Performance Analysis of a Multithreaded PDES Simulator on Multicore Clusters

Jingjing Wang, Dmitry Ponomarev, and Nael Abu-Ghazaleh
Computer Science Department
State University of New York at Binghamton
Binghamton, NY 13902, U.S.A
{jwang36,dima,nael@cs.binghamton.edu}

Abstract—**The performance of fine-grained applications such as Parallel Discrete Event Simulation (PDES) is limited by communication overheads. Multi-core architectures with tightly integrated cores on a single chip can substantially reduce the communication cost. The number of cores available on such machines remains low. To scale the simulation, it is important to be able to effectively use Clusters of Multicores (CMs). However, the high communication overheads between remote machines can significantly limit scalability. It is unclear if, in the presence of relatively slow links, there is a benefit of having the low latency between the cores on the same machine. In this paper, we first extended a multithreaded PDES simulator to support CMs. In addition, we show that remote communication forms a primary challenge to scalability due to both latency and message processing software overheads.**

I. Introduction

Discrete Event Simulation (DES) is a key application used in the design and analysis of systems in which the state changes are discrete [1]. Parallel Discrete Event Simulation (PDES) [2], [3], [4] is used to improve the capacity and performance of DES by leveraging parallel processing. However, PDES applications suffer high overheads from communication.

Today, clusters of multicores (CMs) are widely used to reach higher scales of applications. In such environment, cores on the same machine communicate through low latency shared memory (or other intra-chip communication), while message passing is used for the communication between cores on different machines [5]. While PDES simulations executing on a single multicore exhibit very good performance, it is unclear whether this benefit can carry over to CMs, where some of the links between cores have substantially higher latency and message communication overhead.

To answer this question, we use as our experimental basis, the Rensselaers Optimistic Simulation System (ROSS) [6]: a state of the art PDES simulator that employs message passing for communication using MPICH [7]. To achieve a better performance, a multithreaded version of ROSS (ROSS-MT) was developed for use on a single multi-core machine in our previous work [8]. In order to investigate the impact of the heterogeneous communication latency (intra vs. inter-core) on the performance and scalability of PDES in CM environments, we extend ROSS-MT to support CMs by using MPICH to communicate between cores on different machines; we call this version CM Multi-threaded ROSS (ROSS-CMT). The MPICH libraries are used for the communication across the network. However, the MPICH libraries are not thread safe [9], and access to them is serialized, incurring high overhead when more than one thread invokes MPI functions simultaneously. Our experiments show that when we allow each thread to use MPI for its remote messages, the overall performance is worse than the baseline process-based ROSS (ROSS-MPI). To address this issue, we set aside one thread on each machine to perform communication across the network. It takes messages generated by the simulation threads and sends them to their destinations. It also receives remote messages and forwards to the destination thread.

We first use ROSS-CMT to investigate the impact of the heterogeneous communication latency (intra vs. inter-core) on the performance and scalability of PDES in CM environments. We show that the higher latency causes a higher percentage of rollbacks for optimistic simulation. In addition, we show that ROSS-CMT achieves a better performance than MPI-based version on multi-core clusters.

II. Design Overview of ROSS-CMT

In ROSS, events are classified into three types based on the relationship between the originating and target PEs: (1) a *local event* is one where the sending PE is itself the target; (2) a *regional event* refers to events sent to another PE on the same machine; and (3) a *remote event* is generated from a PE to another on a different machine.

Figure 1 shows the communication mechanism in ROSS-CMT on a cluster of two machines with two threads each. Consider the case where thread 2 on the first machine has a regional event targeted to thread 1. This event, shown with dashed arrows (tagged with number 1) in Figure 1, is communicated using an insertion of a pointer to a copy of the event into the input queue of thread 1. Later thread 1 queues this event into its event queue (tagged with 2 on the figure).

The solid arrows in Figure 1 show the communication of a remote event originating from thread 2 on the first machine

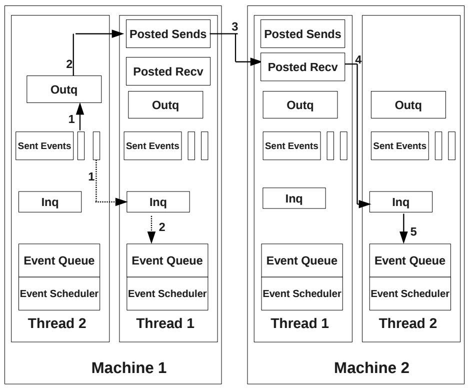

Figure 1. ROSS-CMT Communication Mechanism

to thread 2 on the second machine. The remote event is queued into the output queue of the sending thread. Thread 1 (the communication thread) later looks up this output queue, and sends the event to thread 1 of the second machine. Once thread 1 of the second machine probes (or polls) this event successfully, it then inserts the pointer of this event into the input queue of thread 2. For a situation where more than two threads exist on each machine, the communication thread looks up the output queue of each thread in round robin fashion.

III. Performance Evaluation

In the experiments we use *Phold* benchmark [10] to evaluate the performance of both ROSS-CMT and ROSS-MPI on a cluster of 4-core Intel core i7 machines. Each machine has 4 cores, but with hyperthreading, can execute up to 8 concurrent threads. The machines are connected through a Gigabit Ethernet switch.

A. *Rollbacks caused by Heterogeneous Latency*

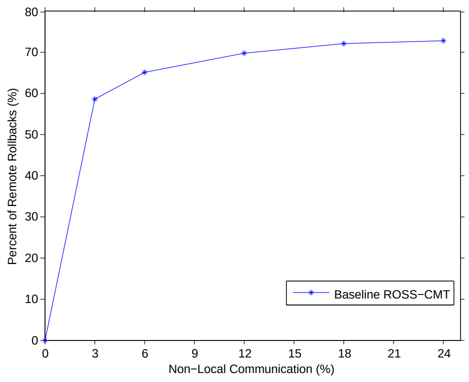

Figure 2. The Percentage of Rollbacks caused by remote events

Figure 2 shows the percentage of rollbacks caused by remote events as a function of the percentage of these remote events. Since these events incur significant latency, they are often late by the time they arrive, causing a rollback. Clearly, a disproportionate percentage of rollbacks are caused by remote events; even at 2% remote event percentage, over 50% of the rollbacks are caused by remote events. However, addressing latency at the simulation kernel level is difficult since it is a problem of the underlying network protocol stack and network hardware.

B. *Performance of ROSS-CMT vs. ROSS-MPI under different percentages of non-local communication*

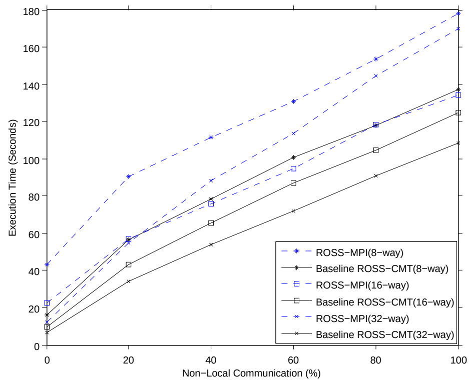

Figure 3. Performance of ROSS-CMT and ROSS-MPI

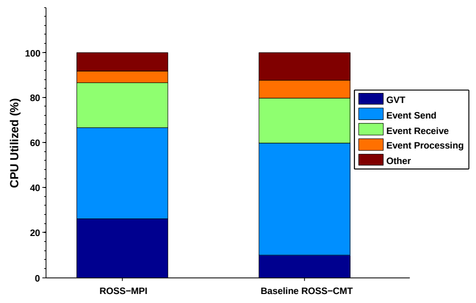

Figure 4. CPU Usage of Baseline ROSS-CMT vs. ROSS-MPI under 32-way simulation (100% non-local communication)

Figure 3 shows the performance of ROSS-CMT vs. ROSS-MPI under 8-way, 16-way and 32-way simulation on 4 machines, with 2 cores, 4 cores and 8 cores each machine respectively. The x-axis shows the percentage of non-local communication, which is the sum of both regional (events to cores on the same machine) and remote (to cores on other machines) events. At 0% non-local communication, the simulation run-time is low. However, as the remote communication increases, even modestly, the execution time increases substantially. For the 32-way simulation, at 50% non-local communication (or 25% remote communication), the simulation runs 5 time slower than the 0% remote communication case for both ROSS-CMT and ROSS-MPI. At 100% non-local communication, it is an order of magnitude slower than the case with 0% remote communication. Clearly, the impact of network communication is significant.

It is worth it to note that the performance gain from ROSS-CMT over ROSS-MPI increases as the non-local communication percentage increases, which allows more regional events to take advantage of the faster direct communication available in ROSS-CMT.

Figure 4 shows the CPU usage of each stage in the 32-way simulation with 100% non-local event percentage (50% regional and 50% remote) for both ROSS-CMT (the communication thread) and ROSS-MPI. In ROSS-CMT, a smaller percentage of time is spent on GVT computation (10% compared to 26.1% for MPI), but more time is spent on sending events (49.8% compared to 40.55%). In addition, ROSS-CMT and ROSS-MPI have a similar percentage of time spent on event processing (7.7% for ROSS-CMT compared to 5.2% for ROSS-MPI). It is important to consider optimizations that can reduce this overhead.

C. Scalability Analysis

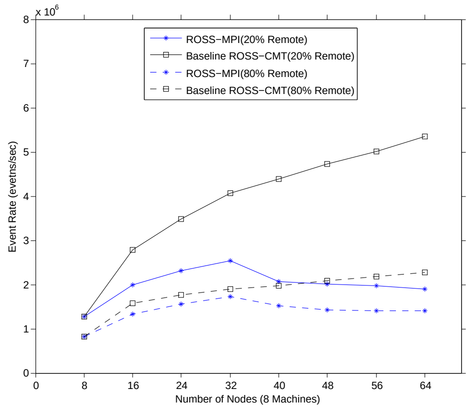

Figure 5. Scalability as number of cores used per machine is increased

Figure 5 shows the scalability of the simulator with remote percentage of 20% and 80% (communication across the network). In particular, the simulation is performed on 8 machines, and we increase the number of cores used on each machine increasing the total number of cores used on the x-axis. When only a single core is used (8 cores on the x-axis), the two versions perform close to each other since they each have a single thread per process communicating through MPI. However, as the number of cores per machine is increased, ROSS-CMT is able to take advantage of the more efficient communication among threads on the same machine avoiding the use of MPI for those events.

IV. Concluding Remarks and Future Work

Heterogeneous delays exist in CMs: communication on the same machine is fast, but communication across machines incurs both high overhead and latency. It is unclear if in the presence of such slow/high overhead links, the low-latency on each single machine significantly improves the scalability of PDES. To answer this question, we extended a multi-threaded version of the ROSS simulator to work in CM environments. We discover that: (1) There is some benefit from multi-cores even within a cluster environment compared with the MPI version; however (2) the network connections significantly impact the performance relative to a situation where all links are of low latency. For our future work, we will explore some optimizations to provide significant improvement in the scalability of the ROSS-CMT simulator in CM environments.

Acknowledgements

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-11-2-0004 and by National Science Foundation grants CNS-0916323 and CNS-0958501. 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 and endorsements, either expressed or implied, of Air Force Research Laboratory, National Science Foundation, or the U.S. Government.

References

[1] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed. McGraw-Hill, 2000.

[2] R. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.

[3] J. Misra, "Distributed discrete-event simulation," *Computing Surveys*, vol. 18, no. 1, pp. 39–65, Mar. 1986.

[4] P. F. Reynolds Jr., "A spectrum of options for parallel simulation," in *Winter Simulation Conference*. Society for Computer Simulation, 1988, pp. 325–332.

[5] K.Asanovic, R.Bodik, J.Demmel, J.Kubiatowicz, K.Keutzer, E.Lee, G.Necula, D.Patterson, K.Sen, J.Shalf, J.Wawrzynek, and K.Yelick, "The landscape of parallel computing research: A view from berkeley 2.0," Jun. 2007, presentation slides available at http://view.ececs.berkeley.edu.

[6] C. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low memory, modular time warp system," in *Proc of PADS*, 2000.

[7] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA: MIT Press, 1994.

[8] D. Jagtap, N.Abu-Ghazaleh, and D.Ponomarev, "Optimization of parallel discrete event simulator for multi-core systems," in *Proc. of IPDPS*, 2012.

[9] G. D. Sharma, R. Radhakrishnan, U. V. Rajesekaran, N. B. Abu-Ghazaleh, and P. A. Wilsey, "Time warp simulation on clumps," in *Proc of PADS*, May 1999.

[10] R. Fujimoto, "Performance of time warp under synthetic workloads," *Proceedings of the SCS Multiconference on Distributed Simulation*, vol. 22, no. 1, pp. 23–28, Jan. 1990.