Optimistic Parallel Discrete Event Simulation Based on Multi-core Platform and its Performance Analysis

Nianle Su, Hongtao Hou, Feng Yang, Qun Li, and Weiping Wang
Department of Systems Engineering, School of Information System and Management, National University of Defense Technology, Changsha 410073, China
sunianle@nudt.edu.cn

Abstract

The development of computer processor has stepped into the era of multi-core, providing a good chance to spread the parallel discrete event simulation. The parallel programming model and synchronization problem during the parallelization of discrete event simulation on multi-core platform were discussed. A parallel discrete event simulator based on multi-core platform was designed and implemented using the optimistic synchronization algorithm. On an HP multi-core server with up to 8 cores, both the overheads of the parallel simulator and the effects of event granularity, process number, lookahead on the simulation performance were tested using the Phold model. The experiment results show that the optimistic parallel discrete event simulation based on multi-core platform could achieve good speedup for simulation applications with coarse-grained events.

1. Introduction

The development of current computer processor has stepped into the era of multi-core. With the naissance of multi-core processor and its on-going development, concurrency will be the next major revolution after the object-oriented revolution in how we write software [1]. 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 running of simulation. Multi-core computer will offer a new parallel computing platform with high performance-price ratio and small volume to parallel simulation.

The Multi-core processor has come into the market for just about three years, so currently there is little research on parallel simulation based on multi-core platform. Ref. [2] proposed a multi-threaded methodology for parallelizing multi-core architecture simulators that were used to aid engineers in designing processors. Each POSIX thread simulated the activity of an executing core. Ref. [3] demonstrated the parallelization of Mambo system simulator on multi-core platform. A multi-scheduler methodology was proposed and each thread executed a scheduler. Ref. [2] and Ref. [3] made use of thread synchronization primitives such as mutex, event, critical section and semaphore to ensure the correctness of the simulation causality. The synchronization was used at every time step, limiting the farther improvement of the simulation performance. Ref. [4] made a suggestion to research the parallel logic simulation using multi-core workstations, but the research of [4] is still at the stage of suggestion. Ref. [5] explored the simulation paradigm of simulating each core of a target multi-core processor in one thread and then spreading the threads across the hardware thread contexts of a host multi-core processor. It started with cycle-by-cycle simulation and then relaxes the synchronization condition in various schemes, which are called slack simulations.

Parallel Discrete Event Simulation (PDES) is another formalism of simulation where simulation time does not advance from one time step to the next but, rather, advances from the time-stamp of one event to the next. Event-driven execution will be more efficient because it only updates state when an event occurs. PDES on shared-memory multiprocessors and distributed-memory multicomputers [6-9] has been an active research topic for several decades.

This paper researches the parallel discrete event simulation based on multi-core platform using optimistic synchronization algorithm. Section 2 introduces the multi-core computer and its influence on software development. In section 3, parallel programming models and synchronization problem during the parallelization of discrete event simulation on multi-core platform are analyzed. A parallel discrete

978-0-7695-3575-3/09 $25.00 © 2009 IEEE
DOI 10.1109/CISIS.2009.59

event simulator based on multi-core platform is designed and implemented using the optimistic synchronization algorithm. Finally, the Phold model is used to test the overheads of the simulator and the effect of several factors on the simulation performance.

2. Multi-core computer

It has become harder and harder to exploit higher CPU clock speeds due to not just one but several physical issues, notably heat, power consumption, and current leakage problems. Therefore designers have begun to pay attention to thread-level parallel processor architecture. Simultaneous Multi-threading (SMT) and Chip Multi-Processor (CMP) [10] are two of the most excellent work on the thread-level parallel processor architecture. Intel's Hyper-threading technology belongs to the category of SMT and multi-core processor is the popular name of processor using CMP technology. Herein, pc, notebook, workstation or server configured with CMP, SMT-CMP mixed or SMP-CMP mixed processor is generally called multi-core computer or multi-core platform.

As the CMP architecture has been partitioned into multiple cores and each one is relatively simple, it's easy to be extended and has a very bright future. Intel and AMD has brought their dual-core and quad-core processors into market. Intel's Tera-scale Computing Research Program had published an 80-core prototype. If the cores increase according to the Moore's law, there will be processors with more than a thousand cores in just about 16 years [11].

There are two conclusions for applications running on multi-core platform:

Conclusion 1. When multiple applications are executed, the performance on multi-core platform is better than single-core platform with the same clock frequency.

On single-core platform, only one process is executed at any time and multiple processes are executed by turns in the way of time slicing. On multi-core platform, multiple processes are scheduled into different cores by operating system and executed at the same time. Therefore, the waiting time is reduced.

Conclusion 2. Applications without parallelization can't make full use of computing power on multi-core platform.

If an application is not parallelized, the process produced when it is instanced could only be scheduled into only one core. Other free cores can't be occupied and the anticipated performance can't be achieved. Figure 1 proves this conclusion. It shows the task manager of Windows OS when a serial discrete event simulator is executed on an HP server with up to eight cores. Only one core is occupied by this simulator and other cores are almost in free status.

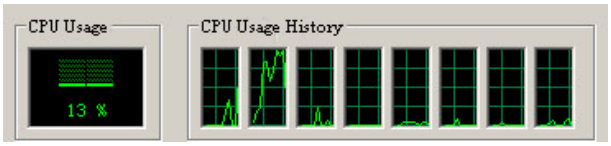

Figure 1. State of task manager when serial simulator is running on HP Server

With the naissance of multi-core processor, the move to parallel software of desktop and laptop systems is just getting underway, so the opportunity to differentiate with parallel performance has never been greater.

3. Optimistic parallel discrete event simulation based on multi-core platform

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.

The prices of super computer and large scale cluster are too high to be afforded, which limits the extensive popularization of PDES. Multi-core platform has the advantages of high performance-price ratio, small purchasing risk, easy moving, high memory access speed, windows OS compatible, easy operation, etc. It will offer a new physical computing platform to the parallel simulation.

In order to parallelize discrete event simulation on multi-core platform, parallel programming model and synchronization algorithm are two of the most important problems to be solved.

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.

Considering the good portability of message passing model, MPI is chosen in this paper.

With MPI adopted, 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.

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. For example, after LP_2 executes an event E_a (with time-stamp 36), LP_1 may execute an event E_b (with time-stamp 15) and generates an event E_c (with time-stamp 21) that LP_2 must execute. Then E_c accesses LP_2 after E_a even though the time-stamp of E_c is smaller than E_a.

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 conservative or optimistic. Optimistic synchronization algorithms are chosen in this paper. 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 (GVT_T) 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.

3.3. Optimistic PDES simulator

Existing PDES simulators commonly run on parallel computers or clusters with Linux or Unix OS. They can't run on multi-core platform with Windows OS directly. A PDES simulator should be newly designed so as to fit the features of multi-core platform. Referring to open-source PDES simulators such as WARPED 2 [14], we 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.

Figure 2 shows the reduced class diagram of this simulator. Four interfaces, i.e. **ICommunicatingEntity**, **ISerializable**, **IConfigurer** and **IConfigurable** are defined in the system. Classes that need to send and receive MPI messages should implement the **ICommunicatingEntity** interface.

Messages that need to be sent through MPI should implement the **ISerializable** interface. Factory classes that implement the **IConfigurer** interface could dynamically create managers according to the external configuration file. Classes that implement the **IConfigurable** interface can configure their attributes according to the external configuration file.

Currently , the abstract class **SimulationManager** has two implementations, i.e. **SequentialSimulationManager** that implements the serial simulation algorithm and **TimeWarpSimulationManager** that implements the optimistic simulation algorithm using Time Warp protocol [6]. **EventSetManager** uses the **deque** data structure to manage the unprocessed events and **vector** data structure to manage the processed events. **SchedulingManager** adopts the minimum time-stamp first method to schedule events. **CommunicationManager** supports two message sending ways, i.e. the direct sending and aggregated sending, and the physical communication layer is implemented using MPI. **StateManager** adopts the periodic state saving method and the period of saving could be set by user. **GVTManager** makes use of Mattern's algorithm to calculate GVT. **TerminationManager** determines whether the events of all LPs have been processed by sending tokens.

Four abstract classes, i.e. **SimulationObject**, **State**, **Event** and **Application**, are provided by the simulator to modelers so as to model entity, state, event and application.

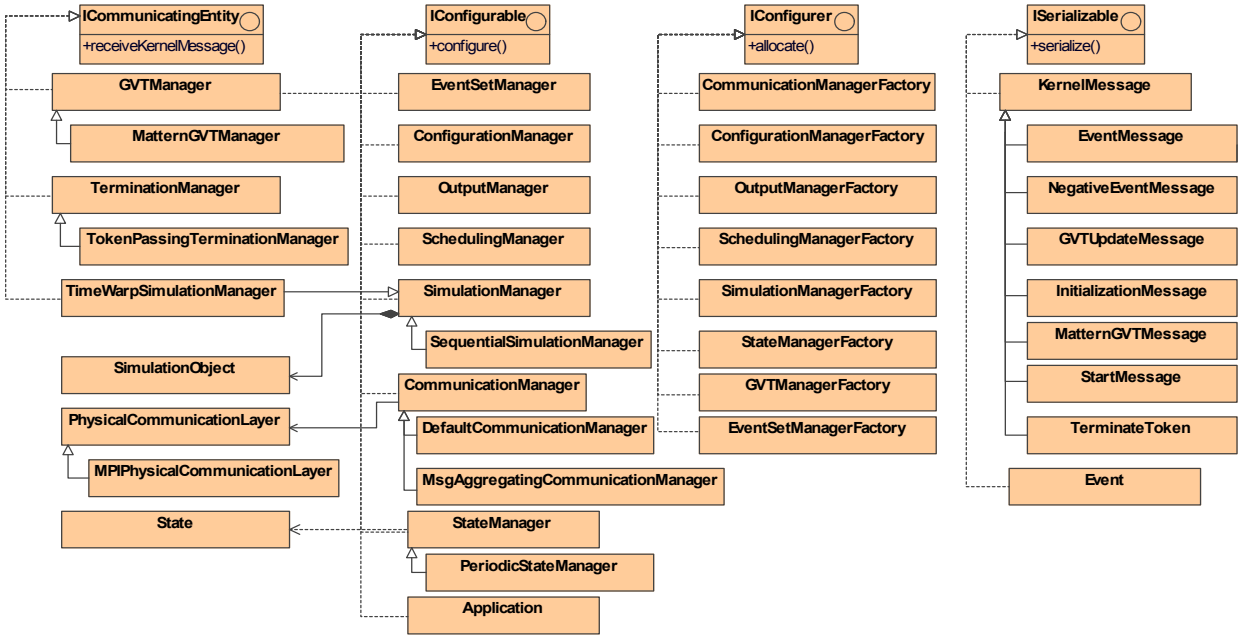

Figure 2. Class diagram of simulator

4. Performance analysis

What speedup will be achieved by running parallel simulation on multi-core platform is the most concerned question. Therefore, the Phold model [15] is developed in this section so as 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 a classical PDES simulator test model with symmetrical load. In this model, there are N entities that are distributed to M LPs and finally distributed to M processor cores. In the initial phase of simulation, each entity generates an initial event. During the simulation, when an entity receives an event, it executes G matrix multiplication operations and then generates a new event. The time-stamp of the new event is determined according to the following rule: if $RAND_{dt}$ is true, the time-stamp of the new event is $LVT + lookahead + dt$, where LVT is the local virtual time of entity, *lookahead* refers to the minimum time between two events, and dt is a random variable that conforms to uniform distribution between 0 and 1; else if $RAND_{dt}$ is false, the time-stamp of the new event is $LVT + lookahead$. The receiver of the new event is determined according to the following rule: if $RAND_{dest}$ is false, the receiver is the next adjacent entity; else if $RAND_{dest}$ is true, then a random number is generated, if this number is smaller than *locality* (a factor that means the ratio of local events to global events), then an entity is randomly chosen from the local LP as the receiver, else an entity is randomly chosen from all LPs as the

receiver. The terminated time of the simulation is T_{end}.

The hardware platform of this test is HP ProLiant ML150 server with two-way Intel Xeon Quad-core processors and 4GB memory. This server has up to eight executing cores, and the clock frequency of each core is 2.0GHz. The operating system of this server is Windows Server 2003. Figure 3 shows the state of task manager when the PDES simulator is running on this server. From this figure, it could be seen that all cores are effectively utilized.

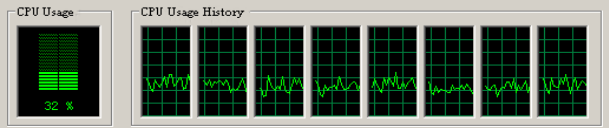

Figure 3. State of task manager when PDES simulator is running on HP Server

Two indexes are usually used to measure the performance of PDES simulator: speedup and event's executing efficiency (we will refer to this as efficiency, for short). Speedup is defined as the ratio of serial running time to parallel running time. Efficiency is defined as the ratio of the submitted events to processed events during the optimistic parallel simulation. The higher the efficiency is, the less the number of rollbacks is.

(1) Overheads of Optimistic PDES Simulator

Table 1. Overheads of PDES simulator

Item	Overhead (second)	Item	Overhead (second)
send initialization message	7.2×10^{-4}	send terminate token	1.0×10^{-3}
send start message	8.4×10^{-4}	calculate matrix multiplication once	2.5×10^{-6}
send event message	9.5×10^{-4}	save entity state once	7.7×10^{-6}
send negative event message	9.8×10^{-4}	restore entity state once	7.7×10^{-6}
send GVT message	1.2×10^{-3}	collect fossil once	1.3×10^{-4}
send GVT update message	9.5×10^{-4}	roll back an event	4.6×10^{-4}

Compared with serial discrete event simulator, there are many other overheads in optimistic PDES simulator, such as sending & receiving messages, saving and restoring the state of entity, collecting fossil, rolling back, etc. Table 1 shows the values of these overheads measured from the test. The overhead of event response is just 2.5×10^{-6} seconds, while the overheads of other items are several times or even tens and hundreds times of the event response. That is one of the reasons why in some case parallel simulation is slower than serial simulation.

(2) Effect of Event Granularity on Performance

Event granularity refers to the processing time of an event. It's the key factor that affects the performance of PDES. Figure 4 shows the speedup versus the event granularity under certain experiment configuration (the configuration parameters are showed in the figure title). As the parameter G increases from 1 to 10^5, the event granularity increases from 2.5×10^{-6} seconds to 7.6×10^{-2} seconds and the speedup increases from 0.27 to 8.06. In general, as the event granularity increases, the ratio of calculation/communication also increases, the communication overhead becomes neglectable, and the speedup increases. Therefore, parallel simulation is suitable for applications with coarse-grained events.

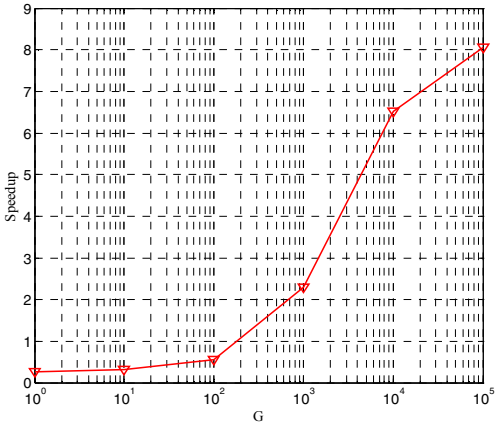

Figure 4. Speedup versus the event granularity (N=M=8, T_{end}=1000, *lookahead*=2, $RAND_{dt}$=$RAND_{dest}$=false)

(3) Effect of Process Number on Performance

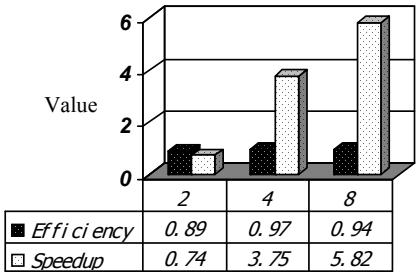

Figure 5. Speedup and efficiency versus process number (N=24, T_{end}=1000, *lookahead*=2, G=5000, $RAND_{dt}$=true, $RAND_{dest}$ =false)

Figure 5 shows the speedup and efficiency versus process number under certain experiment configuration. As the process number increases from 2 to 4 and 8, the efficiency changes little, but the speedup increases observably. In general, the bigger the number of available executing cores is, the more the number of processes that can be created

simultaneously is, the smaller the entity number and event number distributed to one process are, and the bigger the speedup is.

(4) Effect of Lookahead on Performance

Lookahead represents the ability of an LP to predict its future behavior. Lookahead is defined as the minimum time difference in a processed event and an event that it generates. Figure 6 shows the speedup and efficiency versus lookahead under certain experiment configuration. As the *lookahead* increases from 0.2 to 2.2, the efficiency increases from 0.56 to 0.86, and the speedup increases from 2.72 to 4.42. In general, as the lookahead increases, the possibility of receiving straggler events by entities decreases, the number of rollbacks decreases, so the efficiency increases and the speedup also increases.

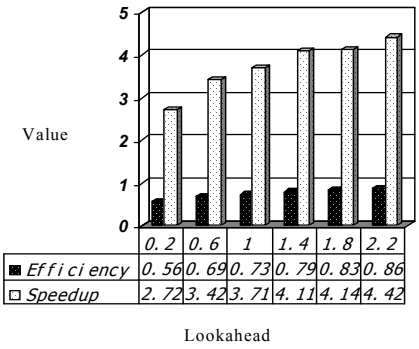

Figure 6. Speedup and efficiency versus lookahead (N=M=8, T_{end}=1000, G=5000, $RAND_{dt}$=true, $RAND_{dest}$=false)

5. Conclusion

The experiment results show that the optimistic PDES based on multi-core platform could achieve good speedup for applications with coarse-grained events. Compared with time-stepped execution formalism showed in Ref. [2,3], event-driven execution showed in this paper will have the potential to produce larger speedup.

With the increasingly advancing development of multi-core processor, the available executing cores will become more and more, and the performance of PDES based on multi-core platform will become better and better. PDES based on multi-core has a good prospect. Parallel simulation that could only be run on super computer and large scale cluster scale formerly is hopeful to be applied to desktop simulation software.

References

[1] H. Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's Journal,* vol. 30, pp. 16-22, 2005.

[2] J. Donald and M. Martonosi, "An Efficient, Practical Parallelization Methodology for Multicore Architecture Simulation," *IEEE Computer Architecture Letters,* vol. 5, pp. 14-17, 2006.

[3] K. Wang, Y. Zhang, and H. Wang, "Parallelization of IBM Mambo System Simulator in Functional Modes," *ACM SIGOPS Operating Systems Review,* vol. 42, pp. 71-76, 2008.

[4] V. Hahanov, V. Obrizan, A. Gavryushenko, and S. Mikhtonyuk, "Parallel Logic Simulation using Multi-Core Workstations," in *CADSM*, Polyana, UKRAINE, 2007, pp. 256-257.

[5] J. Chen, M. Annavaram, and M. Dubois, "SlackSim: A Platform for Parallel Simulations of CMPs on CMPs," Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, USA, Technical Report CENG-2008-6, 2008.

[6] R. M. Fujimoto, "Parallel and Distributed Simulation," in *Proceedings of the 1999 Winter Simulation Conference*, Phoenix AZ, 1999, pp. 122-131.

[7] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*. New York: John Wiley &Sons, Inc, 2000.

[8] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering,* vol. 5, pp. 440-452, 1979.

[9] D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems,* vol. 7, pp. 404-425, 1985.

[10] S. Akhter and J. Roberts, *Multi-Core Programming:Increasing Performance through Software Multi-threading*. US: Intel Press, 2006.

[11] 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.

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

[13] MPICH. MPICH Home Page[Online]. Available: http://www-unix.mcs.anl.gov/mpi/mpich1/.

[14] 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 *Proceedings of the 36th Annual Simulation Symposium*, Orlando, Florida, USA, 2003, pp. 216-223.

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