Parallel Discrete Event Simulation Based on Multi-Core Platform-A New Approach

Manzoor G.Ahmed
EXTC Department
Dr.NPH Polytechnic,
Pusad

Shirish V.Pattalwar
EXTC Department
PRMIT &R, Badnera,
Amravati

Vilas M. Thakare
Deptt. Of Computer Science
SGB Amravati University
Amravati

ABSTRACT

Multi-core designs have become commonplace in the processor market, and are hence a major focus in modern computer architecture research. Thus, for both product development and research, multiple core processor simulation environments are necessary. Multi-core computer offer a new parallel computing platform with high performance-price ratio and small volume to parallel simulation. Existing parallel simulator especially PDES simulators commonly run on parallel computers or clusters with Linux or Unix OS. The prices of super computer and large scale cluster are too high to be afforded, which limits the extensive popularization of PDES. This paper discusses a brief overview of multi-core processors and existing approaches to Parallel Discrete Event Simulation based on multi-core computer platform. A novel approach to proposed to explore the Parallel Discrete Event Simulation based on multi-core computer platform that can run on desktop with windows OS directly.

1. INTRODUCTION

The motivation for this paper is the proliferation of multi-core, many-core and multi-core cluster architectures, which have inspired us to through lights on existing simulation methodology and simulator for parallel simulation based on these platform and proposed different approach for PDES based on multi-core platform that can run on desktop with windows OS directly.

Parallel hardware has been used in server environments for a long time but recent client platforms (i.e., desktop and laptop computers) have also been adopting multi-core processors. In the field of modeling and simulation, simulation applications put forward higher and higher requirement on the executing speed as the modeled physical systems are becoming more and more complicated. Parallel simulation is an effective way to speed up the simulation.

In most of the work carried out on parallel simulation, the target machine and host machine used were cluster or SMP platform with Linux or Unix OS. *W*riting parallel simulators can be extremely difficult since traditional serial and parallel software cannot fully exploit multi-core's capability and computing power without parallelizing restructure [1]. Also maintaining causality during parallel execution is the central challenge both for the correctness of the simulation and for achieving good simulation performance The Multi-core processor has come into the market for just about five years, and according to so-called new Moore's Law, the number of cores per chip wills double every 2 years [2,3]. If this holds true, multi-core machines will soon evolve to many-cores, with 10s if not 100s of cores per chip. The terms

many-core and massively multi-core are sometimes used to describe multi-core architectures with an especially high number of cores (tens or hundreds). Already, there are some special-purpose(research)multi-core processors that are available from a number of vendors and some are under development with 64 cores (Tilera [4]), Intel's 80-core (Polaris prototype [5]), IBM's 80 core (Cyclops-64[6], Ambric's 336 core (Am2045[7]), and even graphics engines with 960 cores (NVIDIA Tesla S1070 [8]). As a result, we have entered the era of Multi-core clusters (MCCs). Currently research on parallel simulation based on multi-core, many-cores and Multi-core clusters platform is in early phase. Specifically research that will shift the platform of PDES from traditional supercomputer to multi-core computer has bright prospect .So there exist great demand & challenge to write the future desktop simulation software that will be the parallel simulation based on multi-core or many-core platform that could run on Windows OS directly. *A* brief overview of multi-core processors is discussed in section-2:, section-3 discusses PDES and its challenges, section-4 previous work i.e. the Existing methodology for PDES based on multi-core platform and section-5 the proposed methodology for PDES based on multi-core platform that can run on Windows OS directly. Finally conclusion drawn on basis of use and complexity.

2. MULTI-CORE PROCESSOR

A multi-core processor is a single computing component with two or more independent actual processors (called "Execution cores"). And each core has its own set of execution and architectural resources required to run without blocking resources needed by the other software threads. Depending on design, these processors may or may not share a large on-chip cache. As with any technology, multi-core architectures from different manufacturers vary greatly. Along with differences in communication and memory configuration another variance comes in the form of how many cores the processor has. Figure1 shows a comparison of typical single core & multi-core processor architecture. Typically cores are integrated onto a single die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package. Multi-core processors are MIMD: Different cores execute different threads (Multiple Instructions), operating on different parts of memory (Multiple Data) in parallel. To exploit parallelism OS perceives each core as separate processor and maps the threads/processes to different cores on time-sliced basis as depicted in figure2.

Since past few years multi-core trend has increased dramatically among manufacturers and computing world, especially Intel and AMD moving along nicely. Current commercial multi-core line ups of Intel includes the latest Intel Core i7, Intel Core i5 and Intel Core i3, and the older Intel Core 2 Solo, Intel Core 2 Duo, Intel Core 2 Quad and Intel Core 2 Extreme lines, and not to forget its under development 80-core research processor(Polaris prototype [5]).Similarly AMD has the Althon lineup for desktops, Turion for laptops, and Opteron for servers/workstations and recently 8-core AMD FX CPU, for which AMD has been awarded by Guinness in septmember2011 for achieving the world's fastest desktop CPU with 8.429 GHz. Much of the motivation and increasing trend for multi-core processors among manufacturers and computing world come from greatly diminished gains in processor performance from increasing the operating frequency. This is due to three primary factors:

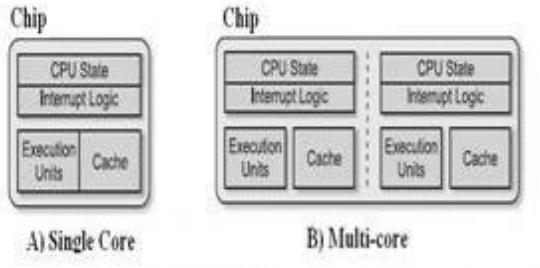

Figure 1: Simple Comparison of Single core & Multicore Architecture

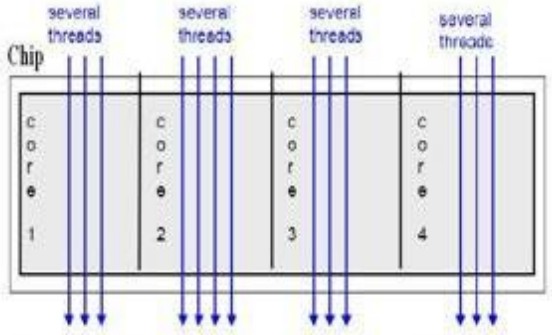

Figure 2: Multi-core Processor (One chip,Multi-core,Mulitple threads)

1. The *memory wall*; the increasing gap between processor and memory speeds. This effect pushes cache sizes larger in order to mask the latency of memory. This helps only to the extent that memory bandwidth is not the bottleneck in performance.
2. The *ILP wall*; the increasing difficulty of finding enough parallelism in a single instructions stream to keep a high-performance single-core processor busy.
3. The *power wall*; the trend of consuming exponentially increasing power with each factorial increase of operating frequency. This increase can be mitigated by "shrinking" the processor by using smaller traces for the same logic. The *power wall* poses manufacturing, system design and deployment problems that have not been justified in the face of the diminished gains in performance due to the *memory wall* and *ILP wall.*

Furthermore due to incapability of single core processor to improve operating systems' ability to perform multitasking applications simultaneously, the computing world has started to embrace multi-cores for increased performance, power efficiency and compute capacity. This is because multi-cores effectively consume less power by running at lower clock rates but still increasing the throughput because of parallel processing. From a white paper released by Intel Corporation [9], based on the experiments conducted in their lab, we see that addition of a second core to an existing single core system, allows the clock speed to be lowered by 20%, at the same time delivering a 73% increase in performance. The largest boost in performance will likely be noticed in improved response-time while running CPU-intensive processes, like antivirus scans, ripping/burning media (requiring file conversion), or file searching. For example, if the automatic virus-scan runs while a movie are being watched, the application running the movie is far less likely to be starved of processor power, as the antivirus program will be assigned to a different processor core than the one running the movie playback.

3. PDES BASED ON MULTI-CORE

As the same with other software, simulation software has to be parallelized so as to make full use of multi-core platform's computing power. Discrete event simulation executes simulation events according to their time-stamp order. Parallel discrete event simulation distributes simulation entities and events to multiple processors (or executing cores) so as to speed up the execution of simulation. PDES can be deemed as multiple serial simulations and each serial simulation is called a Logical Process (LP). Multiple serial simulations run at the same time and communicate with each other by exchanging time-stamped messages [10]-[11],[21]-[22]. In order to parallelize discrete event simulation on multi-core platform, parallel programming model and synchronization algorithm are two of the most important problems or challenges to be solved are programming model and synchronization algorithm.

3.1 Parallel Programming Model

In order to partition simulation into multiple LPs and distribute these LPs among executing cores on multi-core platforms for running, a parallel programming model should be needed. Shared memory model and message passing model are two popular parallel programming models. If shared memory model is used, a software thread will be created for each LP and threads communicate with each other by accessing shared variables and using thread synchronization primitives. The features of this model are single address space, easy to program, bad portability, etc. Shared memory model could be implemented through system calls (Windows and Unix system functions), thread libraries (such as Win32 threads, POSIX threads, OpenMP, Threading Building Blocks [12] or programming language support (such as JAVA and C#). If message passing model is used, a software process will be created for each LP and processes communicate with each other by sending and receiving explicit messages. The features of this model are multiple address spaces, difficult to program, good portability, etc. Message Passing Interface (MPI) [13] and Parallel Virtual Machine (PVM) are two of the most popular message passing libraries. Whether shared memory model or message passing model is adopted, the multiple processes/threads created are all scheduled by operating systems. Generally they will be assigned the same priority. Programmers need not to distribute them to executing cores manually.

3.2 Synchronization Algorithm

By parallel programming model, we distribute multiple LPs to multiple cores on multi-core platform and execute LPs simultaneously. Unfortunately, events can't be ensured to access LPs in time-stamp order i.e. an event with a smaller timestamp has the potential to modify the state of the system and thereby affect events that happen later. This is what we call the causality constraint [14]. For example, after *LP2* executes an event *Ea* (with time-stamp 36), *LP1* may execute an event *Eb* (with time-stamp 15) and generates an event *Ec* (with time-stamp 21) that *LP2* must execute. Then *Ec* accesses *LP2* after *Ea* even though the time-stamp of *Ec* is smaller than *Ea*.This problem is called synchronization of PDES and it's the central problem of PDES. A synchronization algorithm is needed to ensure that events are processed in a correct order and the parallel execution of the simulator yields the same results as a sequential execution

Synchronization algorithms can be broadly classified as either optimistic or conservative.

Optimistic algorithms use a detection and recovery approach. If events are processed out of timestamp order, a mechanism is provided to detect and recover from such errors. The following are some concepts related to optimistic algorithms:

1) State Saving. In order to recover from errors, the states before LP processes events should be saved. There are some commonly used methods for state saving, such as whole state saving, periodic state saving, incremental state saving and reverse computation.

2) Roll Back. When LP receives an event with time-stamp smaller than its local simulation clock (this event is called a straggler event), it should restore its state and send anti-message to cancel the event sent earlier. This process is called roll back.

3) Global Virtual Time (GVT). GVT at wallclock time *T* (*GVTT*) during the simulation execution is defined as the minimum time-stamp among all unprocessed and partially processed messages and anti-messages in the system at wallclock time*T*. Samadi's GVT algorithm and Mattern's GVT algorithm are two of the most commonly used algorithms.

4) Fossil Collection. Optimistic synchronization algorithm should consume much memory to save states and events. After GVT is calculated, memory used by states and events that are older than GVT can be reclaimed and reused. This process is called fossil collection.

Conservative algorithm eliminates the possibility of any causality errors; that is, an LP is blocked from processing the next event in its event-list until it is sure that it will not cause out-of-order event execution due to future events from other LPs.

4. PREVIOUS WORK

Using Message passing model, optimistic synchronization algorithm and referring to open-source PDES simulators such as WARPED 2 [15], Nianle Su, Hongtao Hou, Feng Yang, Qun Li, and Weiping Wang at all [19] choose the C++ language and MPICH message passing library to develop an optimistic PDES simulator which can run effectively on multi-core computer with Windows OS. They have adopted MPI as message passing library, where in interaction among LPs in PDES is completed entirely through explicit messages. Several kinds of messages need to be transferred, such as initialization message, start message, event message, negative event message, GVT message, GVT update message, terminate token. Before these messages are sent, they have to be transformed into byte stream through serialization. After received, byte stream has to be transformed back into different kinds of messages through deserialization. The optimistic simulation algorithm implemented using Time Warp protocol [16].The effects of event granularity, process number, lookahead on the simulation performance are analyzed on Phold model [17] with HP ProLiant ML150 server with two-way Intel Xeon Quad-core processors and 4GB memory. The optimistic PDES based on multi-core platform could achieve good speedup for applications with coarse-grained events.

5. PROPOSED METHODOLOGY

PDES simulator mentioned above is developed using Optimistic synchronization protocol which introduces a roll-back mechanism providing proper synchronization across event cores, by reverting to a previous state if a causality error occurs. Here we have proposed a method which uses conservative synchronization protocol often referred to as the Chandy-Misra-Bryant (CMB) protocol by Chandy and Misra [20], which avoids the possibility of any type of causality error ever occurring by determining when it is safe to process an event. For example if a process P contains an unprocessed event E_1 with time stamp T_1 such that T_1 is the smallest timestamp it has, then it must ensure that it is impossible for it to receive another event with a lower time stamp before executing E_1 . In CMB, LPs are connected via directional links, through which events are transferred from one LP to another in chronological order. This protocol introduces the concept of a lower bound time stamp (LBTS) as the minimum timestamp an individual event core can safely advance to. Additionally, null-messages are broadcast by an event core to inform the other cores of its current local virtual time (LVT), in order to ensure correct LBTS calculation and to avoid deadlock [18].

5.1 Implementation

In the proposed approach, an LP will be implemented by an **EventCore**, containing a priority queue as an EVL. A standard priority queue from the Java collections framework is to be used for the EVL implementation. This implementation is based on a priority heap and provides a time complexity of *O(log(n))* for insertion, *0(n)* for removal and *O(1)* for retrieval operations. The time management service is needed to be provided by the **PDESTimeManager** present in each **EventCore**. The time manager will control the time advances of the **EventCore** by keeping track of the LVT of the other event cores. An event core broadcasts nullmessages to each other core when it advances the LVT to avoid deadlock. All **EventCore's** have incoming and outgoing channel endpoints to each other event core. The set-up and operation of these channels is managed by the **CoreCommunicationManager.** Each input queue Qi has a timestamp field T(Qi) in each channel ,based on which the **PDESTimeManager** performs an LBTS calculation. Graphical representation of this system is shown in figure 3.

The PDESTimeManager on the right shows its input queues, containing events and possibly null-messages with their firetimes *ti*, and the T(Qi) field, linked to the input queue endpoints Qi. The T(Qi) field contains the firetime value of the last received event, used by the PDESTimeManager to calculate the LBTS value. The EventCore depicted on the left, contains the LVT clock and the event queue. Event processing may spawn new internal or outgoing events that are rescheduled locally or sent to other cores respectively. The solid lines represent the flow of events, while the dashed lines represent the interactions relevant to the time synchronization mechanism.

5.2 Performance Analysis

To analyze both the overheads of the parallel simulator and the effects of event granularity, process number, lookahead on the simulation performance PHOLD model is underway, in which the event core requests the time manager to advance its LVT. The request is granted if the requested time is smaller than or equal to LBTS+*l*, with look-ahead *l*. When the event core is not allowed to advance the LVT and if there are no more events to process in the EVL, new events are pulled from all the input queues and inserted into the EVL. This process changes the LBTS, allowing the event core to advance further and execute the pending events in the EVL. If an event core is not allowed to advance and there are no more pending incoming events, the event core enters a wait state until new events arrive at the input channel endpoints.

The hardware platform of this test is HP Inte(R) Core (TM) 2 Duo CPU processors and 3GB memory.

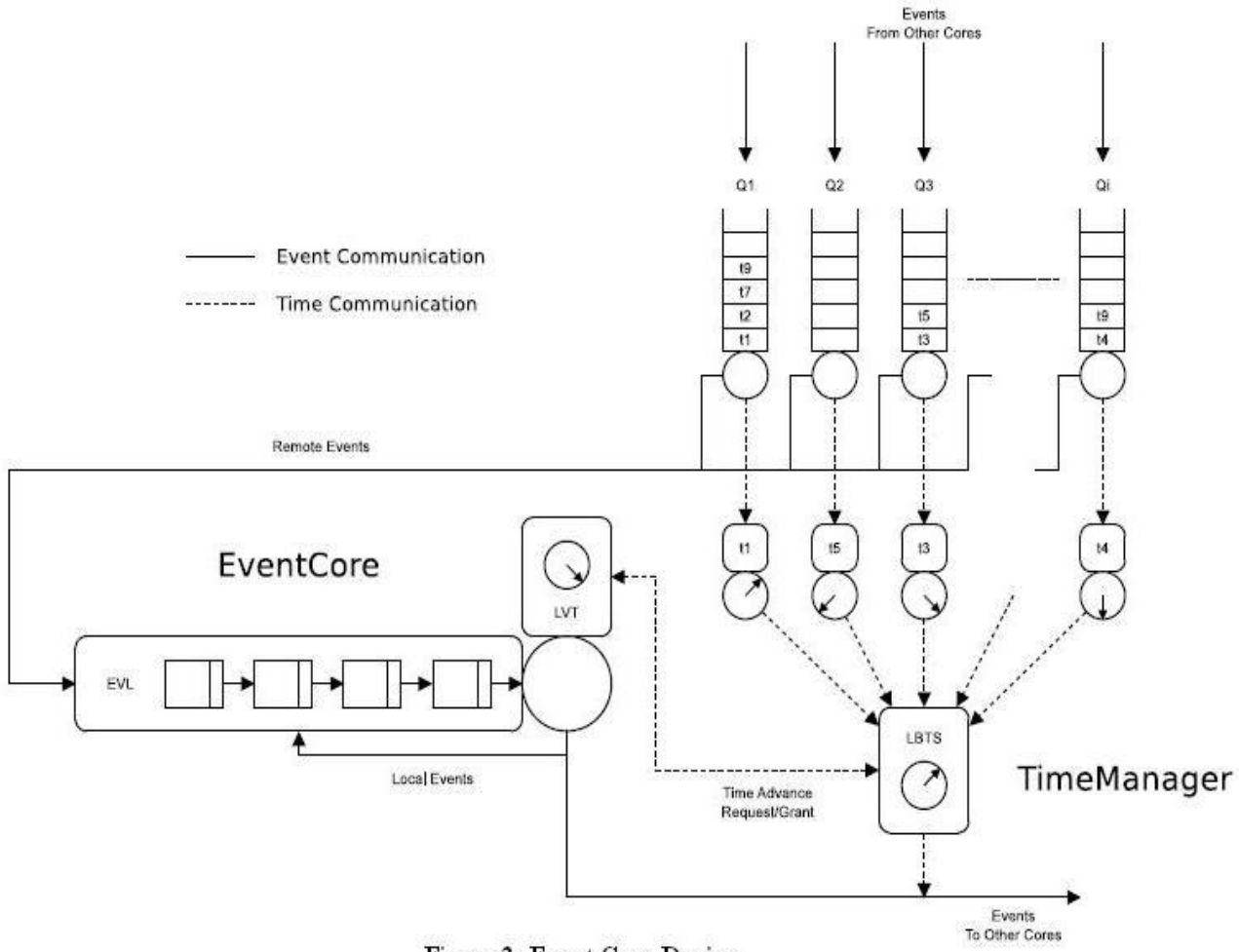

Figure 3: Event Core Design

6. CONCLUSION

Parallel discrete event simulation based on multi-core platform using optimistic methods such as Time Warp are the best way to simulate large simulation problems, while conservative methods offer good potential for certain class of problems. But Optimistic approach takes a large amount of memory, must be able to recover from arbitrary errors , infinite loops, much more complex, keeping in mind we have proposed a new approach adopting conservative synchronization protocol to develop PDES based on multi-core platform , which is relatively less complex and that can run on window OS directly.

7. REFERENCES

[1] Jia-an, W. Cheng-shan, Wu Ai-guo," A study of power system parallel simulation methods based on multi-core multithreaded processor platforms" International Conference on 15-17 April 2011.

[2] Wikipedia,"Moore'sLaw", http://upload.wikimedia.org/wikipedia/commons/0/06/Moore_Law_diagram_(2004).png.

[3] 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.

[4] Sannella, M. J. 1994 Constraint Satisfaction and S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson,J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In Proceedingsof ISSCC, 2007.

[5] Juan del Cuvillo, Weirong Zhu, Ziang Hu, and Guang R. Gao. "Toward a software infrastructure for the cyclops-64 cellular architecture". In Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative.

[6] K. Pedretti, S. Kelly, and M. Levenhagen, "Summary of Multi-Core Hardware and Programming Model Investigations,"Sandia National Laboratories, Albuquerque, New Mexico, USA, Technical Report SAND2008-3205, 2008.

[7] R M. Fujimoto. Parallel discrete event simulation. Commun. ACM, 33(10):30–53, 1990.

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

[9] J. Reinders, Intel Threading Building Blocks. US:O'Reilly, 2007.

[10] Jason Liu Parallel discrete event simulation," School of Computing and Information Sciences Florida International University February 9, 2009

[11] D. E. Martin, P. A. Wilsey, R. J. Hoekstra, R. J. Hoekstra, et al., "Redesigning the WARPED Simulation Kernel for Analysis and Application Development," in

[12] Jefferson DR. Virtual time. ACM Trans. Program. Lang. Syst. 1985; 7(3):404–425, doi:10.1145/3916.3988.

[13] Fujimoto, "Performance of Time Warp under Synthetic Workloads," Proceedings of the SCS Multiconference on Distributed Simulation, vol. 22, pp. 23-28, Jan. 1990

[14] handy KM, Misra J. Asynchronous distributed simulation via a sequence of parallel computations. Commun. ACM 1981; 24(4):198–206, doi:10.1145/358598.358613

[15] Nianle Su, Hongtao Hou, Feng Yang, Qun Li, and Weiping Wang, "Optimistic Parallel Discrete Event Simulation Based on Multi-core Platform and its Performance Analysis" 2009 IEEE.

[16] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering, SE-5(5):440–452, May 197.

[17] M. Chidester and A. George, "Parallel Simulation of Chip Multiprocessor Architectures," ACM Transactions on Modeling andComputer Simulation, vol. 12, no. 3, pp. 176–200, July 2002.

[18] K. Barr et al., "Simulating a Chip Multiprocessor with a Symmetric Multiprocessor," in Proc. of the Boston Area Architecture Workshop, Jan. 2005.