Parallel Simulation Based on Multi-core Platform: A Review

Shirish V. Pattalwar[1], Prof. Dr. Vilas M. Thakare[2], *Manzoor G. Ahmed*[3]
[1]*Deptt.Of Electronics and Telecomm.,*
Prof. Ram Meghe Institute of Tech. & Research, Badnera-Amravati
shirishpattalwar@rediffmail.com
[2]*Deptt. Of Computer Science*
SGB Amravati University, Amravati
vilthakare@yahoo.co.in
3. Dr.N.P.Hirani Institute of Polytechnic, Pusad
labeeb007@rediffmail.com

ABSTRACT:

Multiple core designs dominating 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, Parallel simulation is always been a first choice for speeding up simulations with multi-core computing platform. In this paper we have presented the research on parallel simulation based multi-core architecture. First we present a survey of existing simulators and simulation methodologies for parallel simulation based on multi-core and finally we present some of the challenges and recommendations to encourage research in designing of parallel simulator based on multicore platform that could be run with windows OS directly.

Index Terms—Parallel simulation, multicore platform, PDES simulators, parallel simulation based on multi-core platform.

I. INTRODUCTION

The motivation for this survey is the proliferation of multi-core, many-core and multi-core cluster architectures, which have inspired us to explore existing simulation methodology and simulator for parallel simulation based on these platform. Parallel hardware has been common in, e.g., server environments for a long time already but more recently also client platforms (i.e., desktop and laptop computers) have been adopting multi-core processors. Furthermore 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 are expected to be executed very fast. Even if the modeled physical systems are becoming more and more complicated parallel simulation is an effective way to speed up the running of 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. The prices of traditional super computer and large scale cluster are too high to be afforded, which limits the extensive popularization of parallel simulation specially PDES in simulation software developer community or general research community. 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. However, writing 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 [2]. Also maintaining causality i.e. future event should not be executed before occurrence of past event during parallel

execution is the central challenge both for the correctness of the simulation and for achieving good simulation performance [3].

The Multi-core processor has come into the market for just about few years, and according to so-called new Moore's Law, the number of cores per chip wills double every 2 years [4, 11]. If this holds true, multi-core machines will soon evolve to many-cores, with 10s if not 1000s 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 [5]), Intel's 80-core (Polaris prototype [6]), IBM's 80 core (Cyclops-64[7]), Ambric's 336 core (Am2045[8]), IBM's four PowerPC 450 cores (BlueGene/P [9]) supercomputer product and even graphics engines with 960 cores (NVIDIA Tesla S1070 [10]). 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. Remaining of paper is arranged as section-II: *a* brief overview existing simulation methodologies and simulators for parallel simulation based on multi-core platform and section-III: challenges and limitations for parallel multi-core simulator IV: recommendations to encourage research in designing of parallel simulator based on multi-core platform.

II. SIMULATING METHODOLOGIES & SIMULATORS

Parallel simulation is a vast field employing countless techniques and methodologies based on various computer architecture platforms. In this section, we present an overview of the popular parallel simulation methodologies and corresponding parallel simulators based on multi-core platform.

A. Multi-threaded methodology
B. Multi-scheduler methodology
C. Slack simulation
D. Two-Phase Trace-driven Simulation
E. Parallel Discrete Event Simulation

A. Multi-threaded Methodology

In this approach POSIX threads (pthreads) are used where each POSIX thread simulate the activity of an executing core. J. Donald and M. Martonosi et al used this methodology [2] to convert existing uniprocessor simulators into parallelized multiple-core simulators. Original uniprocessor simulator effectively clones itself into duplicate cores by calling pthread_create. Each core then runs its own simulation in parallel with other cores. This way, the simulator code retains the same structure as the original uniprocessor simulator. This work parallelizes simulators to run on a shared memory host wherein a core cannot access a shared resource if any other cores have not yet passed that timestamp. Thus synchronization is done on a per-cycle basis, although invoked only when shared resources are accessed.

Many simulators written in sequential languages poses characteristic that much of the simulator state is shared across global variables. Since many of these variables should instead have multiple copies to reflect the states of multiple cores, this poses an initial problem which is being exploited by a language construct known as thread-local storage (TLS) [13].Thread-local storage is supported

on many development platforms such as gcc, Microsoft Visual C++,Borland C/C++ Builder and Intel C/C++ compiler [13]. TLS is not, however, a central tenet of this methodology, but rather a useful implementation trick specific to multithreading in languages with global variables such as C and C++.

* ***Turandot (PTCMP)*****:** Turandot (PTCMP) is first parallel multi-core simulator based on Turandot [17]. PTCMP is derived by converting Turandot trace-driven uniprcessor simulator by employing multithreaded simulation methodology. PTCMP supports simultaneous multithreading, parallel benchmarks[14], heterogeneous cores, and frequency scaling all within its parallel framework. PTCMP is the faster simulator in all cases because of Turandot's extensive use of predecoded information [15] and because the overhead of functional modeling is avoided in a trace-driven framework. However, because this simulator is trace-driven it becomes I/O-bound when simulating a moderately large number of input programs. Thus, PTCMP able to achieve at most 1.5X speedup and performance decreases beyond three nodes because of congestion in the HyperTransport channels.

B. Multi-Scheduler Methodology

In a multi-scheduler methodology, simulation engine is implemented as a user-level thread-scheduler. A scheduler sorts and schedules all modules (jobs). Each job is regarded as a user-level non-preemptive thread on scheduler. Multiple schedulers created, each of which runs/executed on a dedicated host thread.

** P-Mambo (Parallel Mambo:*

A Full System Simulation Environment: Parallel Mambo is a multi-threaded implementation of Mambo. Mambo [20] is IBM's full-system discrete event-driven simulator which models PowerPC systems, and provides a complete set of simulation tools to help IBM and its partners in pre-hardware development and performance evaluation for future systems. Mambo's simulation engine is implemented as a user-level thread-scheduler. Mambo simulates target systems on a single host thread or Mambo is implemented as a sequential simulator because there is only one scheduler to schedule Mambo's modules (jobs). When the number of cores increases in a target system, Mambo's simulation performance for each core goes down. So parallelizing Mambo by creating multiple scheduler (multi-scheduler methodology), each of which runs on a dedicated host thread resulted in parallel multi-core simulator P-Mambo [18]. The first version of P-Mambo implemented in functional modes. Some benchmarks have been tested to evaluate the performance of P-Mambo. The benchmark set is the OpenMP implementation of NAS Parallel Benchmark (NPB) 3.2 [19]. The host machine is an IBM Blade Center LS21, which has two dual-core AMD Opteron 275 processors and 8GB memory. The target machine is a 4-core PowerPC machine with 6GB memory. The target OS is linux 2.6.16(ppc64), while the host OS is linux-2.6.18(x64 64). P-Mambo is a full-system simulator, the whole simulation time of a benchmark includes overhead of booting OS. The speedup with overhead is calculated by the whole simulation time of a benchmark, while the speedup without overhead is calculated by the pure workload simulation time of a benchmark. So P-Mambo achieves the maximum and average speedups (without overhead) of 1.9 and 1.8 respectively when running on two host threads and 3.8 and 3.4 respectively when running on four host threads.

C. Slack Simulation:

In slack simulations, the simulated cores do not necessarily synchronize after every simulated cycle, but rather they are granted some slack. *Slack* is defined as the cycle count difference between any two target cores in the simulation. Small slacks such

as a few cycles greatly reduce the amount of synchronization among simulation threads and thus improve the simulation efficiency with no or negligible simulation errors [16].There exist numerous slack simulation schemes , four are to mention here specifically.

i) Cycle-by-cycle ;(zero slack) all threads must synchronize after every simulated cycle.

ii) Quantum-based; all core threads must synchronize every three cycles, i.e. 3-cycle quantum and a 2-cycle slack.

i) *Bounded slack;* the maximum slack among threads is bounded (the slack is kept below a preset number of cycles).

ii) Unbounded slack; the bound on the slack is the entire simulated time.

Slack simulation offers new trade-offs between simulation speed and accuracy. Slack simulation accelerates the parallel simulation of CMPs by relaxing the tight synchronization enforced between simulation threads in cycle-by-cycle (cycle accurate) simulation.

**** SLACKSIM (Slack simulator):***

SlackSim is Parallel simulator used to evaluate CMP (chip multiprocessor)) target by employing slack simulation schemes.CMP consist of multiple cores on a die, where each core has a private L1 data/ instruction caches and all cores on a die share a large L2 cache. The L2 cache is typically organized as a set of banks with non-uniform access times (NUCA [23-24]). Banks can be shared or private per core.

In SlackSim, both target and host systems are CMPs and simulations are parallelized using the POSIX Threads programming model [21] and simulation environment is built on top of Linux. The general framework of SlackSim is made of two types of Pthreads: several core threads and one simulation manager thread. A core thread simulates a single target core of a CMP with its L1 caches. The simulation manager thread has two functions. Its first function is to simulate the on-chip lower-level cache hierarchy including L2 cache banks and their interconnection to cores. Its second function is to orchestrate and pace the progress of the entire simulation. The simulation pace is controlled by two variables shared by each core thread and the simulation manager thread: *local time* and *max local time*. A core thread increments its local time after every simulated clock cycle of its target core. The max local time of *each core* is set by the simulation manager thread in accordance with the slack simulation scheme. A core thread can advance its own simulation and local time for as long as its local time is less than or equal to its max local time. It suspends itself when the local time reaches the max local time. The simulation manager thread maintains the global time, which is equal to the smallest local time of all core threads. As the global time increases, the simulation moves forward. The simulation manager thread synchronizes the progress of the simulation by setting the max local time of each core thread. SlackSim takes advantage of efficient data sharing on CMP platforms.

D. Two-Phase Trace-driven Simulation *(TPTS)*:

Two-Phase Trace-driven Simulation proposed in [29] splits detailed timing simulation into a trace generation phase and a trace simulation phase. Much of the simulation overhead caused by uninteresting architectural events is only incurred once during the cycle-accurate simulation based trace generation phase and can be omitted in the repeated trace-driven simulations. The goal of this methodology is to facilitate fast testing of system design ideas before undertaking more expensive full-system simulation.

* ***TSIM****: tsim*, a prototype multi-core processor simulator is developed based on the TPTS framework. It models a tile-based multi-core processor having a two-level memory hierarchy,

interleaved memory controllers, a directory-based coherence protocol, and a 2D-mesh network, *tsim* is capable of simulating a multiprogrammed workload (composed of independent threads) or a shared-memory multithreaded workload (composed of data-sharing, synchronized threads) When 16 threads were modeled, *tsim* achieved the simulation throughput of 146.5 MIPS (millions of simulated instructions per second) on a commodity Linux box.

E. PDES(Parallel Discrete Event Simulation):

Parallel Discrete Event Simulation (PDES) is a 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 [26]. 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. 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. By using Parallel programming model simulation is partitioned into multiple LPs and distributed these LPs among executing cores on multi-core platforms for running. 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.

Unfortunately, events can't be ensured to access LPs in time-stamp order which leads to problem called synchronization 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 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. In the conservative approach, the simulation of each message (event) is blocked until it is verified that the event is *safe* [3].

* ***Optimistic PDES simulator****:* Referring to open-source PDES simulators such as WARPED 2 [27] Nian-le Su and his team et al [29] developed a PDES simulator which can run effectively on multi-core computer with Windows OS. They have adapted the Message passing model with Optimistic synchronization approach. Simulation environment is formulated using C++ language and MPICH [25] message passing library.

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. To analyze both the overheads of the parallel simulator and the effects of event granularity, process number, lookahead on the simulation performance the Phold model [28] is developed which is a PDES simulator test model with symmetrical load. Optimistic PDES based on multi-core platform achieved good speedup for applications with coarse-grained events. Compared with time-stepped execution formalism showed in [12].

III. CHALENGES AND LIMITATIONS

From our survey, we found the following limitations and challenges that the multi-core parallel simulators face.

i) One of the main challenges in simulating multicore processors is balancing portability of the simulator with the ease of using and extending the simulator. This challenge can be solved by identifying and using a good programming model for multicore and cluster simulation hosts. The key idea behind such a programming model should be exploiting local multiprocessor as well as cluster computing power

ii) Next to increasing the level of abstraction, another key challenge for architectural simulation in the multicore or manycore era is to parallelize the simulation infrastructure in order to take advantage of increasing core counts. One of the key issues in parallel simulation though is the balance of accuracy versus speed. Cycle-by-cycle simulation advances one cycle at a time, and thus the simulator threads simulating the target threads need to synchronize every cycle. Whereas this is a very accurate approach, its performance may be reduced because it requires barrier synchronization between all simulation threads at every simulated cycle. If the number of simulator instructions per simulated cycle is low, parallel cycle-by-cycle simulation is not going to yield substantial simulation speed benefits and scalability will be poor.

iii) In order to take advantage of multi-core architectures it is necessary not only to coordinate multiple threads of execution, but also to be conscious of memory management issues such as cache efficiency, garbage collection and thread safety.

iv) Most of existing parallel simulators run on clusters with Linux or UNIX. The prices of traditional super computer and large scale cluster are too high to be afforded, which limits the extensive popularization of parallel simulation specially PDES in simulation software developer community.

IV. RECOMMENDATION

Based on the aforementioned observations, we make the following recommendations for fostering research in the area of Parallel simulators based on multicore platform.

Recommendation 1**:** *The choice of programming model:*

As identified in the challenges, the choice of programming model is a key challenge for developing simulators for multicore simulation hosts. It is also essential that to ensure scalability of the simulation host, such programming model should be seamlessly extensible to a cluster of computers. Streaming programming models based on well established process calculi such as Communicating Sequential Processes may be the solution to this issue.

Recommendation 2**:** *Use of Parallel Simulation Techniques for Current Simulation Hosts:*

It is essential to note that as multicore machines are growing more complex, the simulation hosts are also becoming more powerful. Over the last few years, even desktop computers with two or more processors/processor cores have become available to the general public [30], [31]. Simulator designers should take note of this, and research simulators using Parallel Discrete Event Simulation (PDES) that could be run on desktop with windows OS directly.

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] Z. 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

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

[4] 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.

[5] http://www.tilera.com/pdf/ProductBrief Tile64 Web v3.pdf.

[6] 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.

[7] Juan del Cuvillo, Weirong Zhu, Ziang Hu, and Guang R. Gao. "Toward a softwareinfrastructure for the cyclops-64 cellular architecture". In Proceedings of the 20th In-ternational Symposium on High-Performance Computing in an Advanced Collaborative.Environment (HPCS'06), volume 0, page 9, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[8] 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.

[9] http://www.nvidia.com/object/product tesla s1070 us.html.

[10] http://www.intel.com/technology/architecture/downloads/quad-core-06.pdf.

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

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

[13] "Thread-localStorage," http://en.wikipedia.org/wiki/Thread Local Storage, 2006.

[14] J. Donald and M. Martonosi, "Power Efficiency for Variation-Tolerant Multicore Processors," in *Proc. of the Intl. Symp. on Low Power Electronics and Design*, Oct. 2006.

[15] M. Moudgill, J.-D. Wellman, and J. H. Moreno, "Environment for PowerPC Microarchitecture Exploration," *IEEE Micro*, vol. 19, no. 3, pp. 15–25, May/June 1999.

[16] J. Chen,Murli A. ,M. Dubois, *"Exploiting Simulation Slack to Improve Parallel Simulation Speed"* Department of Electrical Engineering - Systems, University of Southern California.

[17] M. Moudgill, J.-D. Wellman, and J. H. Moreno, "Environment for PowerPC Microarchitecture Exploration," *IEEE Micro*, vol. 19, no. 3, pp. 15–25, May/June 1999.

[18] 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

[19] http://www.nas.nasa.gov/Resources/Software/npb.html.

[20] P. Bohrer, M. Elnozahy, A. Gheith, C. Lefurgy,T. Nakra, J. Peterson, R. Rajamony, R. Rockhold, H. Shafi, R. Simpson, E. Speight, K. Sudeep, E. V.Hensbergen, and L. Zhang. Mambo – A Full System Simulator for the PowerPC Architecture. ACM SIGMETRICS Performance Evaluation Review, 8–12, March 2004.

[21] 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 2008

[22] P. Bohrer, M. Elnozahy, A. Gheith, C. Lefurgy,T. Nakra, J. Peterson, R. Rajamony, R. Rockhold, H. Shafi, R. Simpson, E. Speight, K. Sudeep, E. V.Hensbergen, and L. Zhang. Mambo – A Full System Simulator for the PowerPC Architecture. ACM SIGMETRICS Performance Evaluation Review, 8–12, March 2004.

[23] H. Dybdahl P. Stenstrom,"An Adaptive Shared/Private NUCA Cache Partioning Scheme for Chip Multiprocessors, " in *Proc. of the Int. Symposium on High Performance Architecture (HPCA* 2007

[24] J. Huh et al.,"A NUCA Substrate for flexible CMP Cache Sharing," *IEEE Transactions on Parallel and Distributed Systems*, Vol.18 No.8, August 2007, pp.1028-1040.

[25] http://www-unix.mcs.anl.gov/mpi/mpich1/.

[26] 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"

[27] 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.

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

[29] Hyunjin Lee, Lei Jin, Kiyeon Lee, S. Demetriades, M. Moeng," Two-phase trace-driven simulation (TPTS): a fast multicore processor architecture simulation approach". *Exper*. 2010; Published online 21 January 2010 in Wiley Inter Science

[30] Intel Multi-core Website, http://www.intel.com/multi-core/

[31] J. Eker *et al*., "Taming heterogeneity–the Ptolemy approach" In Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, vol. 91, pp. 127-144, Jan 2003.