Distributed Simulation on a Many-Core Processor

Karthik Vadambacheri Manian and Philip A. Wilsey
Experimental Computing Laboratory,
School of Electronic and Computing Systems,
PO Box 210030, Cincinnati, OH 45221–0030
vadambkk@mail.uc.edu, and philip.wilsey@uc.edu

***Abstract*—Parallel Discrete Event Simulation (PDES) using distributed synchronization supports the concurrent execution of discrete event simulation models on parallel processing hardware platforms. The multi-core/many-core era has provided a low latency "cluster on a chip" architecture for high-performance simulation and modeling of complex systems. A research many-core processor named the Single-Chip Cloud Computer (SCC) has been created by Intel Labs that contains some interesting opportunities for PDES research and development. The features of most interest in the SCC system are: low-latency messaging hardware, software managed cache coherence, and (user controllable) core independent dynamic frequency and voltage regulation capability. Ideally, each of these features provide interesting opportunities that can be exploited for improving the performance of PDES. This paper reports some preliminary efforts to migrate an optimistically synchronized parallel simulation kernel called WARPED to an SCC emulation system called *Rock Creek Communication Environment* (RCCE). The WARPED simulation kernel has been ported to the RCCE environment and several test simulation models have also been ported to the RCCE environment. Based on initial efforts, some preliminary insights on how to exploit some of the exotic features of SCC for increasing the performance of PDES applications is noted.**

***Index Terms*—Parallel and Distributed Simulation, Time Warp, Many-core processors.**

I. Introduction

Discrete Event Simulation (DES) is widely used for performance evaluation across many disciplines, including: computer systems, computer networks, wired and wireless networks, emergency evacuation management, wargaming, and others [1], [2]. Often the simulation models grow very large and can easily exceed the capabilities of large single-processor compute platform. Thus, many simulation analysis activities are based on fairly small simulation models whose behavior is projected into the larger reality that the simulation is attempting to model. The results from these projections can be highly inaccurate [1]. PDES propose to help solve this problem by running the simulation on a cluster of computers, but it has thus far failed to deliver a promise of reliable speedup across many applications. This failure is largely due to the huge mismatch in speeds of execution vs communication of classic parallel platforms. The integrated solution and more uniform performance between communication and computation on Intel's SCC [3], [4] many-core solution will provide a key opportunity for parallel simulation to begin having a dramatic role in the cloud services community.

Support for this work was provided in part by the National Science Foundation under grant CNS–0915337. The Intel c++ compiler used for this project is provided by Ohio Super Computing Center.

In PDES, the concurrently executed simulations communicate by exchanging time-stamped event messages. Unfortunately, the event processing step in most simulation applications are fine-grained computations that generate one or more (but only a few) new events per event execution. One of the key problems in optimizing parallel simulation is finding adequate event processing work during the higher latency event communications. Intel's SCC many-core processor chip provides a low latency on-chip communication medium that should substantially reduce the time disparity between event processing costs and event communication costs.

The Intel SCC chip also has other features that present exciting and unique opportunities for optimizing parallel software codes. For example, the current SCC chip has cache memory that does not enforce coherence; it also has program controlled power and frequency islands allowing independent frequency/voltage settings among subsets of the processor cores. From the perspective of optimistically synchronized parallel simulation (e.g., Time Warp [1], [5]), the ability to use software to modulate power and frequency islands provides an opportunity for the simulation kernel to slow the processing that occur off the critical path and accelerate the processing on the critical path (within the bounds of satisfying the total processor thermal envelope). Thus, balancing the load and potentially accelerating the critical path of the total parallel simulation (similar to Intel's dynamic overclocking). The optimistic nature of Time Warp synchronized simulations may also allow one to exploit the incoherent caches by allowing continued execution for some time without forcing unnecessary coherency checks. However, suitable algorithms to successfully exploit this are not yet known to the authors.

This work focuses on the potential opportunities between Time Warp parallel simulation and the Intel SCC many-core platform. The principle objective of this work is to develop techniques to affect ultra-high performance parallel simulation on many-core processors and ultimately on cloud services provided by clusters of many-core processors. To pursue these investigations, the WARPED simulation kernel [6] is used. WARPED was developed at the University of Cincinnati for supporting large scale simulations (millions of concurrently executed simulation objects) on smaller (32-64 node) Beowulf clusters. WARPED is an excellent starting point for this project because it is a modular design setup with threaded objects setup for execution on a heterogeneous Beowulf platform that contains local (shared memory) and remote (Message Passing Interface, MPI, based messaging) communication capabilities.

This paper reports on some preliminary explorations to effectively utilize the SCC many-core processor for efficient parallel simulation modeling and analysis. This work throws light on further research for running PDES on many-core Beowulf clusters. The rest of the paper is organized as follows: Section II presents some background and related work. Section III provides a brief introduction to parallel simulation and WARPED. Section IV describes the challenges faced while porting WARPED to the RCCE environment. Section V describes the experimental setup and presents some preliminary performance results. Section VI presents some future research directions that we hope to follow with SCC. Finally Section VII concludes the paper.

II. Background and Related Work

Cloud computing will play a significant role in the future for providing general purpose and high performance computing capabilities. Furthermore, the cloud computing platform will almost certainly be composed primarily of multi-core and many-core processing nodes. Hence the issues of efficiently running PDES on multi-core and many-core processors in cloud infrastructure is an important area of research worth exploring. Fujimoto *et al* [7] has analyzed this area and studied the different issues encountered while running PDES on cloud. One of the issues is the processing delays of the PDES simulation due to the load sharing of the node with other tasks in the cloud. Malik *et al* [8] has analyzed this issue and come up with a modified version of the Time Warp protocol to mitigate this issue.

PDES researchers have also worked to enhance the performance of PDES execution on multi-core processors [9], [10]. When PDES runs in a cluster of multi-core nodes, communications between the cores will have substantially lower latencies than communications between nodes. Bahulkar *et al* [9] has studied the behavior of communication between the cores and between the nodes and their overall impact in the performance of the simulation. They found that if the frequently communicating cores are present in the same chip, it greatly enhances the performance of the PDES. This has lead them to focus their studies on partitioning and load balancing.

Intel's SCC chip is an experimental many-core processor created by Intel Labs mainly for the purpose of many-core research efforts. SCC is the first Intel chip with x86 compliant cores on a single die. The die has 48 cores organized into 24 Tiles with 2 x86 cores per Tile (Figure 1). The principle features of the SCC platform are: (i) hardware support for message passing between cores implemented by a 2-D on-chip mesh interconnection network, (ii) an absence of hardware cache coherency on the Tile caches, and (iii) a fine grained, software controllable, dynamic power and frequency management capability.

Each SCC Tile contains a hardware router, 256KB of L2 cache for each core (2) on the Tile, a 16KB shared Message Passing Buffer (MPB), and 16KB of L1 caches in each core. The MPB provides high-performance on-chip message passing capabilities. The message bandwidth is around 1 GB/s and on-die 2D mesh bisection bandwidth is 2 Tb/s. The MPB memory is cached only in L1 cache of the core and hardware coherence is not enforced. Hence care must be taken while accessing the MPB memory. Typically the programmer will invalidate the cache entry before accessing the MPB memory. Since the caches are incoherent, a shared memory application running across multiple cores must use software managed coherence to ensure correct memory accesses.

One more interesting feature of SCC is that the software control of the operating frequency and voltage of the processing cores. Specifically the operating frequency of the cores and the 2D communication network can be controlled by the executing software. As illustrated in Figure 1, the frequency and voltage adjustment occurs in groups of cores (called *islands*) on the chip. Each Tile forms a *frequency island* and 2x2 groups of Tiles form *voltage islands*. Thus there are a total of 28 frequency domains (24 for the processing cores; one each for the system interface, the voltage regulator controller, and the 2D communication network and memory controller) and 7 voltage domains (6 domains for the cores and 1 domain for the 2D communication network). All of the cores in a frequency island will share the same frequency and all cores in a voltage island will share the same voltage. Frequency changes take only a few cycles whereas voltage changes occur on the order of a million cycles. Hence in addition to voltage change instructions, additional instructions are also provided to check whether the voltage change is complete. The inter-core latency within the SCC chip over a 2D-message network is directly proportional to the number of hops taken by the packet. As Figure 1 shows, by default 12 cores in each quadrant are mapped to a specific memory controller. External memory requests are serviced by these memory controllers.

III. Parallel Simulation and WARPED

Research in parallel and distributed simulation focuses primarily on distributed synchronization mechanisms and the methods to optimize them [1]. A distributed simulation will organize a sequential simulation into concurrently executing parts that are called *Logical Processes* (LPs). The LP will concurrently process events (following some synchronization protocol) and exchange timestamped messages to communicate event information designated for another LP.

There are two main categories of synchronization protocols for distributed simulation, namely: (i) *conservative* [11], and (ii) *optimistic* [5], [12]. Conservative techniques implement a strict enforcement of the "happens-before" relationship between events [13] to synchronize the LP event processing activities. In contrast, optimistic techniques do not strictly enforce the event causality relations. Instead optimistically

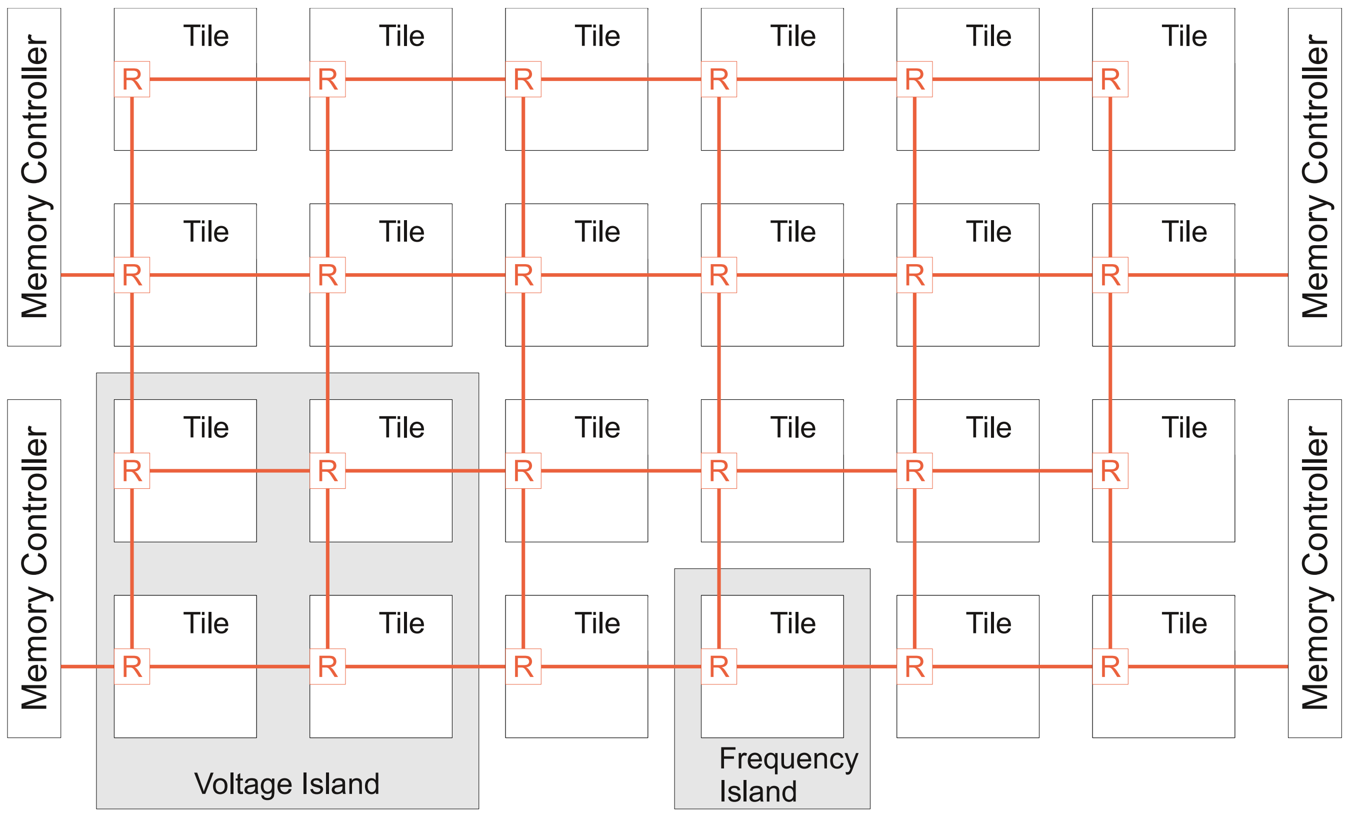

Fig. 1. Architecture of Intel's SCC Processor

synchronized simulations will have some mechanism to detect and recover from an event causality error. This permits optimistic techniques to aggressively process the distributed events and permit greater amounts of parallelism. Of course this comes at the cost of also potentially triggering causality violations that must be repaired.

This paper studies parallel simulation on many-core processors using a simulation kernel called WARPED [6], [10], [14]. WARPED is both a general purpose discrete event API for building simulation models and an implementation of a discrete event simulation kernel (implementing the aforementioned API). The WARPED simulation kernel is highly configurable and has optimized implementations of a sequential and a parallel execution mode. The parallel version implements the Time Warp protocol [1], [5] for synchronization. The design goals of WARPED are to support exploratory research in PDES and to simplify the construction of simulation models for parallel execution. More precisely, the WARPED API hides the implementation details from the simulation model developer. The WARPED code also includes several test simulation models, namely: (i) the classic PHOLD model used by many parallel simulation researchers [1], (ii) a configurable simulation model of a RAID-5 storage array, and (iii) a generic shared memory multi-processor (SMMP). Parallel execution of VHDL models using the WARPED kernel is also possible using the SAVANT/TyVIS tools [15].

Originally WARPED was configured as a collection of highly optimized heavy-weight processes designed to run on Beowulf clusters containing (single or multiple) single-core processors using primarily distributed memory and message passing for communication [6]. More recently the kernel has been expanded and tuned for multi-core processing [10]. While the work to optimize WARPED for execution on multi-core processors is underway, the architecture of many-core processors contain exotic features such as on die network interconnect, no hardware cache coherency, dynamic voltage and frequency regulation, and so on that are not found on conventional multi-core processors. Hence the results obtained for running PDES on multi-core processors cannot necessarily be generalized to many-core processors.

IV. Porting Issues

The experiments with the SCC many-core platform were performed in a software emulation environment that executes on a conventional x86 platform. The emulation environment is called *RCCE*. The RCCE environment provides a framework for software development that closely emulates the SCC communication environment. The WARPED kernel and simulation models are written in C++ and must be migrated to the constrained, shared memory environment and support libraries available for the SCC platform.

The primary language environment for SCC is C and the message passing environment is primarily designed to support a SPMD/synchronous communication application programming environment. Fortunately interfacing the WARPED C++ code with the RCCE API is fairly straightforward. However, there were several challenges to be overcome before the WARPED simulation models could be successfully executed in the RCCE emulator, namely: (i) the use of complex static variables in WARPED, (ii) the asynchronous communication patterns used in the WARPED simulation models, (iii) WARPED has variable length messages sent between LPs, and (iv) issues with the RCCE emulator while executing PDES on more than 30 cores (While this is not a problem that is overcome in these experiments, this section explains why experiments were limited to 30 cores in the emulator). Each of these issues is discussed more fully in the sections below.

A. Static variables issue

The RCCE emulator uses OpenMP to emulate the SCC message passing environment. If the program running in RCCE emulator contains static variables, then they will be initialized only once and shared between all threads. Unfortunately WARPED code contains a significant amount of static variables that are designed to be static to a specific thread, not to all threads. To overcome this issue the RCCE manual recommends the usage of the `#pragma openmp threadprivate` directive on static variables whenever they are encountered and the code is compiled. However, in the current versions of g++, the `threadprivate` directive only works for Plaid Old Data-types (POD) such as `int`, `float`, *etc* and do not work for non POD types such as class objects. Fortunately Intel's icpc C++ compiler can process complex data types in the `threadprivate` directive. Thus, a licensed version of the Intel compiler had to be obtained from the Ohio Supercomputing Center and the WARPED code was modified to build correctly with the icpc compiler.

B. Asynchronous Communication Pattern

The RCCE communication system is designed to support synchronous communication, where a message *send* request must wait for the corresponding message *receive* request. This is not a good match for the asynchronous message passing scenario programmed into WARPED. That is, a WARPED LP sends the message asynchronously and continues with other work. It then periodically polls back to check whether new messages incoming have arrived. Fortunately there exists an asynchronous communication library for the RCCE platform called *immediate RCCE* (iRCCE) [16]. Unfortunately, the ping-ping example pattern in the iRCCE manual and other sample codes in the Many-core Applications Research Community (MARC) [17] forums have all used both non-blocking send or receive in the same function of the application and then they use a blocking call to poll and check whether the requests have completed. Even this communication pattern is not useful for WARPED. WARPED completely decouples the send and receive primitives into separate functions and no blocking call can be used after the send or receive.

The solution that was ultimately successful is to put the asynchronous send and receive requests in a per thread global wait-list. Whenever the application needs to check for messages, it simply checks this global wait-list for completed tasks. This test can be achieved in a non-blocking manner. This method was programmed into the WARPED code for execution with RCCE.

C. Arbitrary length messages

The LPs in a WARPED simulation can exchange multiple message types of varying lengths. Examples of these message types are: initialization, event message, Global Virtual Time (GVT) estimation messages, tests for termination, and so on. As explained in the previous subsection, when messages are obtained from the global waitlist, any type of message can be received from LPs on any other core. Therefore, the message type and size for the next message cannot be known. Unfortunately, the RCCE/iRCCE platform requires that the message size in the send and receive operations match. Thus, the ported WARPED messaging subsystem was modified to send each message in two parts, the first part is a message header containing the length of the actual message and the second part is the actual message. The receive operation is likewise broken into two receives: the first receive reads the message length information and uses that information to trigger the specific command to receive the actual message. Since the order of the messages is guaranteed, this is a workable solution.

D. #pragma omp flush issue

The ported version of WARPED runs in parallel on the RCCE emulator up to 30 cores. However when core count is increased beyond 30, the message headers become polluted, with payload data from the previous message. Problems similar to this are reported in the MARC forums. This may be due to a `#pragma omp flush` issue reported in the MARC forum where the MPB does not reflect the latest content after being written by a thread. As a result, no experimental results are shown in this paper with SCC node counts above 30.

V. Experimental Setup and Results

The simulation experiments were run on two machines. The first is an Intel Core i7-920 with 4 hyper-threaded cores supporting 8 threads and operating at 2.67 GHz. The second is a dual core Intel Core2Duo supporting 2 threads operating at 2.00GHz. Both machines have 3 Gb of RAM and are running Linux (version 2.6.x).

Four simulation models are packaged with the WARPED simulation kernel, namely: PHOLD, RAID, SMMP, PING-PONG. PHOLD is a synthetic simulation widely used by the parallel simulation community for showing performance results. The PHOLD configuration used in these experiments contains 4 LPs with an event density of 4 and with an exponential distribution and a seed of 1.0. The RAID simulation simulates a RAID 5 disk array composed of 4 disks and with total of 100 I/O requests issued by two LPs. SMMP is a simulation model that simulates a symmetric multiprocessing environment containing 8 processors, with cache speed 10

Model	Runtime (secs)
PHOLD	835.50
RAID	72.40
SMMP	229.50
PINGPONG	49.50

TABLE I
SIMULATION ON 30 CORES WITH THE INTEL I7

MPB Size (bytes)	i7 Runtime (secs)			
	PHOLD	RAID	SMMP	PINGPONG
100	1.03	0.16	5.32	4.08
150	1.01	0.15	5.32	2.10
200	1.00	0.15	5.27	2.53
8K	0.98	0.14	5.27	1.93

TABLE II
MPB ANALYSIS WITH 8 EMULATED SCC CORES ON THE I7

times that of main memory and with cache hit ratio of 0.85. During the simulation 1,000 memory requests are made to the memory space by each of the 8 simulated processors. Finally, the PINGPONG simulation contains a fixed set of balls that are circulated among a fixed set of players (LPs). A subset of players start the simulation by circulating the balls to other players. The simulation ends when all the balls are received back at the originating LP.

A summary of the simulation runtimes for the emulated 30 core SCC platform is shown in Table I. These results were run on the Intel i7 platform and they simply show the completion of all of the simulation models on an emulated configuration of 30 cores for the SCC platform. In the next two sections, studies to evaluate the impact of message size and to show the potential impact that voltage and frequency adjustments might have are described.

A. Analysis of MPB size

To show the impact of the message passing buffer size on simulation performance, the above simulation models were run in the SCC emulator for varying sizes of the MPB (Table II). By default, the MPB for each core is 8K. However, the maximum message size used by the simulation models is 235 bytes. Hence the default 8K is more than sufficient for these simulation models. The simulations were run on the Intel i7 and results are presented in Table II. The Table clearly shows that the MPB size affects mainly the PINGPONG simulation and other simulations are not significantly affected. Thus the performance impacts depends not only on computation load of the processors but may also be due to their communication pattern. This is because, of the 4 simulations, SMMP and PINGPONG have simulation objects executing on all the 8 cores and even SMMP have more simulation objects than PINGPONG. But interestingly PINGPONG is more affected by MPB variation than SMMP. This may be due to more inter-core communications in PINGPONG than in SMMP. But this needs to be verified further by a detailed investigation on this subject.

Model	# SCC cores	Core id	Time (sec)	Rollbacks
PHOLD	4	0	346.87	2462
		1	352.06	3794
		2	352.57	2911
		3	352.76	2640
RAID	4	0	26.36	381
		1	26.53	175
		2	26.52	1344
		3	26.54	836
SMMP	4	0	71.92	20052
		1	71.91	2359
		2	71.94	1201
		3	71.92	4531

TABLE III
SIMULATION RESULTS FROM CORE2DUO

VI. PDES RESEARCH DIRECTIONS WITH SCC

With the changes outlined in Section IV, all of the WARPED example simulation models (except VHDL, which was not yet attempted) were run on the RCCE simulator. The work is still embryonic and just barely scratched the surface of possibilities and opportunities with the SCC platform. However, even in this preliminary state, one can draw some interesting insights. These are described below.

A. Harnessing voltage and frequency control of SCC

The dynamic voltage and frequency control features of SCC could be highly useful for balancing and optimizing the performance of Time Warp synchronized PDES simulations. In particular, the concurrent LPs of a Time Warp simulation process events aggressively without regard for all event causalities. Thus, some LPs may have frequent rollbacks while others (on the critical path) may have minimal rollbacks. For example, the above simulation models were run for a configuration of 4 emulated SCC cores and detailed runtime and rollback numbers were collected. These results are shown in Table III. The Table shows that the simulation results for RAID show that the LP on core #1 has only 175 rollbacks while the LP on core #2 has 1344 rollback. Likewise SMMP shows widely varying rollback performance among the various cores. By decreasing the frequency of the cores having excessive rollbacks and increasing the frequency of cores having minimal rollbacks, the simulation may actually be able to accelerate the critical path of execution for faster overall simulation throughput. Thus, on many-core processors with suitable thermal monitoring and frequency control capabilities, application specific dynamic overclocking can function to maximally increase overall throughput. Unfortunately there is no way known to test this hypothesis with the RCCE software emulator.

B. Impact of communication on simulation performance

Another interesting area would be to study the communication between the cores of SCC within a single chip and communication between cores of SCC on different nodes and their impact on the overall performance of PDES. This is similar to the study of Bahulkar *et al* [9] but now on many-core processors.

C. Combining shared memory and distributed memory Time Warp protocols

In addition to the per-core private memory and message passing buffer, SCC has a significant amount of off die memory shared between the cores. The amount of shared memory is configurable. Time Warp protocol is conventionally used on either shared memory or distributed memory architectures and the design of each system varies significantly [6], [18]. The SCC provides a unique opportunity to take the best of both worlds and to come up with an efficient combined solution. A similar work is done by Sharma *et al* [19] on clusters of multiprocessors. The emphasis of their work is on exploiting the parallelism of Symmetric MultiProcessing (SMP) node rather than integrating both shared and distributed memory time warp designs. One important hurdle to cross in this direction is maintaining the cache coherency. In SCC no hardware cache coherency is present for want of scalability of the cores. Hence cache coherency in SCC has to be maintained in software which by itself is an interesting research direction.

D. Multilevel Time Warp

Traditional Time Warp optimizations are designed to hide the high network latency by performing useful work during the network communications. However, the high speed on chip network on the SCC processors supports a relatively low network latency. Hence it may be time to revisit the classical Time Warp optimizations such as lazy cancellation to see whether their overhead outweighs their merit in many-core chips. Finding optimizations for PDES on many-core chips is a new avenue for research. Further, in the cluster of many-core processors, the events can be obtained from a local or remote cores. Hence classical Time Warp optimizations can applied to remote events and switch to many-core specific optimizations for events from local core. Hence multilevel Time Warp protocol can be used to efficiently handle both the local events and remote events. More study needs to be done in this area to see the extent of practical usefulness.

VII. Conclusion

This initial work with parallel simulation on the RCCE emulator has provided a few insights on programming needs for future many-core processors. This is a first step in analyzing the potential perform of the many-core SCC platform for efficiently supporting Time Warp synchronized parallel simulation. The possibility to adjust frequency and voltage settings to optimize critical path performance (while maintaining safety under the processor's thermal limits) is an interesting prospect for study. Likewise, the incoherent caches on the SCC platform present opportunities. The dynamic state saving in Time Warp and the opportunity to repair damage from incorrect or premature computations may allow for the development of algorithms to exploit the incoherent caches in interesting ways to increase performance. In any event, the features of many-core processors present numerous interesting opportunities and challenges for the parallel simulation community.

References

[1] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, pp. 30–53, October 1990.

[2] A. M. Law and W. Kelton, *Simulation Modeling and Analysis*, 3rd ed. Mc Graw Hill, 2001.

[3] Intel Press Release, Intel Corporation, "Futuristic intel chip could reshape how computers are built, consumers interact with their pcs and personal devices," Intel Press Release, Intel Corporation, Tech. Rep., Dec. 2009. [Online]. Available: http://www.intel.com/pressroom/archive/releases/20091202comp_sm.htm

[4] J. Howard *et al.*, "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, 7-11 2010, pp. 108 –109.

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

[6] D. E. Martin, P. A. Wilsey, R. J. Hoekstra, E. R. Keiter, S. A. Hutchinson, T. V. Russo, and L. J. Waters, "Redesigning the warped simulation kernel for analysis and application development," in *Proceedings of the 36th annual symposium on Simulation*, ser. ANSS '03, 2003, pp. 216–223.

[7] R. Fujimoto, A. Malik, and A. Park, "Parallel and distributed simulation in the cloud," *SCS M&S Magazine*, 2010.

[8] A. Malik, A. Park, and R. Fujimoto, "Optimistic synchronization of parallel simulations in cloud computing environments," in *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, ser. CLOUD '09, 2009, pp. 49–56.

[9] K. Bahulkar, N. Hofmann, D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev, "Performance evaluation of pdes on multi-core clusters," in *Proceedings of the 2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications*, ser. DS-RT '10, 2010, pp. 131–140.

[10] R. Miller, "Optimistic parallel discrete event simulation on a beowulf cluster of multi-core machines," Master's thesis, University of Cincinnati, Cincinnati, OH, 2010.

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

[12] K. M. Chandy and R. Sherman, "Space-time and simulation," in *Distributed Simulation*. Society for Computer Simulation, 1989, pp. 53–57.

[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[14] R. King, "WARPED redesigned: An api and implementation for discrete event simulation analysis and application development," Master's thesis, University of Cincinnati, Cincinnati, OH, 2011.

[15] P. A. Wilsey, D. E. Martin, and K. Subramani, "SAVANT/TyVIS/WARPED: Components for the analysis and simulation of VHDL," in *VHDL Users' Group Spring 1998 Conference*, Mar. 1998, pp. 195–201.

[16] C. Clauss, S. Lankes, J. Galowicz, and T. Bemmerl, "ircce: A non-blocking communication extension to the rcce communication library for the intel single-chip cloud computer," RWTH Aachen University, Tech. Rep., Feb. 2011. [Online]. Available: http://communities.intel.com/message/110482#110482

[17] "Marc - manycore application research community." [Online]. Available: http://communities.intel.com/community/marc

[18] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "Gtw: a time warp system for shared memory multiprocessors," in *Proceedings of the 26th conference on Winter simulation*, ser. WSC '94, 1994, pp. 1332–1339.

[19] G. D. Sharma, R. Radhakrishnan, U. K. V. Rajasekaran, N. Abu-Ghazaleh, and P. A. Wilsey, "Time warp simulation on clumps," in *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, ser. PADS '99, 1999, pp. 174–181.