Simulation Modelling Practice and Theory 28 (2012) 55–64

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Core frequency adjustment to optimize Time Warp on many-core processors

Patrick Putnam, Philip A. Wilsey [*], Karthik Vadambacheri Manian

Experimental Computing Lab, School of Electronic and Computing Systems, PO Box 210030, Cincinnati, OH 45221-0030, USA

ARTICLE INFO

Article history:
Received 2 January 2012
Received in revised form 19 May 2012
Accepted 21 May 2012
Available online 25 July 2012

Keywords:
Parallel simulation
Time Warp synchronization
Many-core processors
Threaded execution

ABSTRACT

Time Warp synchronized parallel discrete event simulators are organized to operate asynchronously and aggressively without explicit synchronization between the concurrently executing simulation objects. In place of an explicit synchronization mechanism, the concurrent simulators implement an independent but common virtual clock model and a rollback/recovery mechanism to restore causal order when out-of-order events are detected. When the critical path of execution of the simulation is balanced across this parallel threads of execution, this can result in a highly effective, lightweight synchronization mechanism to implement parallel simulation. However, imbalances in the workload across the threads can result in excessive rollback in some threads and slowed progress of the critical path. On small shared memory multi-core systems, a lowest time-stamp scheduling policy can effectively balance the workload. However, on larger many-core chips, conventional load balancing and workload migration will once again become necessary. Fortunately, emerging many-core chips contain some interesting features that can potentially be exploited to improve the performance of parallel simulations. In particular, the recently developed Intel Single-chip Cloud Computer (SCC) provides mechanisms for the runtime control of the frequency and voltage settings of the chip. Furthermore, the frequency *and* voltage settings are independently set within different regions (called islands) of the chip. Thus, in a Time Warp simulation, one could increase the frequency of the cores executing threads on the critical path (those experiencing infrequent rollback) and decrease the frequency of the cores executing threads off the critical path (those experiencing excessive rollback). This paper investigates the run-time control and adjustment of core frequency in some contemporary x86 multi-core processors to identify the platforms that can support the exploration of dynamic run-time control of core frequency settings. The results show that while all multi-core processors have software controllable core frequency modulation capabilities, they are generally not fully independent as the system comes under load and are therefore unsuitable for these studies. Fortunately, one processor, the AMD X6 line, provides software control for core frequencies that can be fixed (by software) even as the system operates under load.

1. Introduction

Multi-core processors have been available for quite some time now. As the years have progressed, the number of processor cores has continued to increase. At the time of writing, both AMD and Intel offer desktop computer processors that have 6-cores on a single chip, namely the AMD Phenom II X6 and Intel Core i7-980 Gulftown. It is only a matter of time before there

[*] Corresponding author. Tel.: +1 513 556 4779; fax: +1 513 556 7326.
E-mail addresses: putnampp@gmail.com (P. Putnam), wilseypa@gmail.com (P.A. Wilsey), vadambkk@mail.uc.edu (K.V. Manian).

1569-190X/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2012.05.011

are 8-, 10-, or 12-core desktop processors available. At the same time, there are research platforms available with significantly more processors on a single chip. For example, the Intel Single-chip Cloud Computer (SCC) [1–3] platform offers researchers with 48 IA cores on a single chip. Similarly, the Tile-Gx family of processors from Tilera offers 100-cores per chip [4].

These emerging many-core processors contain some interesting features that can potentially be exploited to improve the performance of parallel applications. In particular, the research many-core SCC processor released by Intel contains: (i) on-chip low-latency message passing hardware, (ii) software managed cache coherence, and (iii) mechanisms for the software regulation of frequency and voltage settings of the on-chip processing cores, interconnection network, and memory controllers [1,2]. In the SCC chip, the frequency and voltage can be independently controlled among various sub-regions (called *frequency islands* and *voltage islands*) of the chip. Using the on-chip thermal sensors to ensure safe setup, application programs can attempt to dynamically adjust the operating frequency and voltage of the chip components to optimize run-time performance.

This paper studies the development of a platform using contemporary x86 multi-core processing chips to explore the run-time adjustment of frequency in the individual processing cores on the chip. This has turned out to be surprisingly difficult and locating documentation on the full of operation of the power states has proven challenging. While all ACPI compliant multi-core processor chips have the standard well-defined power states (C-states, D-states, and T-states), the response of these chips to settings of those states are often ignored by the hardware as the system comes under load. Thus, when the system is idle the power states are well-defined and highly effective for minimizing power consumption. However, when the system is operating under load, experimental data shows that dynamic (software) control of core frequencies to throttle one core and overclock another core becomes impossible on most multi-core processors. Fortunately one contemporary multi-core processor, the AMD Phenom II X6, does adhere to the software power settings even when the system is under load. This paper presents the results of these investigations and outlines the application of dynamic frequency control to optimize parallel simulations using the Time Warp synchronization protocol [5,6].

The remainder of this manuscript is organized as follows. Section 2 presents the motivating factors for dynamic frequency control including a description of how frequency control can address optimizing Time Warp synchronized parallel simulations. Section 3 provides some background information about the frequency and power states and their control by software in contemporary x86 multi-core processors. In addition, a review of the Intel SCC processor and its voltage and frequency adjustment capabilities is presented. Section 4 briefly describes some experiments running parallel simulation on an emulation environment for the Intel SCC (many-core) platform. Several simulation models are run and projections of the theoretical increases available from an idealized frequency modulation are explored. Section 5 describes the experimental platforms and software codes that were developed to explore dynamic frequency control. Section 6 reviews the results of the experimental analysis and presents some general discussions of the findings from these studies. Finally, Section 7 contains some concluding remarks and suggestions for future research.

2. Motivation

Migrating parallel programs from multi-core to many-core processors will likely require refactoring to alleviate possible contention to shared resources and move some of the communication events from shared memory to messaging using the on-chip network subsystem. Load sharing and process migration on many-core processors will be more difficult, slow, and computationally expensive. Thread partitioning and core assignment will also become more significant in the effective deployment of parallelism to many-core processors. However, one important new tool to help in this matter is the integration of software controlled frequency adjustment on many-core processors. These adjustments allow software to overclock some cores and underclock others—sometimes by dramatic amounts. For example, on the Intel SCC processor, core frequencies can be adjusted from 300 MHz to 1.3 GHz. Effectively managed, and ensuring that thermal limits are maintained, the cores in the system cores can be underclocked and overclocked to balance the load and potentially accelerate the critical path of execution. This will *not* replace the need for effective partitioning and task assignment/scheduling. Instead it is a potential refinement to further improve total system throughput. Of course the challenge is to learn which cores have threads on the critical path of execution and which are not. While this may not be easily achievable for all application, there are certainly applications for which this should be possible. In particular, as outlined below, Time Warp synchronized parallel simulations are an ideal candidate for this approach.

2.1. Time Warp

The Time Warp mechanism is an optimistic synchronization mechanism for parallel and distributed discrete event-driven simulation [5,6]. Under Time Warp, the discrete event simulation is decomposed into a collection of concurrently executed discrete event simulators called Logical Processes (LPs), Fig. 1. Conceptually, each LP maintains a local simulation time (called the *local virtual time* or LVT) and each LP processes events, in their time-stamp order, without regard to the progress of other LPs in the system. When an event is processed, it may generate one or more time-stamped events that are distributed to the designated LPs (by message passing or by direct insertion into the receiving LPs input event queue). When an LP receives an input event that is in its simulated past (the event's timestamp is less than the LVT of the LP), the LP will rollback to a previously saved state and re-process the events in their proper order. Such events are called *straggler* events (or mes-

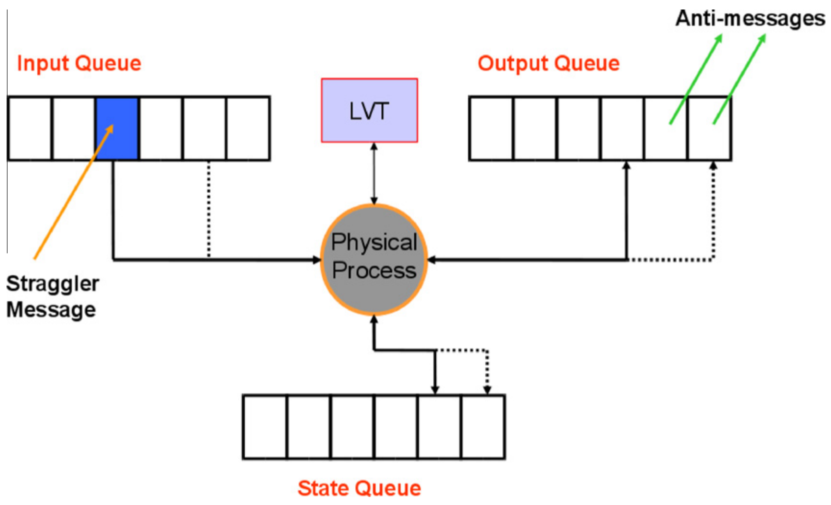

Fig. 1. A Logical Process in a Time Warp Simulation.

sages). If necessary, prematurely sent output events will be terminated by distributing *anti-message* events that tell the receiving LP to erase the premature event (which might also trigger secondary rollback from the receiving LP).

On a shared memory machine, a common configuration for a Time Warp simulator is to have a common event list for the worker threads organize their behavior to process events in time-stamp order across all LPs in that memory space. Thus, insuring that the worker threads balance the work and mostly follow the critical path of causal order of the events. When migrating to a many-core solution, the common event list becomes a point of contention and alternate organizations with multiple event lists for distinct subsets of the LPs are deployed. In this case, one or more worker threads on each core are assigned to a, time-stamp ordered, event list and some form of messaging is used to exchange event information between the distributed event lists. Likewise partitioning and assignment of LPs to the distributed event lists may lead to load imbalances that will need to be addressed. While migrating LPs between event lists may become necessary, a complimentary approach is to also adjust processor frequencies to further fine tune performance to accelerate the critical path of the simulation.

2.2. Runtime tuning of Time Warp

Runtime tuning to optimize performance has been successfully applied to a number of subalgorithms of a Time Warp synchronized parallel simulation [7]. Most significantly it has been used for: (i) sizing the checkpoint interval of an LP [8–10], (ii) selecting the cancellation strategy for an LP [11], and (iii) for event scheduling [12–15]. Each of these mechanisms develop and use some runtime measurements to assess performance and guide the tuning algorithms (e.g., rollback frequency vs rollback costs, distance from the global time, effectiveness of premature computations to produce useful work, and so on). From the perspective of dynamic control of core frequency, it appears (from previous work) that the rollback frequency is an indirect measure of an LP's relation to the critical path of execution. LPs with a higher rollback frequency are processing events prematurely and are further off the critical path than LPs with little or no rollback activity. Thus, monitoring the rollback behavior of the LPs provides an indication of how to adjust the local core's clock frequency. This measure will be discussed more fully in later sections of this paper.

3. Background

Dynamic Voltage and Frequency Scaling (DVFS) are computer architecture techniques by which, as the name suggests, the processor voltage and/or frequency can by adjusted to better compensate for the processing needs of the system. Typically these techniques are most commonly used for reducing the power consumption of the system. Both techniques aim to reduce the dynamic power, or *switching power*, consumed by a CMOS gate. The switching power roughly follows $P = CV^2f$ [16], where C is the capacitance of the gate, V is the voltage, and f is the switching frequency. The dynamic voltage technique reduces the voltage portion of this equation, and the dynamic frequency aims at reducing the frequency. Intel's SpeedStep, and AMD's PowerNow! or Cool'n'Quiet technologies are implementations of dynamic frequency scaling (DFS). In a later section, this paper explores the use of these features of multi-core chips to demonstrate dynamic control of core frequency by a running program. Ultimately this will serve as the basis for future studies with dynamically controlled many-core processors.

The open standard called Advanced Configuration and Power Interface (ACPI) [17] provides industry-standard interfaces for Operating System directed configuration and Power Management of devices. All ACPI compliant processors and devices have well-defined power states, C-states and D-states, respectively. The C0 and D0 states correspond to active/operating states. The ACPI standard also defines Performance States, or P-states. These states are power consumption or capability states available while the processor is in state C0 and devices are in state D0. In terms of a processor, the P-states define the different frequency/voltage states it can be in. The number of P-states is variable, and dependent upon the component in question. P0 is the highest performance state, where a component consumes the most energy and has the highest frequency; and Pn is the lowest performance state. The key advantage of P-states is that switching between states is low latency. Finally, the ACPI standard also defines Throttling states, or T-states, which only control processor frequency throttling. However, these states do not generally reduce power consumption [18] and are not generally used by modern Operating Systems.

The Operating System is usually in control of specifying the system P-state, and may allow some level of control to the user. The Linux 2.6 kernel [19] provides access to devices, device drivers, and device configurations through the `sysfs` virtual file system. In `sysfs` there is a subsystem called CPUfreq [20–22] that provides access to the processor configurations of the current system. This subsystem relies on governors to set the processor frequency to specific levels based on certain criteria. As mentioned, the governor only sets the desired frequency of the processor, it is left to the hardware to select the nearest P-state to the desired frequency. There are several governors available with the Linux kernel, namely: Performance, Powersave, Userspace, Ondemand, and Conservative. The Ondemand governor is usually the default governor, and will dynamically adjust the frequency of the processor based on processor load. Of interest to us is the Userspace governor which provides the user the ability to manually select the processor frequency.

As suggested, the primary use for P-states is reducing power consumption of a system when it is not under load. For example, when a system is idling it can be placed into a lower performance state (higher P-state index), thus causing the system to waste less energy while it is not performing computationally intensive tasks. This power savings model has been useful in many applications. However, this model relies on the OS determining when to throttle the processor up or down based on the task load of the system. What if we are not concerned about the power savings, but more about the processor frequency during the execution of a process? What if we allowed a processor bound thread determine the frequency at which the processor should be running?

3.1. Frequency and voltage adjustment in the Intel SCC chip

The Intel SCC platform is an experimental many-core processor developed and distributed to support research into many-core processing. SCC is the first Intel many-core chip with x86 compliant cores on a single die. The die has 48 cores organized into 24 Tiles with 2 x86 cores per Tile (Fig. 2). Each of the 24 tiles contains a dual-core x86 processor, L1 and L2 caches, and

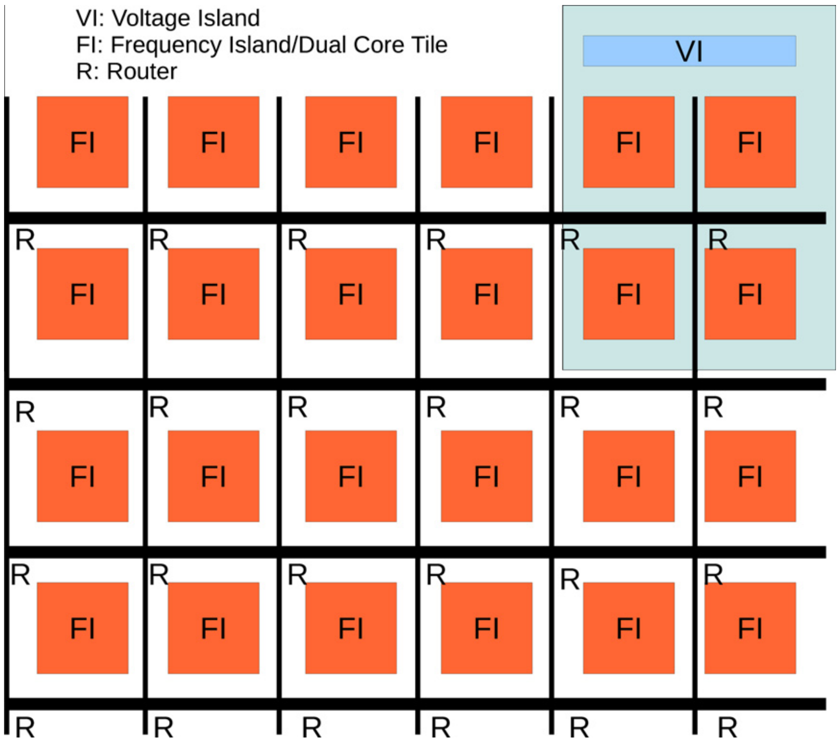

Fig. 2. The Intel SCC chip.

router for sending messages over the 2-D mesh network connecting the tiles. There are four memory controllers on the board, supporting a total of 64 GB of addressable DDR3 memory.

The Intel SCC platform offers an interesting set of features. Of relevance to this body of work are the voltage and frequency throttling features. This platform is a 6x4 grid of tiles, where each tile is essentially a dual-core processor. The chip is divided into 7 voltage domains (called *voltage islands*); 6 for each 2×2 block of tiles, and 1 for the onchip network. There are 28 frequency domains (called *frequency islands*); 24 for each dual-core tile, one for the onchip network, one for the system interface, one for the memory controller, and one for the voltage regulator controller. Within limits defined by the set voltage of a voltage island, the corresponding tiles can be throttled independently of the each other. Basically, the voltage domain specifies a maximum operating frequency for the tiles. The tiles are then able to throttle to that frequency or specific lower frequencies which follow a step function. Frequency changes require only a few cycles to take affect, whereas voltage changes occur on the order of a million cycles.

4. Parallel simulation in the SCC emulation environment

This section focuses on the potential opportunities between Time Warp synchronized parallel simulation and the Intel SCC many-core platform. To pursue these investigations, the WARPED simulation kernel [23] is used. WARPED was developed to support large scale simulations (millions of LPs) on small (32–64 node) Beowulf clusters. WARPED is used 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.

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. An expanded discussion on the work to execute WARPED simulation models on the RCCE environment is presented in [24]. This section summarizes results from that paper and extrapolates the theoretical speedup that could be possible with idealized frequency adjustment.

In these studies, four simulation models packaged with the WARPED simulation kernel are run on the Intel SCC emulation environment executing on a Intel Core2Duo operating at 2.00 GHz, with 3 Gb of RAM, and running Linux (version 2.6). Due to size considerations, the emulated SCC platform was configured as 4 cores (or two tiles). In these experiments, the WARPED simulator was configured with 4 LPs each bound to one of the emulated SCC cores. The four simulation models are a standard parallel simulation model called PHOLD [5], a model of a RAID-4 disk array, and a model of a memory traffic in a shared memory multiprocessor.

In this analysis, we assume that: (i) the operating frequency of all processing cores can be adjusted up or down by any fractional amount; (ii) the frequency adjustments must be balanced, that is any increase must be offset by a corresponding decrease; and (iii) that the total number of rollbacks are uniformly distributed throughout the simulation run and that any frequency adjustment will have a direct correlation with an increase and/or decrease in the rollback frequency. These are highly idealized assumptions, but they should provide an upper bound on the potential performance impact that dynamic frequency adjustment can provide.

The runtime results are shown in Table 1. The results contain the total runtime and rollbacks experienced by each emulated core. In column 5 (Optimal Frequency Adjustment), we computed the target (idealized) frequency changes using these runtime numbers. In particular, the optimal frequency adjustment is computed assuming that the rollbacks were directly correlated to the rollback frequency. Thus, the frequency adjustment was computed as a percent speedup or slowdown (denoted by a negative percent) to match the average number of rollback experienced for all of the LPs for that simulation

Table 1
Simulation results from Core2Duo.

Model	Core	Runtime (s)	Rollbacks	Optimal frequency adjustment	Potential runtime (s)	Potential speedup
PHOLD	0	346.87	2462	16.59%	289.32	1.20
	1	352.06	3794	−28.33%		
	2	352.57	2911	1.38%		
	3	352.76	2640	10.56%		
		Ave:	2951.75			
RAID	0	26.36	381	44.30%		
	1	26.53	175	74.42%	6.79	6.79
	2	26.52	1344	−96.49%		
	3	26.54	836	−22.22%		
		Ave:	684			
SMMP	0	71.92	20,052	−185.00%		
	1	71.91	2359	66.47%		
	2	71.94	1201	82.93%	12.28	5.86
	3	71.92	4531	35.60%		
		Ave:	7035.75			

Table 2
Hardware & Software specifications of the two machines used in testing.

	Intel	AMD
Processor	Xeon W3680	Phenom II X6 1055T
Cores	6	6
Max Frequency	3.33 GHz	2.80 GHz
Hyper-threading	Disabled	N/A
Operating System	Fedora 14	Ubuntu 11.01
Kernel	2.6.35.14–96	2.6.38–6-generic

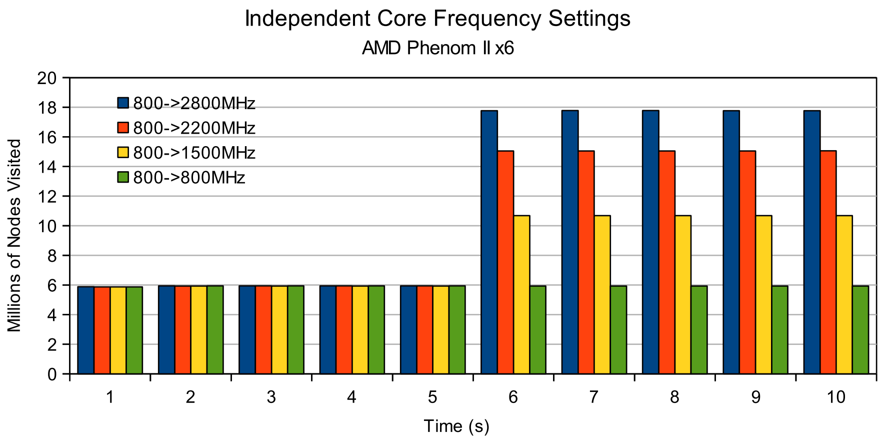

Fig. 3. This is an execution profile upon an AMD Phenom II X6 processor. Four threads are executed over a 10 s period, nodes visited are reported every second, three of the processors throttle to higher frequencies after 5 s.

model. From this, a new completion time is computed for the critical path and then the potential speedup under these idealized conditions is shown. From these results we see that 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. The remainder of this paper explores our experiences evaluating several existing multi-core platforms for use to continue further, more practical dynamic frequency modulation for optimizing parallel simulation.

5. Experiments with dynamic frequency adjustment

In this section we want to accomplish two main goals. First, we set out to establish a multi-threaded application where threads are bound to specific processors and able to specify their own frequency. Second, we endeavored to determine the throttling capabilities of different x86 hardware platforms from Intel and AMD (see Table 2).

5.1. Software

To make the development process simpler, the test application was designed to run on a Linux platform with root level access. Additionally, we assumed there would be a minimal number processes competing for execution time. These assumptions meant that our software would be manually manipulating the system via the `sysfs` interface.

The application followed a four step process: (i) initialize, (ii) bind, (iii) run thread, and (iv) clean up. During (i), the application parses the user arguments, determines the number of available processors, sets their governors to be userspace, and determines the available processor frequencies. Step (ii) attempts to bind each currently running process to a user specified set of processors.

The third step of the application process creates a single processor bound thread for each of the available processors. Each thread controls its processor and measures the processor performance by a circular graph traversal algorithm. The algorithm generates a circular graph with N nodes, throttles the processor to a specified frequency, traverses the graph for a specified period of time counting the total number of nodes visited, and finally cleans up the graph. For the purposes of these experiments, it was sufficient to set $N = 1000$ for all testing scenarios. An independent list of throttling events, or throttling event scenarios, is provided to each thread, where each event is a pair of processing frequency and the loop time duration. Begin and end times are recorded for both the throttling stage and the graph traversal stage. Finally, the "clean up" step returns the system to the pre-initialized state in terms of process binding and processor governors.

5.2. Hardware

Two multi-core processor environments were studied in the experiments. The first system was a workstation built around an Intel Xeon W3680 processor with 6-cores. The second system was a desktop computer built around an AMD Phenom II X6 1055T processor with 6-cores. Although both processors have 6-cores, that is where the similarities end. The Intel processor has an available 15 frequency settings ranging from 1596 to 3326 MHz. The AMD has only 4 ranging from 800 to 2800 MHz.

6. Results

While there were no other computationally intensive applications running on either system during testing, it was decided that the standard Linux system processes should be bound to 2 of the 6 cores. The remaining 4 cores would then be used in the "thread" step of the application. Several different throttling event scenarios were performed on each of the test platforms. Each set of scenarios was repeated 10 times.

Since the AMD and Intel processors have differing frequency ranges and values, it was expected that the number of nodes visited on each platform would be significantly different. When the AMD is running at its maximum frequency (2.8 GHz) it is able to visit roughly 17.7 million nodes per second (Fig. 3). However, when the Intel is running at its lowest frequency (1.59 GHz) it is able visit 16.0 million nodes per second (Fig. 4). The performance difference is also evident in the number of nodes visited per clock cycle. The Intel processor is able to visit roughly 10.2 nodes per 1000 clock cycles, and as the frequency increases so do the number of nodes visited (see Fig. 5). Conversely, the AMD processor starts out visiting more nodes at lower frequencies and decreases as the frequency increases (Fig. 6).

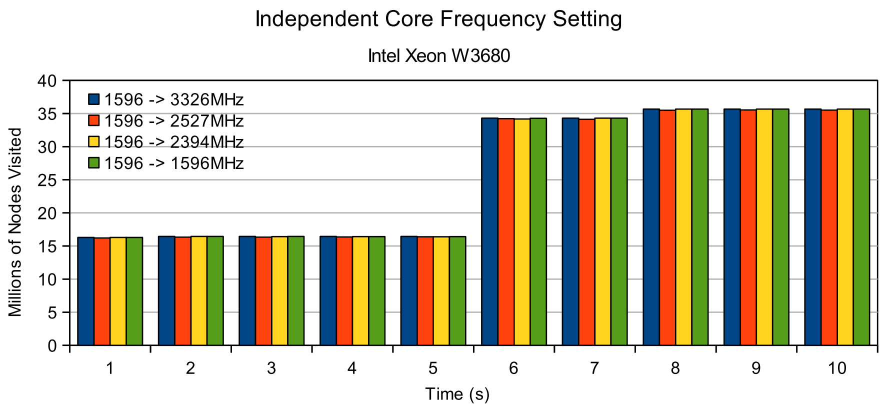

Fig. 4. This is an execution profile upon an Intel Xeon W3680 processor. Four threads are executed over a 10 s period, nodes visited are reported every second, three of the processors throttle to higher frequencies after 5 s.

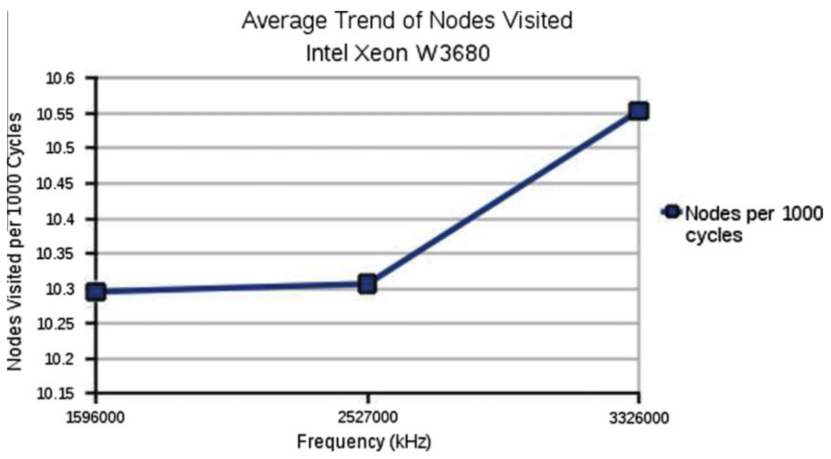

Fig. 5. This graph depict the average number of nodes being visited per 1000 clock cycles on the Intel processor.

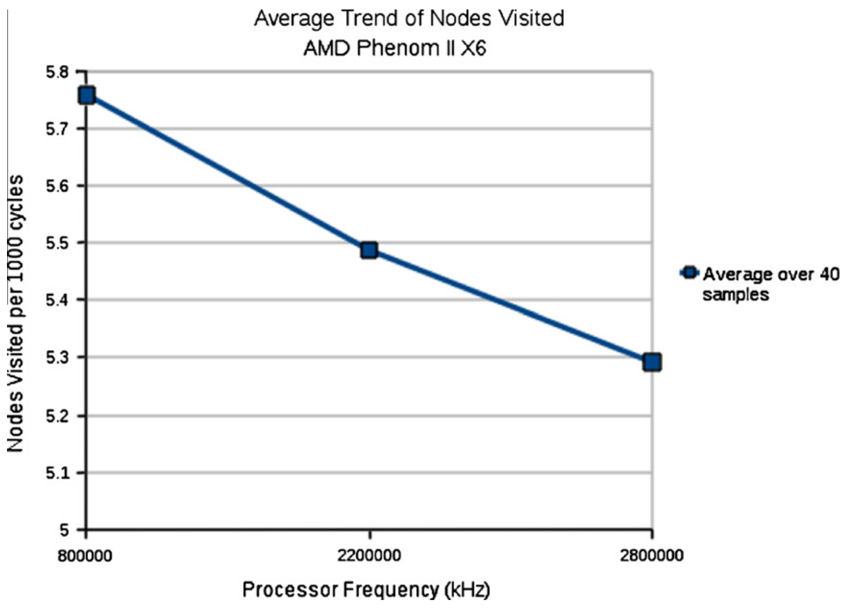

Fig. 6. This graph depict the average number of nodes being visited per 1000 clock cycles on the AMD processor.

The unexpected result was that the number of nodes visited by each thread while running on the Intel platform corresponded to the maximum frequency of all the threads. In other words, if one of the threads throttled itself to the maximum frequency, it does not matter what frequency the other threads throttle to, all of the cores will end up throttled to the maximum frequency. This is clearly shown in Fig. 4 where after 5 s (when the software frequency changes are triggered), all the cores jump to the maximum node visit rates despite being set by the software to different frequencies. Further research indicates that on the Intel platform has a single voltage regulator per socket. "Hence P-state transitions (which impact both frequency and voltage) for all the cores need to happen at the same time" [25]. On the other hand, the AMD is not subject to this same design decision. Fig. 3 clearly shows the cores visiting nodes at different rates after the frequency changes are affected (after 5 s). With the Phenom II X6 line of processors, AMD has "unlocked" the P-states[26], allowing each core to take on its own frequency. Thus, the Intel is unsuitable for experimenting with program controlled frequency throttling to optimize performance. The AMD X6 is, however, capable of supporting software controlled configurations to experiment with dynamic control of core frequency.

6.1. Discussion

As mentioned above, controlling the processors voltage and frequency has been primarily been utilized for power saving purposes. As evidenced by our research and experiments, implementations of this technology varies on common desktop multi-core processor lines. While each processor line implements P-states, the decision on whether all the cores should exist in the same state or be allowed to exist in their own state seems to differ. Our experiments did reveal a significant difference in performance between an Intel and AMD processor. However, it was beyond the scope of our present interests to determine how much of an effect the P-state design decision had on the actual processor performance. We have primarily been focused on utilizing this technology as it will be available on many-core systems, such as the Intel SCC platform. To that end, the AMD processor is better suited for the task. It provides multiple cores on a single chip with the ability to throttle each of the cores individually.

Another interesting feature of the AMD Phenom II X6 processor line is the Turbo CORE technology[27]. This technology provides a set of hidden P-states, called Boost P-states, which only the hardware can control. In addition to current processor P-states, the hardware tracks the load on each core. If a core is in the highest P-state (P0) with a heavy load, then the hardware will attempt to overclock the core for a period of time while remaining with in thermal and electrical limits. The hardware can boost a maximum of 3 cores at the same time. For example, if core 2 of a processor is in a "boost eligible" state on a 2.8 GHz X6, the hardware may attempt to overclock the core to 3.30 GHz for a period of time. In following the trend from our experimental data, assuming that the addition 500 MHz would further decrease the nodes visited per 1000 cycles to just 5 nodes, we would still see an increase of roughly 1.7 million nodes visited.

Further exploration into the use of processor throttling would consist of a couple of tasks. Removing the constraint that the user must have root access to the machine in order to be able to throttle the cores is a first step. There are several daemons available which provide this functionality [28]. Another step is to implement the process into a Time Warp simulation kernel.

7. Conclusion

Many-core and multi-core processing platforms pose interesting challenges and opportunities for optimizing Time Warp synchronized parallel simulation. In a Time Warp simulation, all of the LPs aggressively execute events with those off the critical path experiencing an overly large number of rollbacks due to their over aggressive computation. On small, shared memory multi-core processors, it is possible to use a shared event pool and schedule events in their global least time-stamp order for concurrent execution (to a collection of worker threads). On a larger many-core processor platforms (such as the Intel SCC chip) LP partitioning, load balancing, and network latency issue will become more significant. One feature of many-core platforms that appears to provide an interesting addition to the optimization and balancing of LP execution is the ability to independently throttle (up or down) the frequency of the processing cores on the chip. While this feature does not replace load balancing, it does provide a complimentary mechanism to further refine and tune the execution of Time Warp simulations to optimize performance.

This paper examined the execution of Time Warp synchronized parallel simulations on the Intel SCC emulation framework. From these simulations we assigned LPs to specific cores and observed the total number of rollbacks experienced by each core. Using an idealized theoretical analysis with infinitely selectable core frequency settings, we demonstrated how frequency adjustments could accelerate the critical path for a total speedup of the three simulation models by 1.2, 5.86, and 6.79. While this is an idealized result that is not achievable in practice, this analysis does show that non-trivial speedup numbers could be possible.

The second main contribution of this paper was to explore the feasibility of using existing multi-core chips to study dynamic frequency modulation. It turns out that this is not as easy as one might think. Initially we believed that virtually any multi-core platform would suffice. What we discovered was that only the AMD Phenom II X6 1055T platform would respect software based frequency settings. The other Intel and AMD chips that we studied would not follow the software based frequency settings when all the cores on the chip were under load. Another challenging aspect of using multi-core chips is the relatively coarse settings of the frequencies. Despite this, we are continuing our studies to use dynamic frequency control on the X6 processor to optimize our Time Warp simulator.

Acknowledgment

Support for this work was provided in part by the National Science Foundation under Grant CNS-0915337.

References

[1] J. Howard et al., A 48-core IA-32 message-passing processor with DVFS in 45 nm CMOS, in: 2010 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010, pp. 108–109.
[2] Intel Press Release, Intel Corporation, Futuristic Intel Chip Could Reshape How Computers Are Built, Consumers Interact with their PCs and Personal Devices, Tech. Rep., Intel Press Release, Intel Corporation, December 2009. <http://www.intel.com/pressroom/archive/releases/20091202comp_sm.htm>.
[3] I. Labs, The SCC Platform, Tech. Rep., Intel Corporation, May 2010. <http://techresearch.intel.com/spaw2/uploads/files/SCC_Platform_Overview.pdf>.
[4] T. Corporation, TILE-Gx Processor Family, Tech. Rep., Tilera Corporation, 2011. <http://www.tilera.com/products/processors/TILE-Gx_Family>.
[5] R.M. Fujimoto, Parallel discrete event simulation, Communications of the ACM 33 (1990) 30–53.
[6] D. Jefferson, Virtual time, ACM Transactions on Programming Languages and Systems 7 (3) (1985) 405–425.
[7] A. Palaniswamy, P.A. Wilsey, Parameterized Time Warp: an integrated adaptive solution to optimistic pdes, Journal of Parallel and Distributed Computing 37 (2) (1996) 134–145.
[8] L. Auriche, F. Quaglia, B. Ciciani, Run-time selection of the checkpoint interval in time warp based simulations, Simulation Practice and Theory 6 (5) (1998) 461–478.
[9] J. Fleischmann, P.A. Wilsey, Comparative analysis of periodic state saving techniques in Time Warp simulators, in: Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS 95), 1995, pp. 50–58.
[10] R. Rönngren, R. Ayani, Adaptive checkpointing in Time Warp, in: Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94), Society for Computer Simulation, 1994, pp. 110–117.
[11] R. Rajan, R. Radhakrishnan, P.A. Wilsey, Dynamic cancellation: selecting Time Warp cancellation strategies at runtime, VLSI Design 9 (3) (1999) 237–251.
[12] C. Burdorf, J. Marti, Non-preemptive Time Warp scheduling algorithms, Operating Systems Review 24 (2) (1990) 7–18.
[13] A. Palaniswamy, P.A. Wilsey, Scheduling Time Warp processes using adaptive control techniques, in: J.D. Tew, S. Manivannan, D.A. Sadowski, A.F. Seila (Eds.), Proceedings of the 1994 Winter Simulation Conference, 1994, pp. 731–738.
[14] F. Quaglia, V. Cortellessa, Grain sensitive event scheduling in time warp parallel discrete event simulation, in: Proc. of 14th Workshop on Parallel and Distributed Simulation (PADS 00), 2000.
[15] T. Som, R. Sargent, A probabilistic event scheduling policy for optimistic parallel discrete event simulation, in: Proc. of 12th Workshop on Parallel and Distributed Simulation (PADS98), 1998, pp. 56–63.
[16] Wikipedia, Dynamic Voltage Scaling—Wikipedia, the Free Encyclopedia, 2011. <http://en.wikipedia.org/w/index.php?title=Dynamic_voltage_scaling&oldid=451282154> (accessed 04.10.11).
[17] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation, Advanced Configuration and Power Interface Specification, fourth ed., 2010. <http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf>.
[18] ThinkWiki, How to Make Use of Dynamic Frequency Scaling, 2011. <http://www.thinkwiki.org/wiki/How_to_make_use_of_Dynamic_Frequency_Scaling>.
[19] L.K.O. Inc., The Linux Kernel Archives, Tech. Rep., Linux Kernel Organization Inc., 2011. <http://www.kernel.org>.
[20] D. Brodowski, N. Golde, Linux CPUFreq – CPUFreq Governors, Tech. Rep., Linux Kernel. <http://www.mjmwired.net/kernel/Documentation/cpu-freq/governors.txt>.
[21] J. Hopper, Reduce Linux Power Consumption. Part 1: The CPUfreq Subsystem, Tech. Rep., IBM. <http://www.ibm.com/developerworks/linux/library/l-cpufreq-1/index.html>.

[22] V. Pallipadi, A. Starikovskiy, The Ondemand Governor: past, present, and future, in: Proceedings of the Linux Symposium, 2006, pp. 223–238. <http://www.linuxinsight.com/ols2006_the_ondemand_governor.html>.
[23] D.E. Martin, P.A. Wilsey, R.J. Hoekstra, E.R. Keiter, S.A. Hutchinson, T.V. Russo, L.J. Waters, Redesigning the warped simulation kernel for analysis and application development, in: Proceedings of the 36th annual symposium on Simulation, ANSS '03, 2003, pp. 216–223.
[24] K.V. Manian, P.A. Wilsey, Distributed simulation on a many-core processor, in: The Third International Conference on Advances in System Simulation (SIMUL 2011), 2011.
[25] S. Siddha, Multi-Core and Linux Kernel, Tech. Rep., Intel Inc., 2007. <http://software.intel.com/sites/oss/pdfs/mclinux.pdf>.
[26] S. Wasson, C. Kowaliski, AMD's Phenom II X6 processors. <http://techreport.com/articles.x/18799>.
[27] A.L. Shimpi, AMD Divulges Phenom II X6 Secrets, Turbo Core Enabled, Tech. Rep., AnandTech, April 2010. <http://www.anandtech.com/show/3641/amd-divulges-phenom-ii-x6-secrets-turbo-core-enabled>.
[28] ThinkWiki, How to Configure CPUfreqd, 2011. <http://www.thinkwiki.org/wiki/How_to_configure_cpufreqd>.