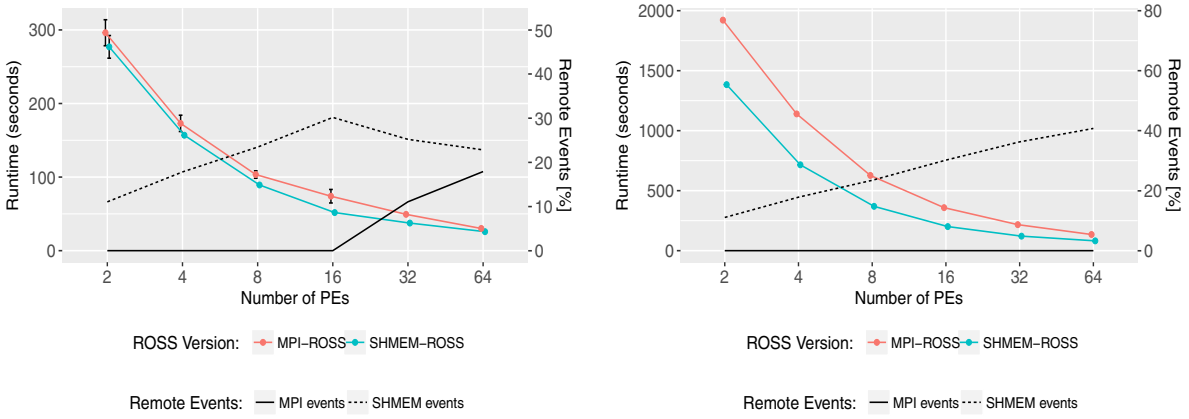
Table 1: Percentage of remote events for NeMo and the CODES network models.

| Network | Dragonfly | Fat-tree | NeMo |
|---|---|---|---|
| Remote Events (%) | 5–28% | 10–40% | $< 0.1\%$ |




(a) DRP Cluster Results (b) Theta Results

Figure 2: Runtimes for CODES Dragonfly model on DRP (a) and Theta (b). Red and blue lines denote runtime results for the two ROSS versions, while black lines show percentage of remote events for shared memory (dashed) and MPI (solid). Data points for the runtimes are the average of 10 simulation runs for each configuration, while error bars denote the standard deviation. Note that data points without error bars means that the standard deviation is near 0.

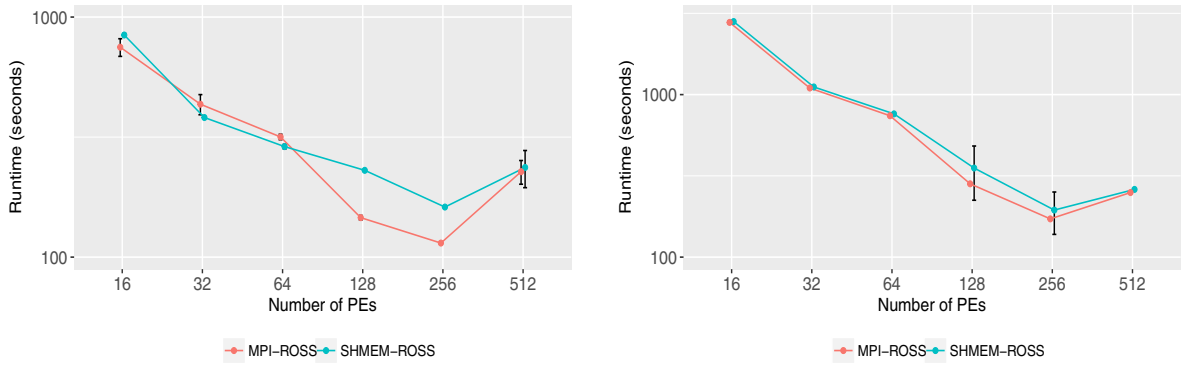



(a) DRP Cluster Results (b) Theta Results

Figure 3: Runtimes for CODES fat-tree model on DRP (a) and Theta (b). Red and blue lines denote runtime results for the two ROSS versions, while black lines show percentage of remote events for shared memory (dashed) and MPI (solid). Data points for the runtimes are the average of 10 simulation runs for each configuration, while error bars denote the standard deviation. Note that data points without error bars means that the standard deviation is near 0.

being experienced at lower PE counts. However, the rollback behavior (not pictured) remains the same for the fat-tree model across machines and ROSS versions. It is not clear why rollback behavior is affected by the system and version of ROSS used for the dragonfly model, but not the fat-tree model. In future work, we plan to extend our previous work in visual analysis of ROSS instrumentation data (Ross et al. 2016) to better understand the effect of the interplay of the hardware, simulation engine, and model characteristics on rollback behavior and simulation performance.

The runtime results for the NeMo model are shown in Figure 4 for DRP on the left and Theta on the right. MPI-ROSS shows performance similar to or better than that of SHMEM-ROSS in all configurations on both systems tested. These results are most likely due to the fact that NeMo experiences much fewer remote events than either CODES network model. NeMo has less than 0.1% remote events at all scales, which is much less than the dragonfly and fat-tree models (Table 1). Since NeMo has so few remote events, it does not experience the same level of MPI communication overhead as the CODES models, so it is unable to benefit from the use of shared memory.




(a) DRP Cluster Results

(b) Theta Results

Figure 4: Runtimes for NeMo model on DRP (a) and Theta (b). Data points are the average of 10 simulation runs for each configuration, while error bars denote the standard deviation. Note that data points without error bars means that the standard deviation is near 0.

We also perform a scaling studying using the PHOLD model on Theta, scaling from 4,096 to 131,072 PEs. We perform each configuration only once, due to CPU allocation hours on Theta and the scale of the runs performed (up to 2,048 nodes). Figure 5 shows the event rates from the PHOLD runs. We show event rate for PHOLD benchmark results to be consistent with other literature in this area. Each color represents a different remote event rate used in the simulation, while solid lines represent MPI-ROSS runs and dotted lines represent SHMEM-ROSS runs. Running PHOLD with SHMEM-ROSS outperforms MPI-ROSS in all cases, with the improvement in event rate getting larger as we increase the remote event rate of the model. In the case of a remote event rate of 0.1, we see an improvement in event rate by a factor of up to 1.14. For remote event rates of 0.5 and 0.9, we see improvements by factors of up to 1.43 in both cases. This is in line with the earlier discussed results, where the NeMo model experiences very few remote events and thus sees no performance improvement from the use of shared memory, while the CODES network models have higher remote event rates that may be able to take advantage of shared memory.

## 6 CONCLUSION

Because of the prevalence of many-core processors in modern HPC systems, such as Intel Xeon Phi processors in the Theta supercomputer at ANL, improving the intranode performance of applications, including PDES, is important. The work in this paper explores the use of shared memory with MPI processes in the ROSS Time Warp simulator. In particular, we avoid combining the use of MPI with pthreads, in order to avoid complexities and overheads associated with ensuring pthreads access MPI functionality in a thread-safe way. We tested the performance of SHMEM-ROSS against the pure MPI implementation of ROSS on two systems: the Theta supercomputer, with 64 cores per node, and a Linux cluster, with 16 cores per node. For the PHOLD benchmark, we found a consistent speedup in runtime performance while scaling up to 2,048 nodes of Theta. For the NeMo model, we found no performance

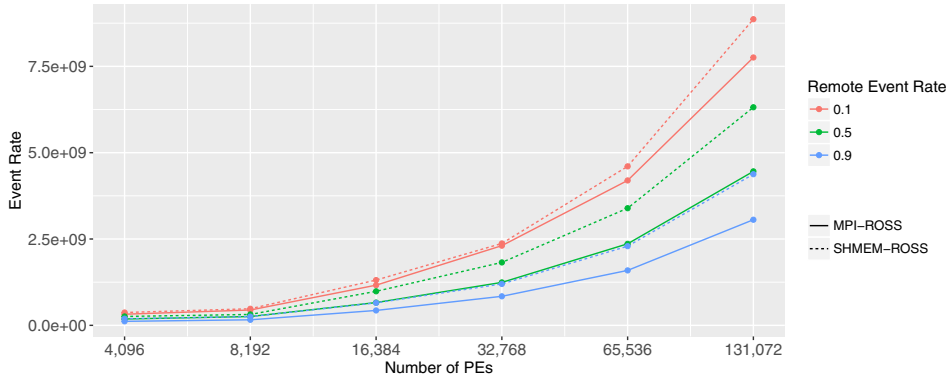



Figure 5: Event rate for PHOLD model simulations performed on Theta for various remote event rate settings.

improvment when using SHMEM-ROSS, which is due to NeMo having a remote event rate less than 0.1% at all scales.

For the two CODES network models on Theta, SHMEM-ROSS tends to result in faster runtimes than MPI-ROSS, however the difference between versions decreases as the number of PEs increases. On the DRP Linux cluster, this trend is reversed for dragonfly, where the largest performance differences are seen at higher PE counts. We believe this is also due to the fact that SHMEM-ROSS appears to help reduce the amount of rollbacks experienced at larger scales. However for the dragonfly runs performed on Theta, the slower processors appear to help control rollback behavior, and the use of shared memory provides no additional decrease in rollbacks. In contrast, the fat-tree model experiences no change in rollback behavior due to the system used nor ROSS version used, so in future work, we plan to use time series data from the simulation to better understand the interplay of the hardware, simulation engine, and model and its effect on the rollback behavior. In addition to this, we plan to make further use of shared memory in ROSS to add additional features, such as load balancing schemes. Currently LP to PE mappings are computed on the fly, making it more difficult to provide better mappings for load balancing. With SHMEM-ROSS, we plan to add mapping tables that are shared by all PEs on a node, which will help in generating more efficient mapping schemes. This should enable further performance improvements to models such as CODES network models that can experience large load imbalances depending on the type of network traffic being simulated.

## ACKNOWLEDGMENTS

This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computer Research (ASCR), under contract DE- AC02-06CH11357 and DE-SC0014917.

## REFERENCES

Adiga, N. R., 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. 2005. “Blue Gene/L Torus Interconnection Network”. *IBM Journal of Research and Development* 49(2):265–276.

Argonne Leadership Computing Facility 2017. “Theta”. https://www.alcf.anl.gov/theta. Accessed Mar 30, 2018.

Barnes, P. D., C. D. Carothers, D. R. Jefferson, and J. M. LaPre. 2013. “Warp Speed: Executing Time Warp on 1,966,080 Cores”. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. May 19th-22nd, Montreal, Canada, 327–336.

Bauer, D., C. D. Carothers, and A. Holder. 2009. "Scalable Time Warp on Blue Gene Supercomputers". In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*. June 22nd-25th, Lake Placid, NY, USA, 35–44.

Carothers, C. D., D. Bauer, and S. Pearce. 2000. "ROSS: A High-Performance, Low-Memory, Modular Time Warp System". In *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation*. May 28th-31st, Bologna, Italy, 53–60.

Carothers, C. D., D. Bauer, and S. Pearce. 2002. "ROSS: A High-Performance, Low-Memory, Modular Time Warp System". *Journal of Parallel and Distributed Computing* 62(11):1648–1669.

Chen, D., P. Heidelberger, C. Stunkel, Y. Sugawara, C. Minkenberg, B. Prisacari, and G. Rodriguez. 2016. "An Evaluation of Network Architectures for Next Generation Supercomputers". In *Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*. November 13th–18th, Salt Lake City, UT, USA, 11–21.

Das, S., R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. 1994. "GTW: A Time Warp System for Shared Memory Multiprocessors". In *Proceedings of the 1994 Winter Simulation Conference*, edited by J. D. Tew et al., 1332–1339. Piscataway, New Jersey: IEEE.

Faanes, G., A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard. 2012. "Cray Cascade: A Scalable HPC System Based on a Dragonfly Network". In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. November 10th-16th, Salt Lake City, UT, USA.

Fujimoto, R. M. 1990. "Parallel Discrete Event Simulation". *Communications of the ACM* 33(10):30–53.

Ghosh, K., and R. M. Fujimoto. 1991. "Parallel Discrete Event Simulation Using Space-Time Memory". Technical report, Georgia Institute of Technology.

Gropp, W., and R. Thakur. 2006. "Issues in Developing a Thread-Safe MPI Implementation". In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. September 23rd-26th, Vienna, Austria, 12–21.

Jagtap, D., N. Abu-Ghazaleh, and D. Ponomarev. 2012a. "Optimization of Parallel Discrete Event Simulator for Multi-Core Systems". In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*. May 21st-25th, Shanghai, China, 520–531.

Jagtap, D., K. Bahulkar, D. Ponomarev, and N. Abu-Ghazaleh. 2012b. "Characterizing and Understanding PDES Behavior on Tilera Architecture". In *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*. July 15th-19th, Zhangjiajie, China, 53–62.

Jefferson, D. R. 1985. "Virtual Time". *ACM Transactions on Programming Languages and Systems* 7(3):404–425.

Martin, D. E., P. A. Wilsey, R. J. Hoekstra, E. R. Keiter, S. A. Hutchinson, T. V. Russo, and L. J. Waters. 2003. "Redesigning the WARPED Simulation Kernel for Analysis and Application Development". In *Proceedings of the 36th Annual Symposium on Simulation*. March 30th-April 2nd, Washington, DC, USA, 216-233.

Mehl, H., and S. Hammes. 1993. "Shared Variables in Distributed Simulation". In *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation (PADS '93)*. May 16th-19th, San Diego, CA, USA, 68–75.

Mubarak, M., and R. B. Ross. 2017a. "Validation Study of CODES Dragonfly Network Model with Theta Cray XC System". Technical report, MCS Division, Argonne National Laboratory.

Mubarak, M., C. D. Carothers, R. B. Ross, and P. Carns. 2017b. "Enabling Parallel Simulation of Large-Scale HPC Network Systems". *IEEE Transactions on Parallel and Distributed Systems* 28(1):87–100.

Mubarak, M., N. Jain, J. Domke, N. Wolfe, C. Ross, K. Li, A. Bhatele, C. D. Carothers, K.-L. Ma, and R. B. Ross. 2017c. "Toward Reliable Validation of HPC Network Simulation Models". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan et al., 659–674. Piscataway, New Jersey: IEEE.

Pelkey, J., and G. Riley. 2011. "Distributed Simulation with MPI in ns-3". In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. March 22nd-24th, Barcelona, Spain, 410–414.

Pellegrini, A., and F. Quaglia. 2015. "NUMA Time Warp". In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS '15)*. June 10th-12th, London, UK, 59–70.

Plagge, M., C. D. Carothers, and E. Gonsiorowski. 2016. "NeMo: A Massively Parallel Discrete-Event Simulation Model for Neuromorphic Architectures". In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. May 15th-18th, Banff, Canada, 233-244.

Ross, C., C. D. Carothers, M. Mubarak, P. Carns, R. Ross, J. K. Li, and K.-L. Ma. 2016. "Visual Data-Analytics of Large-Scale Parallel Discrete-Event Simulations". In *Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*. November 13th–18th, Salt Lake City, UT, USA, 87-91.

Wang, J., D. Ponomarev, and N. Abu-Ghazaleh. 2012. "Performance Analysis of a Multithreaded PDES Simulator on Multicore Clusters". In *Proceedings of the 2012 ACM/IEEE/SCS 26th workshop on Principles of Advanced and Distributed Simulation*, 93–95. July 15th-19th, Zhangjiajie, China, 53–62.

Wang, J., K. Bahulkar, D. Ponomarev, and N. Abu-ghazaleh. 2013. "Can PDES Scale in Environments with Heterogeneous Delays?". In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. May 19th-22nd, Montreal, Canada, 35–46.

Wang, J., D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev. 2014. "Parallel Discrete Event Simulation for Multi-Core Systems: Analysis and Optimization". *IEEE Transactions on Parallel and Distributed Systems* 25(6):1574–1584.

Wolfe, N., M. Mubarak, N. Jain, J. Domke, A. Bhatele, C. D. Carothers, and R. B. Ross. 2017. "Preliminary Performance Analysis of Multi-Rail Fat-Tree Networks". In *Proceedings of the 2017 17th IEEE/ACM international symposium on Cluster, Cloud and Grid Computing*. May 14th-17th, Madrid, Spain, 258–261.

## AUTHOR BIOGRAPHIES

**Caitlin Ross** is a Ph.D. candidate in the Department of Computer Science at Rensselaer Polytechnic Institute. Ms. Ross received her B.S. in computer science from the University of North Carolina at Greensboro in 2014. Her email address is rossc3@rpi.edu.

**Jianping Kelvin Li** is a graduate student in the Computer Science Department at the University of California, Davis. Mr. Li received his B.S. in computer engineering from the University of California, Davis, in 2009. His email address is kelli@ucdavis.edu.

**Misbah Mubarak** is an assistant scientist in the Mathematics and Computer Science Research Division at Argonne National Laboratory. Dr. Mubarak received her Ph.D. in computer science from Rensselaer Polytechnic Institute in 2015. Her email address is mmubarak@anl.gov.

**Christopher D. Carothers** is a professor of computer science at Rensselaer Polytechnic Institute. Professor Carothers received his Ph.D. from Georgia Institute of Technology in 1997. His email address is chrisc@cs.rpi.edu.

**Kwan-Liu Ma** is a professor of computer science at the University of California, Davis. Professor Ma received his Ph.D. in computer science from the University of Utah in 1993. His email address is ma@cs.ucdavis.edu.

**Robert B. Ross** is a senior computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory. Dr. Ross received his Ph.D. in computer engineering from Clemson University in 2000. His email address is rross@mcs.anl.gov.