Performance Evaluation of PDES on Multi-Core Clusters

Ketan Bahulkar, Nicole Hofmann, Deepak Jagtap, Nael Abu-Ghazaleh and Dmitry Ponomarev
Computer Science Department
State University of New York at Binghamton
{kbahulkar, nhofman1, djagtap1, nael, dima}@cs.binghamton.edu

***Abstract*—Trends in VLSI and microarchitecture design have ushered in the multi-core era, where the number of cores on a chip is expected to grow with every processor generation. Soon, each chip will have a large number of tightly integrated processing cores with communication latencies substantially lower than those present in conventional clusters. Clusters made of such microprocessors experience non-uniform latencies between cores: cores on the same chip can communicate faster than cores on different chips; cores on the same machine can communicate faster than cores on different machines. In this paper, we characterize the performance of PDES models on a cluster of dual quad-core machines using a parameterizable modified version of *Phold*, a standard benchmark for parallel simulation.**

We study various combinations of regional and remote communication patterns to quantify the impact of communication on overall performance of simulation. We discover that the amount of communication has determining impact and it's essential to optimize this communication at each level to take maximum advantage of multi-core platform. We show that partitioning significantly improves performance. We also explore the impact of load imbalance on application performance and provide critical insight into how to partition for these different environments. We believe that this study represents a significant first step in characterizing the performance space for PDES on this emerging platform.

I. Introduction

The performance of modern microprocessors is no longer following Moore's Law as the increasing power consumption with every new VLSI generation limits future improvement in clock speed. As a result, computer architects have turned to multi-core designs, as a potential approach to continue to scale microprocessor performance. The so-called new Moore's Law forecasts that the number of cores per chip will double every 2 years [1]. If this holds true, multi-core machines will soon evolve to manycores, with 10s if not 100s of cores per chip, as permitted by power budgets.

This paradigm shift in microprocessor architecture impacts computing clusters since they are built from these commodity CPUs. As a result, we have entered the era of Multi-core clusters (MCCs): clusters where each machine is made up of one or more multi-core CPUs. As the number of cores per CPU increases, MCCs represent a significantly different environment from traditional clusters. The communication delays between cores on the same chip are substantially lower than those experienced among machines connected by a network. This lower latency can provide a dramatic boost to the performance of fine-grained applications, such as Parallel Discrete Event Simulation (PDES), especially when communication remains within a multi-core chip.

The contribution of this paper is to study the performance of PDES in a cluster of multi-core machines in comparison to the cluster of traditional single core machines. We focus our study on the impact of communication and load imbalance at inter-machine and intra-machine levels. We also present our observations for other factors such as event processing granularity and impact of memory limitations. We provide results for different combinations of multi-core cluster configurations. On the basis of our observations we propose multi-level partitioning for optimizing communication at various levels and present a basic study to support our observations. To carry out our experiments, we augment WarpIV with the capability to perform manual object decomposition across the simulation nodes, evolve the classic *Phold* benchmark enabling it to represent a range of application behaviors and finally incorporate a topology component into this benchmark to enable hierarchically clustered communication patterns.

The remainder of the paper is organized as follows. In Section II we will cover the basics of Time Warp Simulation and Multi-core Cluster environment and the benchmark. Section III presents the motivations behind this study. In Section IV we present the performance evaluation of PDES on MCCs, including the effect of communication latencies, load imbalance, as well as the implications of partitioning and memory impact. Section V presents the related work. In Section VI we offer our concluding remarks.

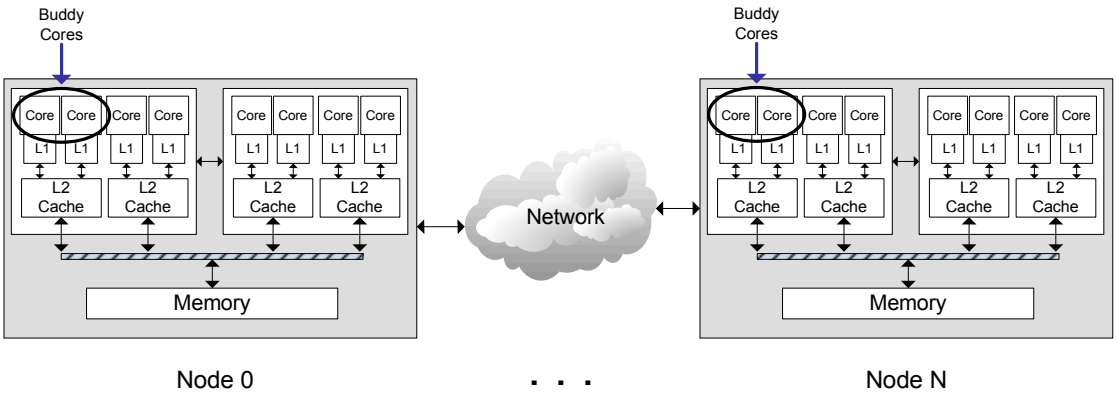


Figure 1: Architecture of a Typical Dual Quad-core Cluster

II. Background

In this section, we first briefly describe optimistic parallel discrete event simulation and the WarpIV simulator [2], overview a typical multi-core cluster organization and describe our benchmark.

A. Optimistic Parallel Discrete Event Simulation and WarpIV

A Parallel Discrete-Event Simulation (PDES) is organized as a collection of Logical Processes (LPs) that communicate by exchanging time-stamped event messages [3], [4]. Each LP processes its events in time stamp order (to ensure causality). PDES simulators differ in the synchronization algorithm used to ensure correct event ordering among events on different LPs. The LPs in conservative simulators exchange messages to upgrade each other of their progress and guarantee correctness. Alternatively, optimistic simulation may be used where LPs process events with no explicit synchronization occurring among them. Causality is preserved among different processes by exchanging time-stamped event messages and using rollback upon receiving a message with a time in the past. Thus, the state of the simulation must be saved to allow rollbacks when a causality breach is detected. The progress of the simulation (the Global Virtual Time, or GVT) is computed as the minimum of the timestamps of all LPs as well as messages in transit. GVT is used to garbage collect state information, commit output events and often to adapt configuration parameters of the simulation. When a rollback occurs, the state of the simulation is restored to a valid state before the rollback time. Any messages erroneously sent to other LPs must be cancelled by sending anti-messages. Cascading rollbacks can occur when a rollback at one node causes a sequence of rollbacks at other nodes as it sends out it's anti-messages. Cascading rollbacks significantly harm performance. For this reason, some simulators throttle the unchecked simulation progress to reduce the chance of cascading rollbacks. More frequent communication, or higher communication latencies can lead to rollbacks and cascading rollbacks and delays in computing GVT resulting in larger memory footprint, and slower overall execution and so communication frequency and latency play a major role in determining the performance of simulation [5].

In our experiments, we use the WarpIV [2] simulator which is the successor to the SPEEDES simulator [6]. WarpIV supports conservative, optimistic and throttled optimistic synchronization protocols. To perform experiments with object partitioning across the simulation nodes and study the performance impact of partitioning on MCC, we augmented WarpIV with the capability to perform object placement based on the user-specified assignments.

B. Multi-core Clusters

Figure 1 shows the typical organization of a multi-core cluster. On a single chip, two cores share the L2 cache and have a much lower communication latency than the other two cores on the chip (in a quad-core Xeon architecture). Delays between cores on different chips, but on the same machine are likely to be higher. Finally, the latency of communication to cores that are on different machines is significantly higher as this communication has to travel much longer distances, and

due to the overhead of switching to the operating system and traversing the network protocol stack.

In our experiments we use a gigabit Ethernet cluster where each node has 2 quad-core Intel Xeon chips. The Traditional Cluster in our experiments refers to the use of single core on each machine. Multi-core Cluster (MCC) refers to 6 cores per machine unless otherwise noted.

C. Benchmark

To capture wide range of application characteristics we developed a synthetic, controllable benchmark that is a variant of the classical *Phold* benchmark. Classic *Phold* simulation model consists of number of logical processes where each logical process has number of objects. Objects are initialized with specified number of events each. Each object picks another object and sends an event. total event population is preserved in Phold model.In particular, we modified *Phold* to allow control of the following parameters: (1) the communication pattern to control the breakdown of the communication to the different classes of cores (2) the processing granularity; and (3) the degree of load imbalance.

In our benchmark the cross-objects communications can be of three forms: 1) *Local communication* when the receiving object is located on the same core as the sending objects; 2) *Regional communication* when the sending and the receiving objects are located on the same machine, but on different cores; and 3) *Remote communication* when two objects are located on different machines and communication requires traversing a network.

To simulate hierarchical clustered communication patterns, we integrated a configurable topology component in our benchmark. For example, we can create a 3-level hierarchical topology with 300 objects by specifying: Local Cluster size 10 (60% events), Regional Cluster size 5 (30% events), Remote Cluster size 2 (10% events). This configuration will result in total of 30 local clusters on lowermost level each with 10 objects, 5 local clusters together form a cluster on second level and 2 regional clusters together form third level clusters. During execution each object will send a local event 60 times by choosing a target object from it's local cluster, a regional event 30 times by choosing a target from another second-level cluster and a remote event 10 times by choosing a target from another third-level cluster to create a hierarchy of communication as specified by the configuration.

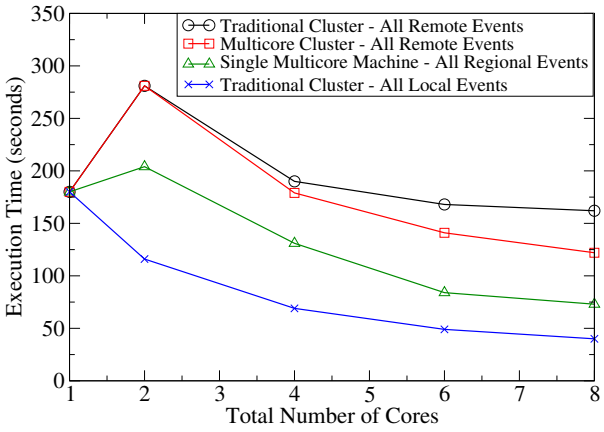


Figure 2: Optimization Space for Remote Communication

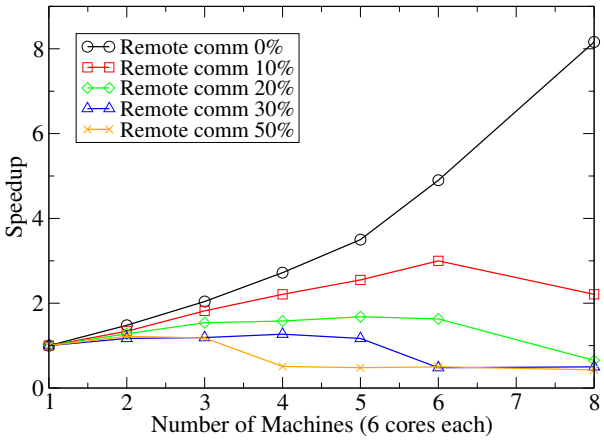


Figure 3: Multicore Cluster: Speedup

III. Motivation

Figure 2 shows the performance of PDES on equivalent Traditional, Multi-core Cluster and Single Multi-Core machine platforms. In Traditional Cluster, each machine has only a single core. Multicore Cluster in this experiment consists of 2 multi-core machines with the total number of cores evenly distributed among these machines.

The top three lines on this graph show the performance of these three platforms when all communications occur between different cores (i.e., there are no local events). For traditional clusters, this means that all communications occur across different machines. On a multi-core cluster, some of the remote events now become regional (due to grouping of multiple cores) and are thus scheduled on the same machine. In case of a single multi-core machine, all the cores reside on the

same machine and so all communication is regional.

The execution time drops from 160 seconds to 120 seconds when we move from a traditional cluster of 8 machines to a MCC of 2 machines with 4 cores each. It further drops to 70 seconds when a single multi-core machine with 8 cores is used. The purpose of these graphs is not just to show that the performance progressively improves (which is obvious), but to demonstrate the range of possible performance gains that can be achieved by converting remote communications into regional.

The bottom line (Traditional Cluster - All Local Events) shows the performance when all events were scheduled locally and no events were sent across the cores. This is an ideal scenario, as it distributes the workload evenly across different machines (each with its private last-level cache and memory), but does not incur the price of network communications to achieve it. It demonstrates the potential for additional performance gains if regional communications can somehow be turned into local.

To further emphasize the impact of remote communications, Figure 3 shows the possible speedup on MCC for various percentages of remote communications. At high percentage of remote communication, the MCC provides limited speedup, but the situation improves as the percentage of remote communications is reduced. At 10% remote communication, MCC is capable of providing a speedup of 3x to 2x while at 50% remote communication speedup is reduced to 0.3x.

PDES simulations are fine-grained applications and therefore traditional PDES systems are mainly focused on hiding the latency impact. As demonstrated by the above experiments, the emergence of MCCs somewhat alleviates the communication bottlenecks. We wish to investigate the following question: Is it important to reconsider and optimize the design of other aspects of the simulation with MCCs, or the latency still remains an overwhelming bottleneck and everything else is still of secondary importance, just as with traditional cluster computing environments? We attempt to address this question in the following sections.

IV. Performance Evaluation of MCCs

PDES is a layered system, consisting of an application model, simulation kernel and the underlying hardware platform. Parameters at each of these different levels and dependencies between them contribute towards the performance. Background section provides the details of the WarpIV simulation engine we used in our experiments, the Clustered *Phold* benchmark and the typical organization of our Multicore Cluster. In this section, we study the impact of communication patterns on different configurations of these platforms. We then study the impact of load imbalance. Finally, we present a partitioning study and possible impact of memory limitations of multi-core cluster.

A. Effect of Communication

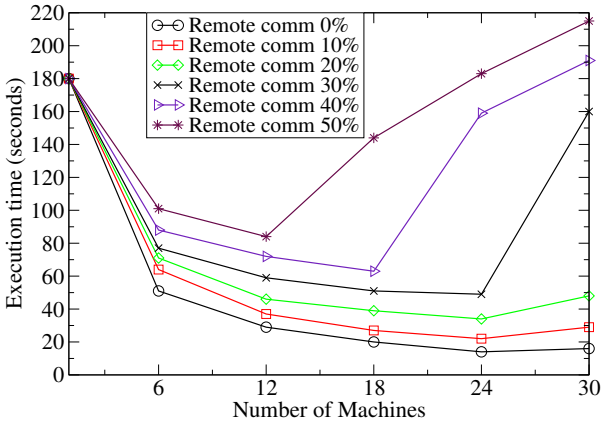
We start our experiments with presenting the impact of remote communication on Traditional Cluster and MCC platforms. Then we study the impact of regional communication on MCC and single multi-core machine. These communication patterns were created using the parameters in our Clustered *Phold* benchmark.

Impact of Remote Communication

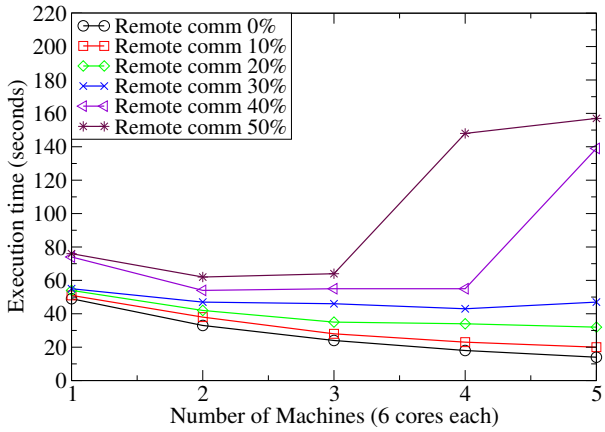
Figure 4(a) shows the scalability of traditional clusters (clusters with one core per machine), for different percentages of remote events. As the percentage of remote events increases, simulation scales up to lesser number of machines. For example, the performance continues to scale up to 24 machines when remote communication accounts for 30% of all messages, it scales up to 18 machines for 40% and only to 12 machines when 50% of all communications are remote.

The use of MCC instead of traditional clusters allows the reduction in remote communication percentage, because some of the remote communications now become regional (assuming the same simulation model). Intuitively, this reduction should lead to a better scalability. Figure 4(b) shows that the performance of MCC continues to scale up to 4 machines (6 cores each) for 40% remote communication, it scales up to 3 machines (6 cores each) for 50%. We can observe that even though the simulation performance scales better with MCC than with traditional cluster, the scalability limits are still reached relatively fast, especially for the scenarios with large percentage of remote communications.

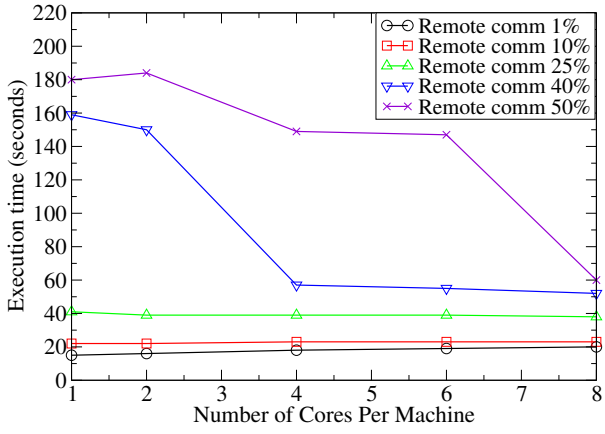
To gain a better understanding of the impact of remote communication on various combinations of MCC configurations, we analyzed 5 possible variations of 24-way simulation. These are as follows: 1) 24 machines with 1 core each; 2) 12 machines with 2 cores each; 3) 6 machines with 4 cores each; 4) 4 machines with 6 cores each; and 5) 3 machines with 8 cores each. Results from figure 4(c) show that when the percentage of remote communications is higher than 25%, the execution time decreases with increasing number of cores per machine. This clearly indicates the advantage of regional communication over remote communication. Also note that if remote communication is less (below 25%) then there is no significant change in execution time in-spite of increase



(a) Traditional Cluster

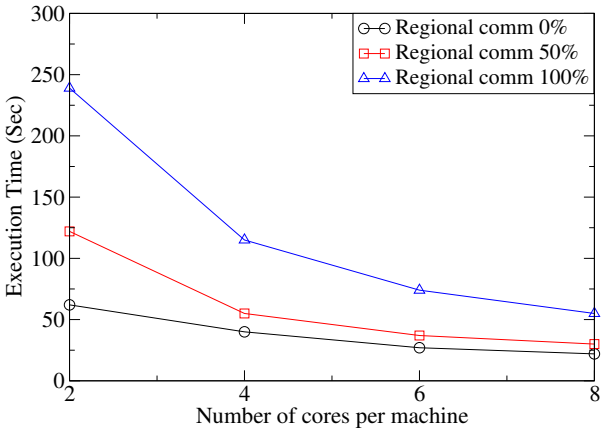


(b) Multicore Cluster

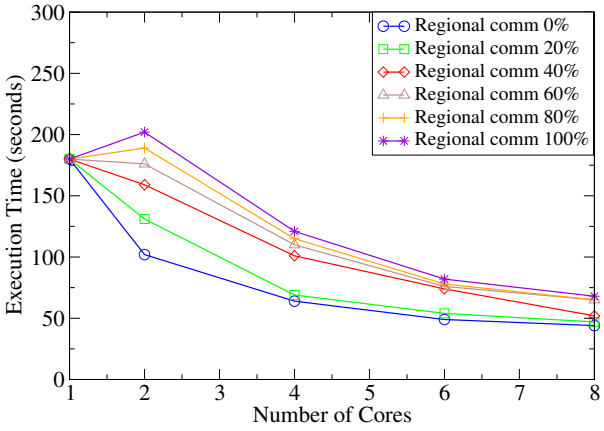


(c) Various MCC configurations (total 24 cores)

Figure 4: Impact of Remote Communication



(a) Multicore Cluster(4 machines)



(b) Single Multi-core machine

Figure 5: Impact of Regional Communication

in the number of cores per machine.

Impact of Regional Communication

Figure 5(a) shows the impact of regional communication on a MCC of 4 machines with increasing number of cores per machine. Results show that regional communication also has noticeable impact on performance. Using more cores per machine can help alleviate the impact of regional communication as the communicating objects are spread across multiple cores. This can result in smaller event queues on each core resulting in faster communication.

In case of MCCs, remote communication plays a major role but in case of single multi-core machine, remote communications is not present and so regional communication becomes a key factor affecting the performance. Figure 5(b) shows the performance on a single multi-core machine with increasing percentage of regional communication. Even though regional communications

do not have as high an impact on scalability and performance as remote communications, there is still a significant room to improve performance if the regional communication is minimized (by using object partitioning schemes, for example). As seen from the figure, the gap between highest and lowest execution times is larger at smaller number of cores which indicates that this optimization becomes even more beneficial if fewer cores are used on each machine.

B. Performance Effect of Load Imbalance

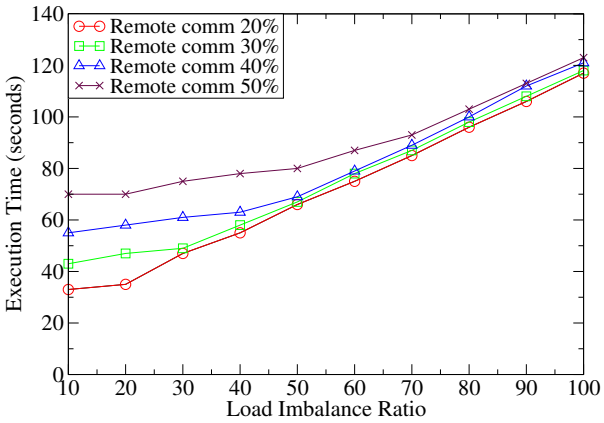
In our Clustered *Phold* benchmark, a new parameter called Load Imbalance Ratio was added to control the degree of load imbalance in the model. A higher value of Load Imbalance Ratio results in higher variation in the distribution of computational loads among the processing cores. Figure 6(b) shows that on MCC, a model with 20% remote communications is tolerant to the load imbalance of 20 (that is, processing delay at the slowest node is up to 20 times higher than the processing delay at the fastest node), while a model with 50% remote communications is tolerant to the load imbalance ratio of 30. In general, models with low percentage of remote communications are more sensitive to the load imbalance, as the communication overhead becomes less dominant and other factors (such as the load imbalance) start playing a larger role in the simulation performance.

In contrast to the trends observed for the MCCs, the effect of load imbalance is significant on a single multicore machine. As shown in Figure 6(c), even a small load imbalance has significant impact on performance even at high regional communication and results in a linear increase in execution time as the load imbalance increases.

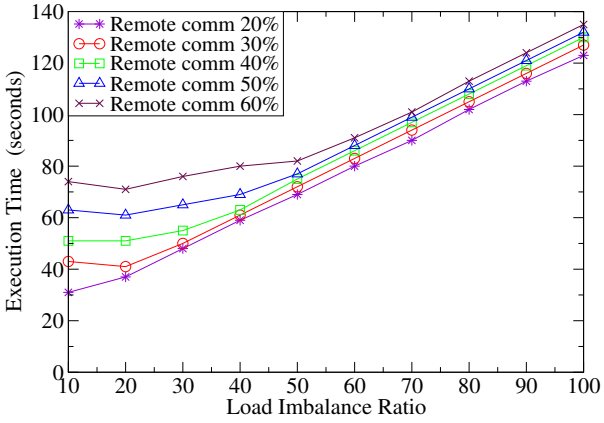
We also studied the impact of load imbalance for various MCC configurations. Figures 7(a) and 7(b) show load imbalance at various percentages of remote communications. The results demonstrate a uniform trend for different MCC configurations. In particular, as we increase the remote percentage to 50%, the execution time for the load imbalance ratio of 50 does not differ significantly from the execution time for the load imbalance ratio of 25, thus indicating more tolerance to the load imbalance. This implies that various MCC combination exhibit equal tolerance to the load imbalance irrespective of the number of cores per machine.

C. A Case for Multi-level Partitioning

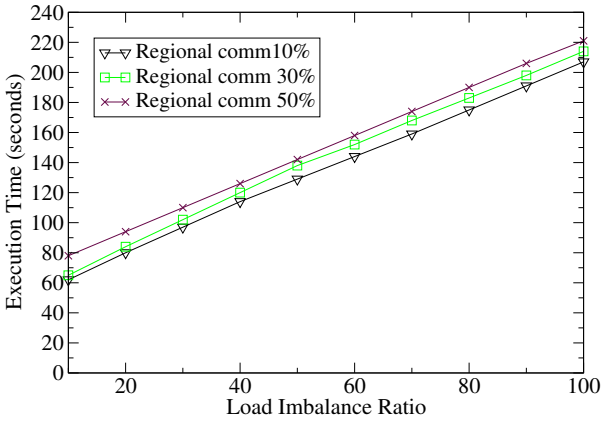
Partitioning refers to splitting the model (e.g. distribution of simulation objects) into different parts that



(a) Traditional Cluster(24 Machines)

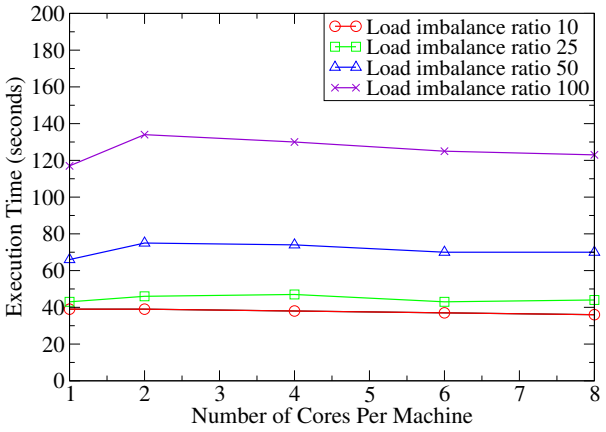


(b) Multicore Cluster(4 Machines/6 Cores Each)

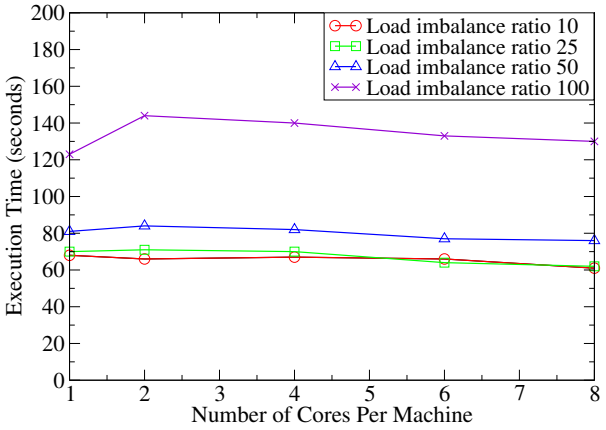


(c) Multicore Machine(8 Cores)

Figure 6: Effect of Load Imbalance



(a) 25% Remote Communication



(b) 50% Remote Communication

Figure 7: Effect of Load Imbalance on Performance with Various MCC Configurations

can be executed simultaneously on different processors. The stock implementation of WarpIV only supports basic object partitioning schemes, such as BLOCK and SCATTER. For our partitioning experiments we needed the capability to separately assign each object to a particular core. In order to achieve this, we augmented WarpIV with the capability to perform manual object decomposition across the simulation nodes by reading the decomposition information from the placement file (similar support was implemented in SPEEDES, but was lacking in WarpIV).

The information about communication frequencies was collected during the simulation (appropriate profiling was implemented in WarpIV) and an object communication graph was obtained. To generate the object partition (i.e. the placement file), we used HMETIS [7] - a well-known graph partitioning tool. It takes three primary inputs - a hypergraph, where each edge is also annotated with the communication frequency, the number of required partitions and an imbalance factor. HMETIS can obtain a range of quality partitions on the basis of this imbalance allowance. The tool can do a better partition if we accept a limited degree of imbalance. In our case this process of partitioning could result in load imbalance on MCC i.e some cores will have more objects and more event processing load on that core, while some cores have less object and less event processing load. Another change that we have implemented in WarpIV was to set up the process affinity, it guarantees that the same logical simulation process always executes on the same core throughout the entire simulation run. This allows better exploitation of caches and also makes the partitioning results more meaningful.

In traditional clusters, a flat partitioning is sufficient because each partition is placed on a separate machine and the only type of communication involved is remote communication. So any partition can be assigned to any machine. In case of MCC, flat partitioning is not aware of two levels of latencies and does not distinguish between remote and regional communication. For example, a 24-way flat partition calculated for a 24-node traditional cluster cannot be directly used for MCC of 4 machines (6 cores each), as the relationship between these partitions is not known. If this placement results into a LP on one machine communicating frequently with LP on another machine, then this can add up to the remote communication. So, in case of MCC of 4 machines (6 Cores each) we first need a machine level 4-way partition which will minimize the remote communication between these machines. Then each of these partitions can be again partitioned 6-way to minimize the regional communication.

As described in background section, the topology component of our benchmark can control various structural properties of the communication hierarchy, such as the height, and dynamic properties such as the extent of communication at each level. For our experiments, we created two sample models using this topology component. *Model 1* is highly clustered, meaning that small groups of objects communicate very frequently within the same group. *Model 2* is Random, meaning that an object communicates with any other object at similar frequency regardless of group membership.

Table I shows the results of three partitioning strategies used on Model 1 and Model 2. We used a multi-core cluster of 3 machines (6 cores each) for these experiments. Block partition is default partitioning scheme available in WarpIV. It places equal sized partitions

Partitioning Strategy	*Model 1* Clustered (seconds)	*Model 2* Random (seconds)
Block	454	454
Flat	160	375
Multi-Level	11	350

Table I: Comparison of Partitioning Schemes

of objects on each core without taking into account any relationships among the objects. Flat partitioning refers to an 18-way partition by HMETIS. Multi-level refers to a 2-pass partitioning where the first pass minimizes inter-machine remote communication (3-way machine level partition) while the second pass attempts to minimize intra-machine regional communication (6-way partition for each of 3 machine-level partitions).

HMETIS	Partitioning Strategy	
Imbalance Factor	Flat (seconds)	Multi-Level (seconds)
1	160	11
5	120	12
10	95	13
15	76	18
20	27	21

Table II: Effect of Load Imbalanced Partitioning

Table II demonstrates the impact of improved partitioning as we increase the imbalance allowance for HMETIS while doing the partitioning. In case of Flat partitioning, the improvement can be attributed to the reduction of remote communication through better quality partitioning.

D. Impact of Event Processing Granularity and Memory Capacity on MCC

Event Processing Granularity (EPC) of an application indicates how fine-grained(low EPC) or coarse-grained (high EPC) the application is. Applications with fine processing granularity tend to be more difficult to parallelize than coarse-grained applications, as they spend less time in independent computation and need to communicate and synchronize more frequently. Figure 8 shows the effect of various EPCs if 10% of all scheduled events result in remote communication. At 10% remote communication, coarse-grained application (EPC 5000) and fine-grained application (EPC 1) exhibited good scalability. It shows that the EPC does not matter as long as the model is scalable.

In a MCC environment, memory can be a major bottleneck because it is shared by multiple cores. The

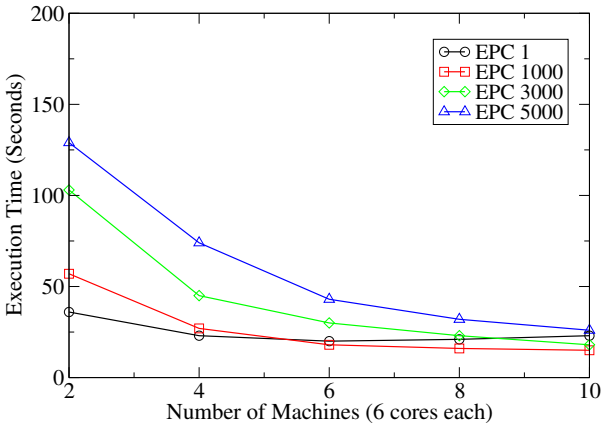


Figure 8: Impact of Event Process Granularity (10% Remote Communication)

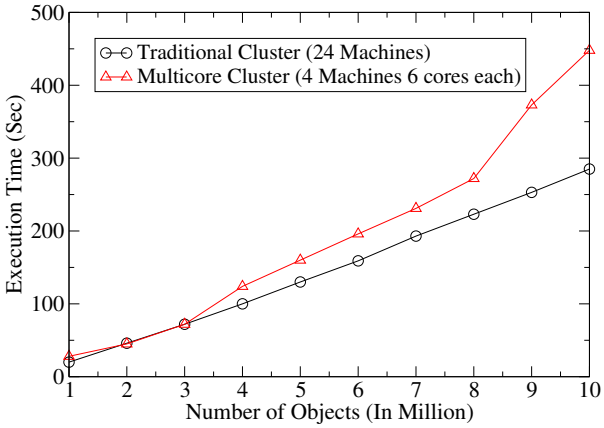


Figure 9: Multicore Cluster: Memory Impact

primary advantage of MCCs over traditional clusters is reduced number of remote communications due to tighter integration of LPs. On the other hand, each individual machine in an MCC now hosts more objects, thus increasing the pressure on the last-level cache, memory bus and main memory. As a result, the cache performance is likely to degrade.

To quantify the impact of the increased memory system pressure on PDES performance, we performed the following experiment. Specifically, we compared the execution time of simulation on traditional cluster with 24 machines and equivalent MCC of 4 machines with 6 cores each with the number of objects increasing in steps of 1 Million. To avoid the impact of communication on execution time, regional and remote communication was kept at 0%, i.e. all communications were local.Figure 9 shows the performance of the MCC platform as a function of the number of objects. The

results demonstrate that the traditional cluster has a linear increase in execution time, while the MCC cluster exhibits linear increase initially and a superlinear increase beyond certain limits. The performance gap between the two systems widens with increased number of objects, which is a direct consequence of increased memory pressure and lower cache performance on MCC. Traditional and Multicore doesn't have any impact on the execution time up to 3 million objects. After this point execution time increases significantly to 448 seconds on MCC platform as compared to 285 seconds on traditional cluster for 10 million objects indicating almost 57% increase in execution time.

Partitioning of objects may result in computational as well as memory load imbalance. From the above results we can conclude that MCCs are also tolerant to memory load imbalance up to certain extent. Therefore the benefit of partitioning on MCC depends on an optimal choice of parameters considering gain of reduced communication and memory and computational load imbalance.

V. Related Work

The importance of optimizing communication in PDES is widely understood [8], [5]. Several researchers have investigated approaches for reducing the number of messages [9], [5], or improving the communication performance [10], [11]. The communication frequency can be reduced with effective partitioning that maps the most commonly communicating objects to the same process [12], [13]. However, existing approaches have not studied partitioning in a system with heterogeneous latencies.

Distributed Shared memory architectures (DSMs) [14] virtualize a cluster of workstations as a single shared memory machine. However, since access to remote memory is much more expensive than access to local memory, performance can severely degrade if heavily used data is placed in a region of shared memory mapped to a remote machine. As a result, these systems are called Non Uniform Memory Access (NUMA) systems; they expose these non-uniform delays to the programmer and leave it to them to place data to localize access. This problem is quite similar to ours and is due to the same physical reasons: the distance between cores, and the latency for communicating between them is not uniform.

The Message Passing Interface (MPI) community (MPI is widely used as the basis for message passing applications) has identified the hierarchical latency issue in Multi-core clusters and have developed support for it [15], that is integrated in most recent versions of MPI. This support provides mechanisms to expose the relationship of cores to each other, and provides support for processor affinity (to control what core a process runs on); the programmer uses this information to partition their application appropriately and to control the mapping of application processes to cores. This paper studies the impact of such architecture sensitive partitioning on PDES. Please note that WarpIV does not use MPI.

Chai, et al [16] have studied the impact of multi-core architecture in cluster computing. This study is geared towards helping application and communication middleware developers to adapt to the Multi-core cluster environment. In our case, the PDES application represents a class of fine grained applications which are much more sensitive to the the non-uniform latency characteristics of the multicore clusters. In addition, we study the effects of load imbalance as this factor is important in case of PDES applications. We also present a multilevel partitioning approach as a possible solution to alleviate the impact of communication and support our observations with a simple partitioning study.

[17] is a study of optimistic parallel discrete event simulation on multi-core platform and is limited to a single multicore machine. Our experiments encompass much broader scope as the multi-core cluster environment is significantly more complex.

[18] and [19] study scalability of PDES on large number of cores. Their experimental setup uses relatively small percentage of remote communication (10%-25%) and does not distinguish between multiple levels of communication. Specialized hardware with tightly integrated cores and high performance network interconnects can contribute towards better scalability demonstrated. We demonstrate results with multiple levels of communication - regional and remote and use a cluster of commodity mainstream multicore machines. We also exercise a wider range of remote and regional percentage to obtain an estimate of impact of communication and possible optimization space.

VI. Concluding Remarks

In this paper, we presented a performance evaluation of PDES on a cluster of multicore processors. To the best of our knowledge, this is the first study which explores the implications of multi-core cluster environment for PDES applications in a systematic fashion. We explored the impact of regional (between cores on the same machine) and remote (between cores on different machines) communications on PDES performance in MCC. We conclude that even though multi-core clusters have tremendous potential to benefit PDES, careful

optimization of remote and regional communication is essential. Other factors such as load imbalance have different degree of impact at inter-machine and intra-machine levels and need to addressed accordingly. Finally, we made a case for multi-level partitioning of the simulation model such that the first level of partitioning reduces the amount of remote communication, while the second level of partitioning reduces the impact of regional communications. We showed significant performance improvements of such a scheme compared to the basic flat partitioning.

Through this study, we want to highlight the need for more research focused towards different aspects of PDES and possible ways of optimizing each aspect to fully exploit the potential of this complex yet promising emerging platform.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants CNS-0916323, CNS-0958501 and CNS-0454298. Nicole Hofmann was supported in part through NSF REU program award no. CCF-0649252.

REFERENCES

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep. UCB/EECS-2006-183, December 2006.

[2] WarpIV Technologies (J. Steinman et al), "The warpiv parallel simulation kernel version 1.5.2," 2008, software available from http://www.warpiv.com/.

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

[4] D. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 405–425, Jul. 1985.

[5] M. Chetlur, N. Abu-Ghazaleh, R. Radhakrishnan, and P. A. Wilsey, "Optimizing communication in Time-Warp simulators," in *12th Workshop on Parallel and Distributed Simulation*. Society for Computer Simulation, May 1998, pp. 64–71.

[6] Metron, Inc., "The SPEEDES parallel environment for emulation and discrete event simulation," 2008, available from http://www.speedes.com.

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

[8] C. D. Carothers, R. M. Fujimoto, and P. England, "Effect of communication overheads on Time Warp performance: An experimental study," in *Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94)*. Society for Computer Simulation, Jul. 1994, pp. 118–125.

[9] J. S. Steinman, "Breathing Time Warp," in *Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS 93)*. Society for Computer Simulation, Jul. 1993, pp. 109–118.

[10] G. D. Sharma, N. B. Abu-Ghazaleh, U. V. Rajasekaran, and P. A. Wilsey, "Optimizing message delivery in asynchronous distributed applications," in *11th International Conference on Parallel and Distributed Computing Systems*, Sep. 1998, (submitted).

[11] G. D. Sharma, R. Radhakrishnan, U. V. Rajesekaran, N. B. Abu-Ghazaleh, and P. A. Wilsey, "Time warp simulation on clumps," in *13th Workshop on Parallel and Distributed Simulation (PADS '99)*, May 1999.

[12] L. Li and C. Tropper, "A design-driven partitioning algorithm for distributed verilog simulation," in *Proc. 20th International Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2007, pp. 211–218.

[13] M. Liljenstam and R. Ayani, "Paritioning PCS for parallel simulation," in *Proc. 5th International Workshop on Modeling Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Jan. 1997.

[14] Http://lse.sourceforge.net/numa/faq/.

[15] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications,," in *pdp, pp.180-186, 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010.

[16] L. Chai, Q. Gao, and D. K. Panda, "Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system," in *The 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2007.

[17] N. Su, H. Hou, F. Yang, Q. Li, and W. Wang, "Optimistic parallel discrete event simulation based on multi-core platform and its performance analysis," in *cisis, pp.675-680, 2009 International Conference on Complex, Intelligent and Software Intensive Systems*, 2009.

[18] K. S. Perumalla, "Scaling time warp-based discrete event execution to 10^4 processors on a blue gene supercomputer," in *Proceedings of the ACM Computing Frontiers*, 2007.

[19] D. Chen and B. K. Szymanski, "Dsim: Scaling time warp to 1,033 processors," in *Winter Simulation Conference, Orlando, FL*, 2005.