Can PDES Scale in Environments with Heterogeneous Delays?

Jingjing Wang, Ketan Bahulkar, Dmitry Ponomarev and Nael Abu-Ghazaleh
Computer Science Department
State University of New York at Binghamton
{jwang36, kbahulkar, dima, nael}@cs.binghamton.edu

ABSTRACT

The performance and scalability of Parallel Discrete Event Simulation (PDES) is often limited by communication latencies and overheads. The emergence of multi-core processors and their expected evolution into many-cores offers the promise of low latency communication and tight memory integration between cores; these properties should significantly improve the performance of PDES in such environments. However, on clusters of multi-cores (CMs), the latency and processing overheads incurred when communicating between different machines (nodes) far outweigh those between cores on the same chip, especially when commodity networking fabrics and communication software are used. It is unclear if there is any benefit to the low latency among cores on the same node given that communication links across nodes are significantly worse. In this study, we examine the performance of a multi-threaded implementation of PDES on CMs. We demonstrate that the inter-node communication costs impose a substantial bottleneck on PDES and demonstrate that without optimizations addressing these long latencies, multi-threaded PDES does not significantly outperform the multiprocess version despite direct communication through shared memory on the individual nodes. We then propose three optimizations: message consolidation and routing, infrequent polling and latency-sensitive model partitioning. We show that with these optimizations in place, threaded implementation of PDES significantly outperforms process-based implementation even on CMs.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event, Parallel*

General Terms

Performance

Keywords

PDES, Cluster of Multi-cores, Multi-thread

1. INTRODUCTION

Communication latency represents one of the major challenges in scaling many important classes of parallel applications, especially those that exhibit fine-grained communication and synchronization patterns, such as Parallel Discrete Event Simulation (PDES). The emergence of multi-core processors and the trend of increasing the number of cores per processor offer a significant advantage for PDES applications: the low communication latency between cores allows for faster communication, and the tight memory integration allows for more efficient communication abstractions that reduce the software overheads of communication. Unfortunately, these advantages are limited to small scales: for large scale PDES applications that require more cores than is available on a single node[1], it is necessary to use a cluster of multi-cores (CMs). In such an environment, the communication delays and software overheads for communication across machines (inter-node communication) can be substantially higher than those between cores on the same machine (intra-node communication).

To understand and quantify this impact, we investigate the performance of PDES [16] on CMs. Specifically, the question that we address in this paper is the following: In the presence of heterogeneous delays, do fine-grained applications such as PDES benefit from the low latency available between some cores, or are they limited by the performance of the slowest links? To explore this question, we perform experiments on a cluster of multi-core nodes connected using Gigabit Ethernet and using MPI for communication across nodes. For cores on the same node, we explore the use of both MPI as well as more efficient communication exploiting the shared memory hierarchy between the cores on the same node. We show that the remote communication across nodes plays a critical role in determining the performance of PDES. The message processing delay on the communicating thread becomes a performance bottleneck of PDES. As a result, much of the available processing time is consumed in sending and receiving these events. Moreover, the high communication latency results in many messages arriving late, slowing down the simulation progress even further by causing rollbacks.

Therefore, we argue that the impact of the heterogeneous

[1]In this paper, we use node to mean a single multi-core machine.

delays must be considered as a first class design consideration when developing PDES algorithms to run on CMs. We demonstrate three techniques that significantly reduce the impact of the heterogeneous delays. The first technique we investigate is consolidated message routing between machines. In particular, to reduce the impact of message sending and receiving overheads, we combine messages originating from different cores on one machine to different cores on another machine to amortize the software overhead of processing them. Dedicated communication threads combine the messages on the sending side, without delaying messages. On the receiving side, the communication threads extract the individual messages and route them to the appropriate core using shared memory. We further improve the performance of the receiver communication thread by adjusting the frequency of polling for message arrival; with message consolidation, the number of messages is reduced, allowing us to reduce the frequency of the expensive polling operation. Combined, these two optimizations allow the threaded PDES implementation to achieve a 4.5X improvement in performance compared to the process-based implementation.

To further highlight the need for algorithmic awareness of the CM topology, we implement model partitioning in PDES in a way that is aware of the high cost of inter-node communication. In particular, we modified a recently proposed partitioning algorithm [1] to first partition across machines (minimizing the model communication between machines) and then partition the work across the cores on the same machine. We compare this strategy to one that partitions between the cores without consideration to the high communication latency between machines. Again, significant improvement in performance (up to 44% in some scenarios) is achieved by simply informing the partitioning tool of the communication cost hierarchy.

The remainder of the paper is organized as follows. Section 2 provides background information about the multi-core cluster environment we use, as well as PDES simulator. In this section, we also analyze the impact of the heterogeneous communication latencies on CMs for PDES. In Section 3, we focus on optimizations to help overcome this impact for PDES application. Section 4 presents an experimental evaluation of PDES performance. In Section 5, we overview some related work. Finally, in Section 6 we present some concluding remarks.

2. IMPACT OF HETEROGENEOUS CM LATENCIES ON PDES PERFORMANCE

In this section, we first present some background regarding clusters of multi-cores (CMs) used in our experiments. We follow this by evaluating PDES performance on CMs. The goal of these experiments is to show the large impact that the heterogeneous latencies present in these environments on PDES performance.

2.1 Multi-cores and Clusters of Multi-cores

In our experiments we use a cluster of 4-core Intel core i7 machines (Figure 1), running Debian 6.02 distribution for each node. Each core has private L1 and L2 caches, and shares the L3 cache with other cores. In addition, Simultaneous Multi-Threading (SMT) is employed on each node to increase the degree of available parallelism (SMT is called Hyper-Threading on Intel processors). More precisely, each core has two hardware threads which share most of the core pipeline structures, as well as the L1 and L2 caches. With 4 cores, each supporting two hardware threads, each node can execute up to 8 concurrent hardware threads. The nodes are connected using a Gigabit-Ethernet network.

We first use an MPI-based Ping-Pong benchmark to evaluate the communication latency in this environment. There are three types of communication in such CMs : intra-core, inter-core, and inter-node, as shown in Table 1. The intra-core communication occurs between two hardware threads on the same core, while inter-core communication happens among different cores on the same node. Inter-node communication is the communication over the network. Table 1 shows these three types of communication latency under different message sizes on the multi-core cluster connected through a Gigabit Ethernet switch. At any message size, the latency of inter-node communication is approximately two orders of magnitude larger than that of other two types of intra-node communication.

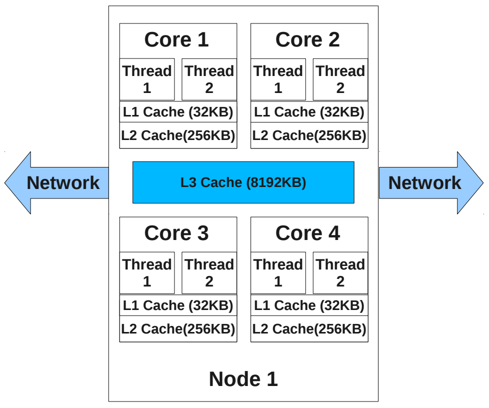


Figure 1: A cluster of Intel Core-i7 nodes

Message Size (Bytes)	intra-core	inter-core	inter-node
4	0.22	0.24	62.38
32	0.25	0.25	62.23
256	0.28	0.3	64.82
1024	0.35	0.38	78.06
8192	1.17	1.42	150.63
16K	2.37	2.88	268.61

Table 1: Heterogeneous Latency on the Cluster of Core-i7 Nodes (μSecs)

2.2 PDES Performance on CMs

We use a Parallel Discrete Event Simulation (PDES) engine to study the impact of heterogeneous latencies on CMs. PDES is an important application as it supports efficient simulation of large scale models in many domains including computing and telecommunication systems. In PDES, a simulation model is partitioned across multiple processing elements (PEs). Each PE processes the events destined to it in time-stamp order. For correct simulation, synchronization is needed to ensure that events generated from other PEs are also executed in the correct order [2, 16]. In conservative simulation, PEs must wait for assurance from other PEs that it is safe for them to process an event. In contrast,

	0% remote	20% remote	40% remote	60% remote	80% remote	100% remote
0% regional	7.3 sec	44.7 sec	70.3 sec	95.6 sec	117.5 sec	141 sec
20% regional	10.9 sec	44.4 sec	70.7 sec	95 sec	117.5 sec	N/A
40% regional	12.3 sec	44.9 sec	71.2 sec	95.4 sec	N/A	N/A
60% regional	12.8 sec	45.5 sec	71.9 sec	N/A	N/A	N/A
80% regional	14.3 sec	47.3 sec	N/A	N/A	N/A	N/A
100% regional	15.3 sec	N/A	N/A	N/A	N/A	N/A

Table 2: Effect of Heterogeneous Latency for baseline ROSS-CMT

	0% remote	20% remote	40% remote	60% remote	80% remote	100% remote
0% regional	N/A	100%	100%	100%	100%	100%
20% regional	0%	72.5%	82.3%	86%	87.3%	N/A
40% regional	0%	58.5%	70%	75.5%	N/A	N/A
60% regional	0%	48.9%	60.6%	N/A	N/A	N/A
80% regional	0%	41.9%	N/A	N/A	N/A	N/A
100% regional	0%	N/A	N/A	N/A	N/A	N/A

Table 3: The Rollback Percentage caused by remote events

optimistic simulation allows each PE to proceed without explicit synchronization. However, if an event is received with a time-stamp earlier than the current simulation time (a straggler event), the simulation is rolled back to a simulation time before that of the straggler event and re-executed. In both types of simulation, communication latency and software overheads play a critical role in determining performance. These overheads determine how fast event messages are communicated affecting simulation progress. The high latency also influences synchronization which has substantial effect on the progress rate and efficiency of the simulation.

We use the Rensselaer's Optimistic Simulation System (ROSS) as the simulation engine for our studies. ROSS is a highly efficient state-of-the-art simulator, that uses processes for each PE. Processes communicate with messages using the MPI library [6]. In the original ROSS simulator (ROSS-MPI), two message copy operations are needed for each message even when the PEs are on the same node, with access to shared memory. Message copies significantly increase the overhead of communication. In order to solve this performance bottleneck, a multi-threaded ROSS simulator (ROSS-MT) was implemented for a single multi-core shared memory node [19]. ROSS-MT uses message pointers to exchange messages and can avoid the message copying, forming, and other MPI overheads.

ROSS-MT is limited to a single node. To reach higher scales, an extended version of ROSS-MT, called ROSS-CMT, was recently developed to support CMs [33]. In ROSS-CMT, in order to avoid the overhead when multiple threads invoke MPI functions simultaneously, only one communication thread on each node performs communication across the network. The communication thread looks up the output queue of each thread in a round robin fashion, and sends remote events to the corresponding destination nodes. Once the communication thread at the receiver side probes (or polls) the event successfully, it then inserts the pointer of this event to the input queue of the destination thread. Generally speaking, ROSS-CMT performs better than ROSS-MPI on CMs, however, the delay of message processing at the side of communication thread imposes a performance bottleneck [33]. In this paper we use ROSS-CMT to study the impact of the heterogeneous latencies on CMs. Different from [33], this work provides three optimizations to hide high latency of inter-node communication for ROSS-CMT. Finally, we offer a detailed performance analysis of the optimized ROSS-CMT, by comparing it with both baseline ROSS-CMT and ROSS-MPI on CMs.

To characterize the impact of CMs on ROSS-CMT, we use the classical Phold benchmark [17]. Phold is a standard benchmark used in performance evaluation of PDES. It consists of a number of simulation objects distributed among multiple PEs. In our experiments, each PE is mapped to one thread in ROSS-CMT, or one process in ROSS-MPI. During execution each object randomly picks up a target and sends a time-stamped event message to the target. Upon receipt of the event, a new event may be generated to another target. Phold is controllable; allowing the percentage of communication between different objects to be specified to control the ratio between local, regional and remote events. Phold is a synthetic model with simple dependencies among objects, which allows us to study behavior under controlled conditions.

Table 2 shows the performance of ROSS-CMT with optimistic simulation, under different percentages of regional and remote communication on 4 nodes, with 8 threads each. Remote communication refers to traffic across nodes, while regional communication indicates the communication between cores on the same node. The rest of the communication is local and occurs within the same PE. Table 2 shows that the execution time increases substantially as the remote communication increases; at the case of 80% remote communication and 20% regional communication, the simulation runs approximately 10 times slower than 0% remote communication case with the same percentage of regional communication. Clearly, the impact of regional communication for ROSS-CMT is much less than that of remote communication. For example, at 0% remote communication, the performance drops only from 7.3 seconds to 15.3 seconds when regional communication increases from 0% to 100%.

Table 3 shows the percentage of rollbacks caused by remote communication. At the case of 20% remote communication and 20% regional communication, 72.5% of total rollbacks are caused by remote events. Rollbacks occur when events are arriving late and the local simulation time proceeds beyond their simulation time. It is considerably more likely for a rollback to be triggered by an incoming remote event, rather than regional event, because of the high latency on the slow communication link used to send the remote events. Thus, it is necessary to consider optimizations to reduce the cost of remote communication in PDES.

3. MANAGING HETEROGENEOUS COMMUNICATION LATENCY

In this section, we discuss the use of three optimizations to reduce the message communication overheads, and to hide the inter-node communication latency on CMs. The theme of these optimizations is to focus on the impact of the expensive communication links, by reducing the frequency of communication across them. We first describe the implementation of message consolidation on ROSS-CMT, where multiple messages are combined and routed through the slow links together. Next, we build up on the message consolidation technique by exploiting the observation that fewer messages now arrive from the distant links. To capitalize on this, we propose infrequent polling to reduce the frequency of the expensive operation to check for the incoming messages. Finally, we investigate making the high cost links visible to model partitioning in ROSS-CMT to better map the model around them.

3.1 Optimization 1: Message Consolidation and Routing

Message send operations across nodes incur significant overheads including multiple buffer copies and system calls/OS delays on both the sender and receiver sides. Therefore, when inter-node communication is frequent, these overheads can dominate, increasing the message processing latency, but also potentially delaying critical messages and slowing down overall application progress.

We address this limitation by employing an optimization called message consolidation and routing. Message consolidation creates designated communication threads on each node that act as consolidation points for communication. Instead of communicating directly, threads prepare their outgoing messages which are then collected by the communication threads and consolidated when possible to reduce communication overhead. More precisely, each thread maintains multiple output queues, and queues each remote event into the appropriate one based on which machine this event will be sent to. The communication thread looks up every output queue of each thread, and aggregates multiple events into one message before transmission. At the receiver side, the messages are deconsolidated and delivered to the appropriate thread.

A critical parameter for message consolidation is the number of messages consolidated in one send. In our approach, we set a threshold based on the cumulative size of the consolidated message. This approach allows us to match the sent message size to the underlying communication medium maximum payload size in order to avoid expensive MAC layer fragmentation. For Ethernet, the maximum payload size is 1500 bytes, which allows us to aggregate up to 10 event messages in ROSS-CMT. The use of Ethernet jumbo frames could offer room for higher degrees of consolidation, especially for applications that have large size messages.

This approach bears some similarity to traditional message consolidation (also known as message aggregation) [11]. In these approaches, messages from the same sender to the same receiver are consolidated to amortize overhead. Often, the sender artificially delays messages in hope of receiving additional messages to send to increase the opportunity for aggregation. In contrast, the proposed optimization combines messages from different senders to different receivers as long as these messages share the source and destination nodes. As such, it exposes significantly higher opportunities for consolidation and avoids the need for delaying messages in hopes of receiving later messages for consolidation. In other words, consolidation is not only carried out over time, it is also carried out across different senders and receivers sharing the same source and destination nodes. By focusing consolidation on the slow inter-node links–no consolidation is carried out between cores on the same node– we achieve the highest reduction in communication overheads.

3.2 Optimization 2: Infrequent Polling for Incoming Messages

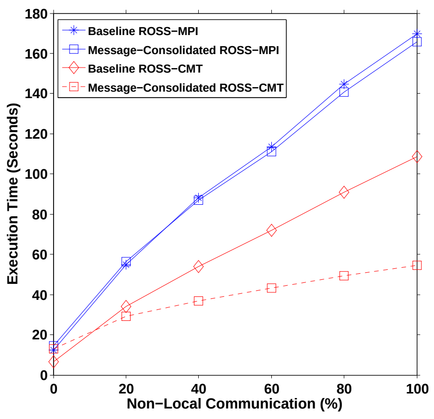
In ROSS-CMT, each communication thread maintains a loop for sending and receiving remote messages. The next loop iteration will be processed if the communication thread sends or receives a remote message during the current loop iteration. In order to detect the incoming remote messages, each communication thread probes (or polls) the network every iteration. However, probing is an expensive operation, and probing too aggressively increases overhead, often discovering that no message is available. To avoid unnecessary expensive probes, we implemented infrequent polling in ROSS-CMT. In particular, the frequency of polling is controlled by a configurable parameter, called *polling frequency*. For example, a polling frequency of 4 indicates that a probe operation occurs after every fourth iterations processed. In addition, we investigated both static and dynamic polling strategies. For the static polling strategy, the polling frequency is fixed during the simulation. On the other hand, the polling frequency can be adjusted in the dynamic polling strategy based on the behavior of previous probes. It reaches an effective probing rate that balances the overhead of probing against the loss of efficiency that may result if some messages are received late. We also note that message consolidation can benefit probing because it reduces the message frequency, allowing us to probe less frequently.

Infrequent polling has been proposed before for optimizing the performance of asynchronous applications such as PDES [29]. However, when applying message consolidation in a CM environment the behavior is significantly different because a single thread polls for a group of simulation threads making the communication pattern different. The interplay between message consolidation (which reduces the number of overall messages) and infrequent polling has not been studied before.

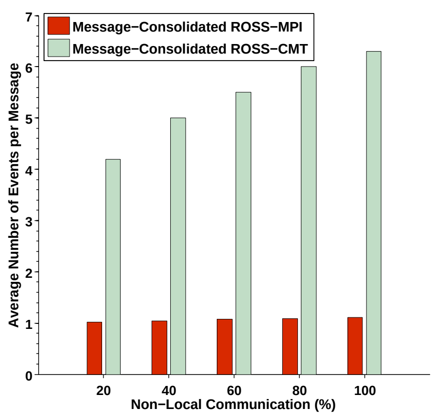
3.3 Optimization 3: Exposing Heterogeneous Latencies to Model Partitioning

Partitioning can play a significant role in reducing the communication overhead for parallel applications [28, 23]; by keeping the most heavily communicating objects together, the overhead of communication can be controlled. A joint consideration of partitioning is maintaining load balancing between the processing elements by evenly distributing the work among them.

In an environment with heterogeneous delays, the higher delays between nodes can be exposed to the partitioning tool to allow it to make more informed partitioning deci-



(a) Execution Time Comparison



(b) Average Number of Events per Consolidated Message

Figure 2: Performance of Message Consolidation on 32-Way PDES Simulation

sions. Without this information, the work would be simply partitioned between the cores without consideration to the heterogeneous delays between them.

As a target object of each event is randomly selected in the classical Phold model, there is no static communication structure in it. Thus, we use the hierarchical Phold model introduced in [1], to study the impact of partitioning on CMs. In this model, groups of objects are arranged in a hierarchical communication structure. Object groups closer to each other communicate more often while the farther groups have progressively less communication. Although this is still a synthetic model, it exhibits features of real models both in terms of topology and communication pattern. We use a partitioning algorithm that first profiles a short simulation using the hierarchical Phold model and then uses its behavior information (e.g., communication frequency of each pair of objects) to carry out partitioning [1]. The implementation uses a state of the art partitioning engine (*hMetis* [20]) to partition the simulation graph which is annotated with the profiling information. *hMetis* uses a multilevel partitioning algorithm for a weighted graph. It first coarsens the graph by collapsing edges and vertices. After multiple levels of coarsening, partitioning is started from the coarsest graph. Finally, a partition for the original graph is constructed and refined during the uncoarsening and refinement phase [20]. In our experiment, two types of partitioning strategies are compared. A latency heterogeneity-sensitive partitioning computes a node-level partition first and then each part is partitioned again at the core level. This strategy ensures that the groups of objects with most communication will be placed on the same node. It is compared with a latency-oblivious strategy introduced in [1] where the model is partitioned across cores, without differentiation between the heterogeneous latency among them.

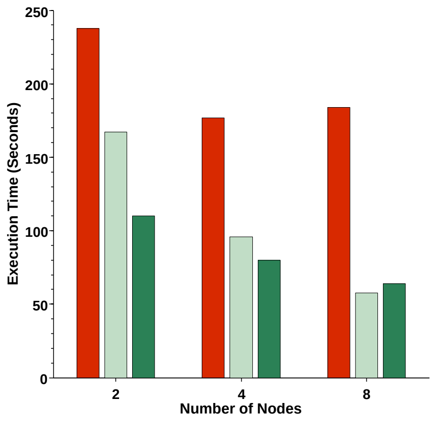
4. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we study the effectiveness of the three proposed optimizations on the performance of PDES. Our results demonstrate that these optimizations allow us to mitigate the negative impact of slow communication links on PDES performance and scalability.

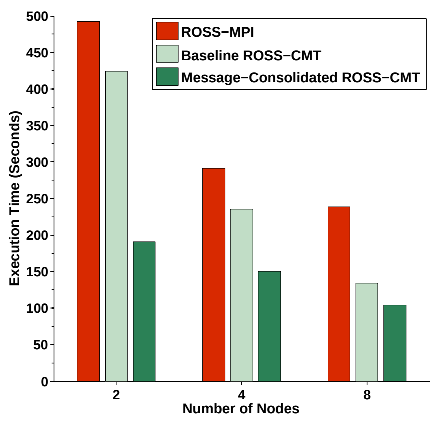
4.1 Message Consolidation

In our first experiment, we evaluate the performance of ROSS-CMT and ROSS-MPI with and without message consolidation, by using the classic Phold [17] benchmark, with 1000 objects for each PE. Figure 2 shows the impact of message consolidation on a 32-way optimistically synchronized simulation. In Figure 2(a), we see the run time as a function of the percentage of non-local messages (i.e., the percentage of Phold event targets that reside on another core; this includes cores on the same node, as well as cores on other nodes). As the non-local message percentage increases, message consolidation allows both versions of the simulation to improve their performance compared to the case without message consolidation. ROSS-CMT performs better than the MPI-based implementation with or without message consolidation. The benefits that ROSS-CMT obtains from message consolidation are significantly higher than those obtained by ROSS-MPI, because the CMT version can consolidate messages originating from any thread within the same node. In contrast, ROSS-MPI can only consolidate the messages from one process to another process, and as a result, it benefits from consolidation only slightly in the best case, and it is even harmed in some cases. These effects are shown in Figure 2(b), which depicts the average number of events that are consolidated into a message. Clearly, ROSS-CMT is able to consolidate a significantly higher number of messages eliminating the high per-message sending overhead. At 100% non-local communication, ROSS-CMT with the message consolidation is capable of providing a performance gain of 2X against the baseline ROSS-CMT, and 3X against ROSS-MPI.

In the next experiment, we show the impact of message consolidation on PDES scalability at 20% remote communication (Figure 3(a)), and 80% remote communication (Figure 3(b)) respectively. We fix the total number of objects (to 60480), and equally distribute them across the PEs. The simulation is performed on 2 nodes, 4 nodes, and 8 nodes respectively, with 8 hardware threads used for each node. Message consolidation achieves a performance gain of up to 2.2X against the baseline, and up to 2.5X against ROSS-MPI at 80% communication. We also discover that the message-consolidated ROSS-CMT performs worse than the baseline

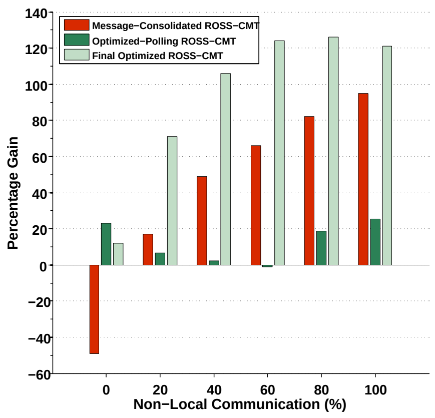


(a) 20% Remote Communication

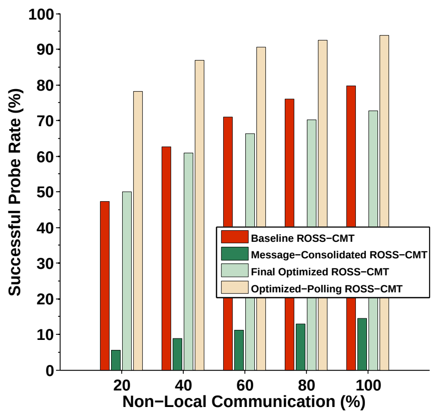


(b) 80% Remote Communication

Figure 3: Impact of Message Consolidation on PDES Simulation for Different Number of Nodes



(a) Performance Improvement (against Baseline ROSS-CMT)



(b) Polling Success

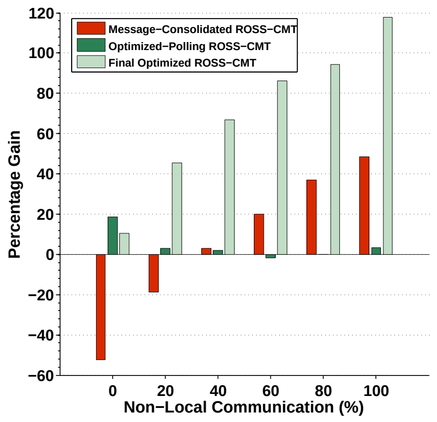
Figure 4: Infrequent Polling and Message Consolidation for 32-way PDES

ROSS-CMT with the case of 8 nodes at 20% remote communication. This is because of a large number of unsuccessful probe operations, leading to our next optimization.

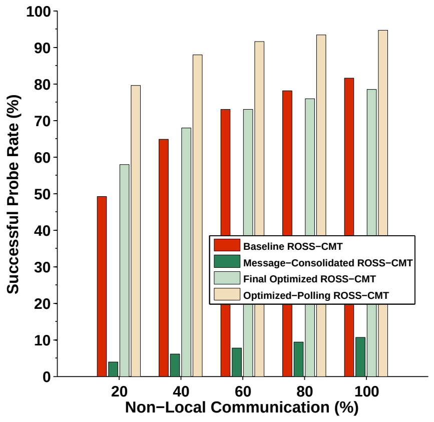
4.2 Infrequent Polling

In asynchronous parallel applications, a receiver does not know when to expect communication from a sender, and has to resort to polling. Polling is an expensive operation on communication channels that are inter-node. In our next experiment, we study the performance impact of infrequent polling strategy applied to PDES, since it is an asynchronous application. We investigated several polling periods, and selected 4 (poll after processing every 4th iteration) because it works well across a range of communication frequencies and simulation scales. We also investigated adapting the polling frequency, but the effect was minor. Polling also interplays with message consolidation, since consolidation reduces the number of communication messages, increasing the chance of unsuccessful polls. Figure 4(a) shows the percentage gain in performance with infrequent polling on its own as well as when it is combined with message consolidation for a 32-way simulation. Somewhat surprisingly, there is a large performance drop of about 50% in message-consolidated ROSS-CMT at the non-local percentage of 0. We expected some performance loss since the message consolidation overhead is incurred, without finding opportunities for consolidation. While infrequent polling on its own results in modest improvements in performance (up to 20%), when combined with message consolidation it significantly improves performance (up to 120% relative to baseline). As message consolidation combines messages, they arrive less frequently, allowing infrequent polling to successfully eliminate message probes. Figure 4(b) shows the *successful probe rate* for each of the three simulator versions, as well as the baseline. We note that the baseline ROSS-CMT has a very good *successful probe rate* because of its frequent communication across the network as each communication point represents all the threads on that node. In contrast, ROSS-CMT with message consolidation only has much lower *successful probe rate* than the others, since message frequency is lower, but still has substantially shorter execution time (up to 90% improvement) than the baseline ROSS-CMT as shown in Figure 4(a). We repeated the experiment for a 64-way simulation (Figure 5(a) and Figure 5(b)), with similar results.

4.3 Latency-Sensitive Model Partitioning

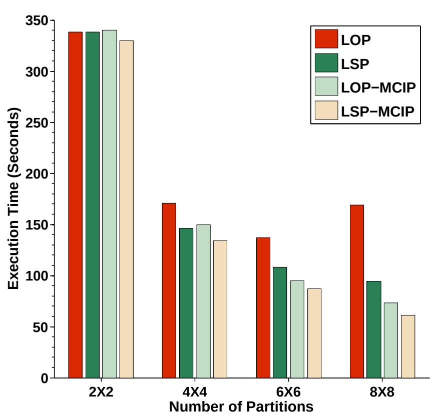


(a) Performance Improvement (against Baseline ROSS-CMT)

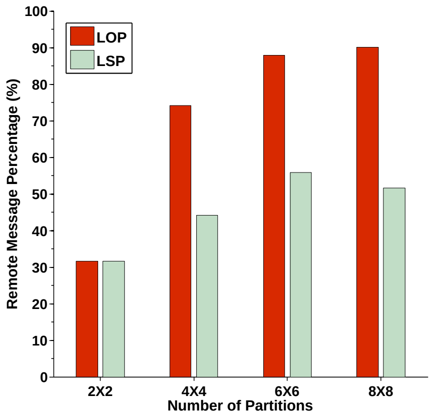


(b) Polling Success

Figure 5: Infrequent Polling and Message Consolidation for 64-way PDES



(a) Execution Time Comparison



(b) Remote Message Percentage

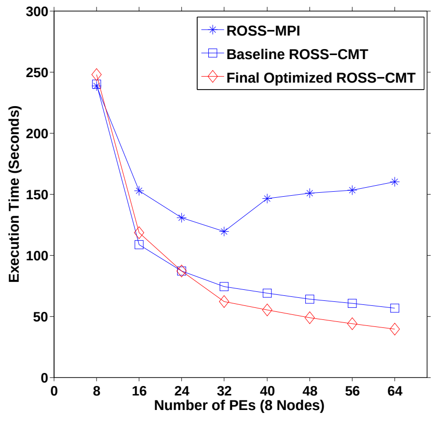
Figure 6: Performance Evaluation of Different Partitioning Strategies

Figure 6 shows the impact of partitioning sensitive to heterogeneous latencies on CMs. In Figure 6, the number of partitions N x T indicates that N nodes are used with T threads per node. Each partition is mapped to one thread. This experiment uses a hierarchical Phold model [1]. The model is initialized with 4 initial events per object. In our experiment, four types of partitioning strategies are compared. A *Latency-Sensitive Partitioning (LSP)* computes a node-level partition first and then each part is partitioned again at the core level. This strategy ensures that the groups of objects with most communication are placed on the same node. It is compared with a *Latency-Oblivious Partitioning (LOP)* where the latency heterogeneity is ignored and the partitioning tool tries to minimize all communication between cores. The remaining two strategies combine the partitioning with both Message Consolidation and Infrequent Polling (MCIP).

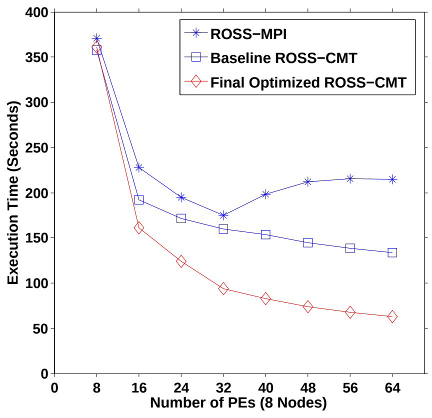
Figure 6(a) demonstrates the impact of each of these strategies with the hierarchical model described above. It shows that the benefit of LSP increases as the number of partitions increases. LSP performs 15% better than LOP in the case of 16 partitions while up to 44% better in the case of 64 partitions. In addition, MCIP optimization is orthogonal to partitioning strategies and significantly improves the performance both in the case of LSP and LOP. Figure 6(a) also indicates that LSP and MCIP optimizations when used together (LSP-MCIP) provide the best performance of these 4 strategies compared here. Figure 6(b) shows the percentage of remote messages across nodes out of the total messages among partitions in the case of LSP and LOP. LSP places the most communicating object groups on the same node as indicated by the significant reduction in the percentage of remote messages.

4.4 Scalability Analysis of PDES

The next experiment shows the scalability of the simulator as the number of hardware threads per node is increased (Figure 7(a) with remote percentage of 20% and Figure 7(b) with remote percentage of 80%). In particular, the simulation is performed on 8 nodes, and each PE is mapped to one hardware thread. The total number of PEs used on the x-axis is increasing as we increase the number of hardware threads used on each node. In the case of 8 PEs on the x-axis, the three versions have a similar performance with each other because they each have a single thread or a pro-

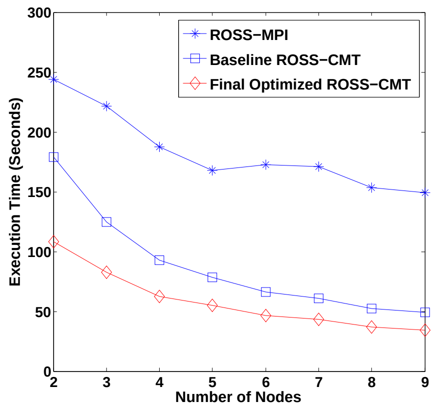


(a) 20% Remote Communication

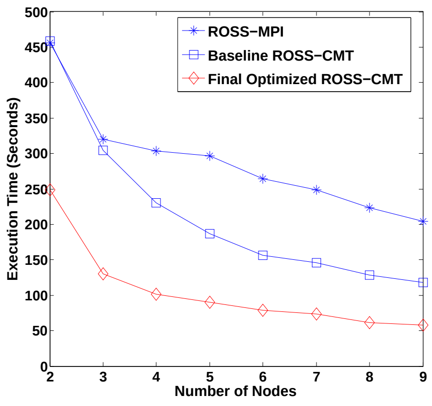


(b) 80% Remote Communication

Figure 7: PDES Scalability as Number of Hardware Threads per Node is Increased



(a) 20% Remote Communication



(b) 80% Remote Communication

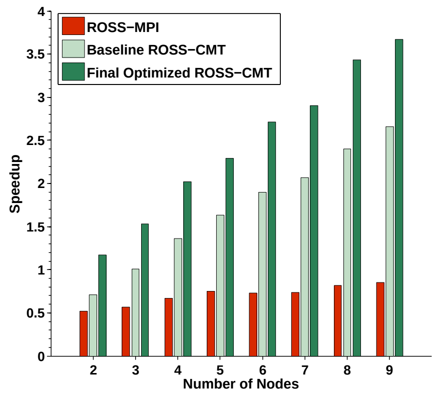
Figure 8: PDES Scalability for Different Number of Nodes

cess on each node communicating through MPI. However, as the number of hardware threads per node is increased, ROSS-CMT is able to take advantage of the more efficient communication among threads on the same node avoiding the use of MPI, achieving over 2.8X speedup over ROSS-MPI for a 64 way simulation at 20% remote percentage (the improvement is a more modest 1.5X in the case of 80% remote percentage).

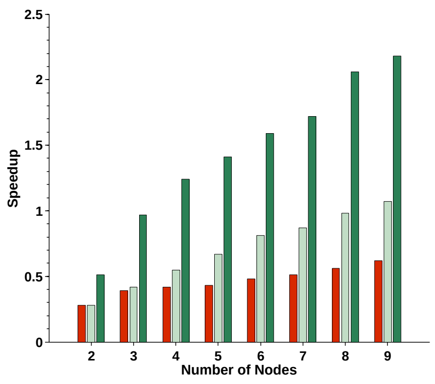
When message consolidation and infrequent polling are used, the optimized version of ROSS-CMT is able to scale much better, achieving about 4X improvement in performance over ROSS-MPI at both 20% and 80% remote event percentages. In addition, the performance of optimized ROSS-CMT exceeds that of baseline ROSS-CMT by a factor of 1.4X to 2X.

In the next experiment we show the scalability of the simulator as we increase the number of nodes used with a fixed number of hardware threads used per node (6 hardware threads). Figure 8(a) and Figure 8(b) show the scalability for optimistic simulation at 20% and 80% remote communication percentage respectively. We start with 2 nodes since it is impossible to create remote message traffic when only one node is used. Again, performance and scalability of the optimized ROSS-CMT are significantly better than the ROSS-MPI, achieving about 4X speedup at 80% remote message percentage and about 4.5X at 20% remote communication. In addition, in this experiment the performance of optimized ROSS-CMT exceeds that of baseline ROSS-CMT by a factor of up to 2X. Figure 9(a) and Figure 9(b) show the speedup of optimistic simulation against sequential simulation at both 20% and 80% remote communication percentages. Clearly, the optimized ROSS-CMT achieves better speedups than both baseline ROSS-CMT and ROSS-MPI. For example, at the case of 9 nodes, the speedup of optimized ROSS-CMT is 3.7X at 20% remote percentage, and 2.2 at 80% remote percentage.

In the next experiment we show the impact of Event Processing Granularity (EPG) for the scalability of PDES. EPG controls the amount of computation executed for each event. In our implementation the value of EPG determines the number of computation loops required for each event processing. A higher value of EPG makes the application more coarse-grained, and gives more computation load to the application. Note that most PDES models tend to require relatively small amounts of processing, typically to update state variables. However, it is possible to have models with significant event processing, and it is instructive to see how the ratio of computation to communication influences the per-

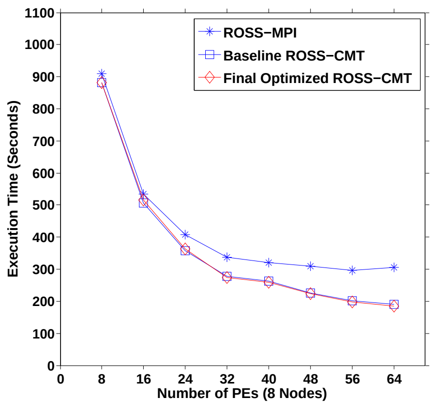


(a) 20% Remote Communication

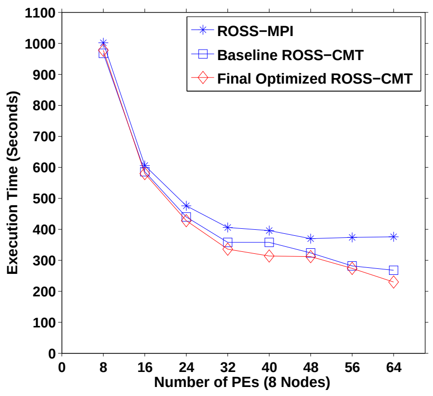


(b) 80% Remote Communication

Figure 9: Speedup of PDES Optimistic Simulation against Sequential Simulation



(a) 20% Remote Communication



(b) 80% Remote Communication

Figure 10: Impact of Event Processing Granularity (EPG=3000)

formance of PDES on CMs. Figure 10(a) and Figure 10(b) show the three versions of ROSS with EPG value of 3000 under 20% and 80% remote communication respectively. We find that the baseline ROSS-CMT performs close to optimized ROSS-CMT. This indicates that with a high value for EPG, the performance of ROSS-CMT is not dominated by communication.

In the next experiment, we evaluate PDES on a real model of a Personal Communication Services (PCS) system [8]. In this model, a cellular provider infrastructure is simulated as a number of mobile customers use it. A mobile phone call is simulated as an event, moved from one cell phone tower to another. Upon receiving a phone call, the cell phone tower assigns an available channel to the call. Once this phone call ends, the allocated channel is released. In addition, another phone call or more may be generated. If all channels are busy, the call is blocked. Moreover, if a call's connected portable is leaving the cell's area, then the call is handed-off to the destination cell phone tower [8]. In our experiments, the PCS simulation consists of 57600 cell phone towers (LPs). In addition, the number of channels per cell phone tower is fixed at 10. Figure 11(a) shows the PCS scalability of three versions of ROSS simulator as the number of hardware threads per node is increased. In this experiment, the number of nodes is fixed at 8. At the case of 64-way simulation, the optimized ROSS-CMT exceeds the baseline ROSS-CMT by a factor of 1.7, and 2 over ROSS-MPI. Figure 11(b) shows PCS scalability as the number of nodes is increased, with a fixed number of hardware threads used per node (6 hardware threads). Clearly, the optimized ROSS-CMT outperforms both the baseline ROSS-CMT and ROSS-MPI.

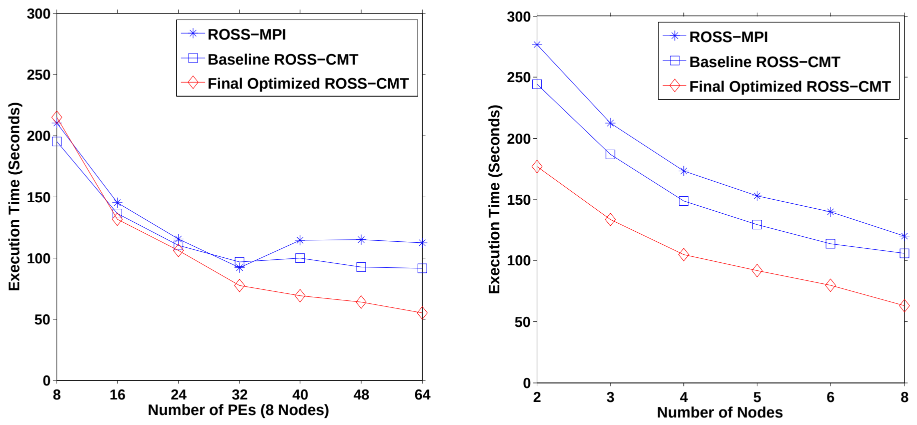
4.5 Experimental Results Summary

In summary, we evaluated the performance of ROSS-CMT with proposed optimizations on CMs with highly heterogeneous delays. We discovered that:

1. Message consolidation significantly improves the performance of ROSS-CMT PDES simulator. It provides a performance gain of about 3X in ROSS-CMT against ROSS-MPI at high percentage of remote communication. However, the overhead of unsuccessful probes can't be ignored, especially in the case of low percentage of remote communication.

2. Infrequent polling is capable of providing up to another 20% gain in performance of ROSS-CMT.

3. Latency-sensitive partitioning without any of above optimizations provides up to 44% performance improve-



(a) Different Number of Hardware Threads per Node

(b) Different Number of Nodes

Figure 11: Scalability Study of PCS Model

ment over latency-oblivious partitioning in the model we considered.

4. With the proposed optimizations the performance of the optimized ROSS-CMT exceeds that of non-optimized ROSS-CMT by a factor of about 2X, and that of ROSS-MPI by a factor of about 4.5X.

5. RELATED WORK

Communication latency is one of the traditional challenges to the performance and scalability of parallel applications. One of the approaches to reduce the impact of this problem is to develop fast network interconnection fabrics [24]. Multi-threaded implementations of algorithms [27, 14] on multi-core systems are being increasingly used due to the fast thread-based communication through shared memory on such systems. To scale to large size applications using commodity processing nodes, clusters of multi-cores must be used; however, high latency on the network becomes a potential bottleneck for the performance in this environment, especially in some communication-bound applications such as Parallel Discrete Event Simulation (PDES).

5.1 Parallel Discrete Event Simulation

It has been long-recognized that PDES suffers from communication overheads due to its fine-grained nature [7]. As a result, significant PDES research exists on optimizing the communication of PDES applications to improve their performance. Partitioning [13] plays an important role in reducing the communication latencies, by placing the objects with frequent communication on the same processor before the execution of the simulation. In [1], both communication and object activity were profiled first, and then used for a partition tool [20] to make partitions. However, static partitioning may not be efficient because the behavior of some models may change dynamically. Therefore, run-time object migration [4, 26] has been proposed to adjust the partitions at run time and to balance the workload of each processor as well. El-Khatib et. al. [15] provide three metrics to monitor the behavior changes of the model for optimistic simulation. The workload of each processor is evaluated periodically, and the workload migration occurs once the condition of dynamic adjustments is satisfied.

5.2 Optimizing the communication cost

For PDES applications, several techniques have been proposed at the simulation kernel communication system level to reduce either the cost or frequency of communication. Sharma et. al. [29] argue that too aggressive polling strategy for message delivery in asynchronous applications may harm the performance due to the overhead of unsuccessful probes. However, it is difficult to predict the optimal polling frequency at compile-time. Thus, they studied three heuristics for infrequent polling. Message aggregation is often used to reduce the frequency of the communication by grouping multiple messages into a single one before transmission. For example, a dynamic message aggregation approach is proposed in [11], which defines an aggregation window whose value can be assigned statically or dynamically, with the objective of controlling the behavior of the message aggregation in a Time-Warp simulator. Sharma et. al. [30] proposed a multi-threaded based Time Warp simulator on Clusters of SMPs (Symmetric Multi-Processors). While, conceptually, clusters of SMPs are similar to clusters of multi-cores, the tighter integration on multi-cores makes the environment significantly different. Moreover, other advances in processor architectures, communication libraries, and network fabric over the past 15 years substantially influence the relative latencies and overheads on the two environments.

For other types of applications, message aggregation is also widely used. For example, in the Simulink application, message aggregation is used to combine forwarding of multiple values between two threads into a single message [5]. [9] presents an identity-based aggregate signature algorithm in a vehicular ad-hoc network application, which aggregates signatures of multiple messages into one, for the purpose of security as well as performance efficiency.

The main difference between our proposal and traditional message aggregation approaches is: In message aggregation, messages from the same sender to the same receiver are consolidated to amortize overhead. In contrast, our proposed optimization combines messages from different senders to different receivers as long as these messages share the source and destination nodes. Thus, it exposes significantly higher opportunities for consolidation, and can significantly improve the performance of application. Traditional message aggregation often requires delaying messages to improve opportunities for aggregation; in the proposed approach, since messages are aggregated across multiple senders, there are ample opportunities for aggregation without delaying messages.

5.3 Application Study on Multi-cores

The multi-core architecture can substantially reduce the latency of the intra-machine communication due to its tightly integrated cores on a single chip. Thus, some researchers study the performance of applications on emerging multi-cores and multi-core clusters. Scalability studies of PDES on a blue gene supercomputer demonstrate that scalability up to thousands of processors can be achieved due to the low communication latency [22, 3]. In [21], a multi-process based PDES kernel is used to evaluate the performance on both multi-core clusters and traditional clusters, and shows that a better performance can be achieved on CMs. Wilsey et al. [12] proposed an algorithm to allow dynamic core frequency adjustment during Time Warp simulations on multi-cores. This approach can improve the performance of simulation, by overclocking the cores containing LPs with smaller rollbacks, as well as underclocking the cores having LPs with larger rollbacks.

In order to achieve better performance on multi-cores, multi-threaded implementations of PDES simulation engines are widely used. [19, 18] developed a multi-threaded PDES kernel and optimized for a standalone multi-core node, where each thread worked for a different simulation kernel instance. Chen et al. [10] proposed a different multi-threaded PDES simulation engine that employed a global event scheduling mechanism, where each thread selected the smallest time-stamped event from a global queue shared by other threads. However, these studies are limited to a single multi-core node. To support CMs, a multi-threaded MPI simulator was recently developed, however, very small performance gain was achieved compared with multi-process based simulator due to the impact of inter-node communication [33][2]. Liu et al. [25] developed a multi-threaded simulator, called *MiniSSF*, to support CMs. Different from our approach, two additional threads were created on each node to send and receive remote events respectively. In addition, a blocking receive operation was used to receive remote events. Vitali et al. [32, 31] proposed a load-sharing scheme developed in a multi-threaded PDES simulator, where the workloads of each simulation kernel instance were distributed across a dynamically changing set of threads.

6. CONCLUDING REMARKS

Parallel algorithms went through a phase where the impact of the communication topology was exposed to the programmer because of the high communication latency. Algorithms were developed in way specialized to the communication topology on the machine; it was common to have substantially different algorithms with different communication topologies. This situation changed substantially with the advent of high performance communication infrastructure where it became common to abstract communication as an ideal all-to-all medium, where messages were sent to destinations without regards to the underlying physical topology. In this paper, we argued that the high delay links necessitate a step back towards algorithms that are aware of the presence of these high delay links. We explored a set of such techniques and optimizations that can enable PDES application to tolerate the effect of high latency links.

In particular, our goal was to answer the following question: in the presence of highly heterogeneous delays on CMs, do fine-grained applications such as PDES benefit from the low latency between cores on a single machine, or are they limited by the communication across machines with relatively higher delay? We illustrate this problem by using ROSS-CMT PDES simulator. We find that on CMs, for ROSS-CMT, the network connections impact its performance and scalability significantly.

We proposed three optimizations to reduce the impact of communication across machines: consolidated message routing, infrequent polling, and partitioning aware of the heterogeneous latency. We discover that the performance of optimized ROSS-CMT with classical Phold model achieves a 2X speedup against non-optimized ROSS-CMT, and 4.5X speedup against the original multi-process ROSS simulator. Latency-sensitive partitioning can provide up to 44% performance improvement against latency-oblivious partitioning on CMs.

Acknowledgements

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-11-2-0004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. We also gratefully acknowledge support from the National Science Foundation grants CNS-0916323 and CNS-0958501.

7. REFERENCES

[1] K. Bahulkar, J. Wang, N. Abu-Ghazaleh, and D. Ponomarev. Partitioning on dynamic behavior for parallel discrete event simulation. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 221–230. IEEE, 2012.

[2] M. L. Bailey, J. V. Briner, Jr., and R. D. Chamberlain. Parallel logic simulation of VLSI systems. *ACM Computing Surveys*, 26(3):255–294, sep 1994.

[3] D. Bauer, C. Carothers, and A. Holder. Scalable time warp on bluegene supercomputer. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 35–44, 2009.

[4] A. Boukerche and S. Das. Dynamic load balancing strategies for conservative parallel simulation. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 32–37, 1997.

[5] A. Canedo, T. Yoshizawa, and H.Komatsu. Automatic

[2]Note that this is a poster that used a preliminary version of the simulator without characterizing PDES behavior on CMs, and without any of the optimizations discussed in this paper.

parallelization of simulink applications. In *Proc. of CGO*, pages 151–159, 2010.

[6] C. Carothers, D. Bauer, and S. Pearce. ROSS: A high-performance, low memory, modular time warp system. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 53–60. IEEE, 2000.

[7] C. D. Carothers, R. M. Fujimoto, and P. England. Effect of communication overheads on Time Warp performance: An experimental study. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 118–125, jul 1994.

[8] C. D. Carothers, R. M. Fujimoto, and Y.-B. Lin. A case study in simulating pcs networks using time warp. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 87–94. IEEE, 1995.

[9] C. Chen, J. Zhang, R. Cohen, and P.Ho. Secure and efficient trust opinion aggregation for vehicular ad-hoc networks. In *Proc. of VTC*, pages 1–5, 2010.

[10] L. Chen, Y. Lu, Y. Yao, S. Peng, and L. Wu. A well-balanced time warp system on multi-core environments. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 1–9. IEEE, 2011.

[11] M. Chetlur, N. Abu-Ghazaleh, R. Radhakrishnan, and P. A. Wilsey. Optimizing communication in Time-Warp simulators. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 64–71. IEEE, 1998.

[12] R. Child and P. Wilsey. Dynamically adjusting core frequencies to accelerate time warp simulations in many-core processors. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 35–43. IEEE, 2012.

[13] J. Cloutier. Model partitioning and the performance of distributed timewarp simulation of logic circuits. *Simulation Practice and Theory*, 5(1):83–99, 1997.

[14] J. Doi and Y. Negishi. Overlapping methods of all-to-all communication and FFT algorithms for torus-connected massively parallel supercomputers. In *Proc. of Int'l Conference on Supercomputing*, pages 1–9, 2010.

[15] K. El-Khatib and C. Tropper. On metrics for the dynamic load balancing of optimistic simulations. In *Proc. 32nd Hawaii International Conference on Systems Science (HICCS)*, 1999.

[16] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, oct 1990.

[17] R. Fujimoto. Performance of time warp under synthetic workloads. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):23–28, 1990.

[18] D. Jagtap, K. Bahulkar, D.Ponomarev, and N.Abu-Ghazaleh. Characterizing and understanding pdes behavior on tilera architecture. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 53–62. IEEE, 2012.

[19] D. Jagtap, N.Abu-Ghazaleh, and D.Ponomarev. Optimization of parallel discrete event simulator for multi-core systems. In *Parallel and Distributed Processing Symposium (IPDPS)*, pages 520–531. IEEE, 2012.

[20] G. Karypis and V. Kumar. hmetis: a hypergraph partitioning package. Available on WWW at URL: http://www.cs.umn.edu/ karypis/metis/hmetis.

[21] K.Bahulkar, N.Hofmann, D.Jagtap, N.Abu-Ghazaleh, and D.Ponomarev. Performance evaluation of pdes on multicore clusters. In *14th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*, pages 131–140, 2010.

[22] K.S.Perumalla. Scaling time warp-based discrete event execution to 104 processors on a blue gene supercomputer. In *in Proceedings of the ACM Computing Frontiers*, pages 69–76, 2007.

[23] L. Li and C. Tropper. A design-driven partitioning algorithm for distributed verilog simulation. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 211–218. IEEE, 2007.

[24] J. Liu, B. chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. Panda. Performance comparison of mpi implementations over infiniband, myrinet and quadrics. In *Proc. of ACM/IEEE conference on Supercomputing*, pages 58–71. IEEE, nov 2003.

[25] J. Liu and R. Rong. Hierarchical composite synchronization. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 3–12. IEEE, 2012.

[26] P. Peschlow, T. Honecker, and P. Martini. A flexible dynamic partitioning algorithm for optimistic distributed simulation. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 219–228. IEEE, 2007.

[27] R. Preissl, N. Wichmann, B. Long, J. Shalf, S. Ethier, and A. Koniges. Multithreaded global address space communication techniques for gyrokinetic fusion applications on ultra-scale platforms. In *Proc. of Int'l Conference on Supercomputing*, 2011.

[28] V. Sarkar and J. Hennessy. Compile-time partitioning and scheduling of parallel programs. In *Proc. of the SIGPLAN Symposium on Compiler construction*, pages 17–26, 1986.

[29] G. D. Sharma, N. B. Abu-Ghazaleh, U. V. Rajasekaran, and P. A. Wilsey. Optimizing message delivery in asynchronous distributed applications. In *Proc. of Euro-Par*, pages 1204–1208, 1998.

[30] G. D. Sharma, R. Radhakrishnan, U. V. Rajesekaran, N. B. Abu-Ghazaleh, and P. A. Wilsey. Time warp simulation on clumps. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 174–181, may 1999.

[31] R. Vitali, A. Pellegrini, and F. Quaglia. Assessing load-sharing within optimistic simulation platforms. In *Proceedings of the 2012 Winter Simulation Conference*. IEEE, 2012.

[32] R. Vitali, A. Pellegrini, and F. Quaglia. Towards symmetric multi-threaded optimistic simulation kernels. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 211–220. IEEE, 2012.

[33] J. Wang, D.Ponomarev, and N.Abu-Ghazaleh. Performance analysis of a multithreaded pdes simulator on multicore clusters. In *Principles of Advanced and Distributed Simulation (PADS) (Short Paper)*, pages 93–95. IEEE, 2012.