Dynamically Adjusting Core Frequencies to Accelerate Time Warp Simulations in Many-Core Processors

Ryan Child and Philip Wilsey
School of Electronics and Computing Systems
University of Cincinnati
Cincinnati, OH
Email: ryan.child@gmail.com, wilseypa@gmail.com

***Abstract*—Time Warp synchronized parallel discrete event simulators are organized to operate asynchronously and aggressively without explicit synchronization between the concurrently executing simulators. In place of an explicit synchronization mechanism, the concurrent simulators maintain a common virtual clock model and implement 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 these parallel simulators, this can result in a highly effective, lightweight synchronization mechanism. However, imbalances in the workload across the parallel simulators can result in excessive rollback at some nodes and ultimately result in an overall slowing of the simulation as prematurely computed and transmitted events are processed. On small shared memory multi-core systems, a lowest time-stamp first 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. For example, the Intel Single-chip Cloud Computer (SCC) provides mechanisms that a running application can use to adjust the frequency/voltage of different regions (called islands) of the chip. These islands are network and processing core centric and thus, in a Time Warp simulation, one can 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 an AMD Phenom II X6 multi-core processor to explore and demonstrate that the dynamic run-time control of core frequency can sometimes improve the performance of a Time Warp synchronized parallel simulation.**

***Keywords*-parallel simulation; time warp synchronization; many-core processors; run time tuning**

I. Introduction

Recent trends have shown that parallel processing is emerging as the new frontier for the mass computing market [1], [2]. Trends in microprocessor development have shown that the shift from multi-core to many-core is on the horizon. The road maps of all the major processor providers (Intel, AMD, Sun, and IBM) clearly show this progression. The Intel i7 processor has hardware support for up to 12 simultaneous threads. IBM announced their next generation Power7 product that is expected to support up to 32 threads per chip [3]. Sun MicroSystems has single chip processors providing hardware support for up to 64 threads. Intel has their "single-chip cloud computer" research chip that contains 48 x86 compatible cores [4], [5], [6] and they have recently announced a planned release of a new 50 core chip to be called Knights Corner [7]. Following these patterns, it is clear that commercial processors may soon contain hardware support providing capabilities for hundreds of simultaneously executing threads.

In addition to increases in the number of parallel cores, these emerging many-core processors contain some interesting features that can potentially be exploited to further 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 [4], [5]. In the SCC chip, the frequency and voltage can be independently controlled among various sub-regions 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 use of run-time frequency adjustment to overclock and underclock different cores across a many-core chip in an attempt to accelerate the critical path of execution. In one sense, this is taking the hardware based turbo-boost concept and folding it into software to dynamically control the CPUs clock rate. The idea is to balance the frequency to accelerate the critical path and preserve the processor's power, current, and thermal limits within safe operating limits. In this paper, we use an AMD Phenom II X6 multi-core processor to demonstrate some of the principles of run-time frequency control in many-core processors. This paper presents the results of our preliminary investigations to use dynamic frequency control to optimize parallel simulations using the Time Warp synchronization protocol [8], [9].

The remainder of this manuscript is organized as follows. Section II presents the motivating factors for dynamic frequency control including a description of how frequency control can address optimizing Time Warp synchronized parallel simulations. Section III provides a review of the Intel SCC processor and its voltage and frequency adjustment capabilities is presented. In addition, some background information about Dynamic Voltage and Frequency Scaling (DVFS) in x86 processors with a quick review of the corresponding specifications in the ACPI standard to control DVFS. Section IV briefly describes the framework and plans we follow to use multi-core processors to demonstrate dynamic frequency control to optimize Time Warp simulations. Section V describes the experimental platforms and software codes and reviews the results of the experimental analysis. Finally, Section VI contains some concluding remarks and suggestions for future research.

II. Motivation

A. Emerging Many-Core Chips

As vendors grow their multi-core chips to many-core, the systems will increasingly take on a structure more akin to distributed memory, message passing Beowulf clusters and away from the global shared memory processing model. Migrating parallel simulation kernels from multi-core to many-core processors will require refactoring to alleviate possible contention to shared resources and move the communication events from shared memory to message passing through the on-chip communication network. While the onchip communication network will probably have much higher performance capability than conventional Beowulf clusters, load balancing and process migration on many-core processors will remain costly and computationally expensive; thread partitioning and core assignment will become significant issues.

While many-core chips present a more distributed memory processing model, they also present new features not generally available or exploited in contemporary Beowulf clusters. For example, the Intel SCC platform exposes control of the chip's voltage and frequency settings to the programmer [10], [11]. Much like contemporary multi-core chips, these settings are semi-independent across the cores. While contemporary systems primarily use frequency control for lower power, the SCC platform presents a highly flexible infrastructure for voltage and frequency control that can both underclock and overclock the processing cores. For example, on the Intel SCC processor, core frequencies can be adjusted from 100MHz to 1.3GHz (this control is explained more fully in Section III). This flexibility presents an opportunity for the system to dynamically adjust core frequency to fine tune core frequency to accelerate threads on the critical path and decelerate threads off the critical path. 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 build software control structures that properly identify threads for acceleration and deceleration.

B. Time Warp

Time warp synchronized parallel simulators process events optimistically and without strict adherence to the causal orders of the events throughout the entire simulation [8], [9]. While this may allow the parallel simulation to run faster than the critical path of execution [12], it can also result in premature computations that trigger rollbacks and event reprocessing. On a shared memory platform, the event queue can be managed so that the parallel threads process events in a least time-stamp first policy and that the parallel simulation more or less end up following the critical path through the global event chain. However, when migrating to a many-core solution, the global event list becomes a point of contention and alternate organizations with multiple event lists for distinct subsets of the LPs will become necessary. In this case, the worker threads may be processing events well off the critical path until, and if, some load balancing mechanism redistributes the work in a more even manner. While load balancing LPs among the worker threads can help the threads track the critical path, the possibility of binding threads to cores and independently adjusting the operating frequency (up or down) of each core to maximize its operating efficiency can also fine tune the system performance to accelerate the critical path of the simulation.

Run-time tuning to optimize performance has been successfully applied to a number of subalgorithms of a Time Warp synchronized parallel simulation [13]. Most significantly it has been used for: (i) sizing the checkpoint interval of an LP [14], [15], [16], (ii) selecting the cancellation strategy for an LP [17], and (iii) for event scheduling [18], [19], [20], [21]. Each of these mechanisms develop and use some run-time measurements to assess performance and guide the tuning algorithms (*e.g.,* rollback frequency vs rollback costs, effectiveness of premature computations to produce useful work, and so on). The work most closely related to this work is LP scheduling. Most of this previous work focuses strictly on scheduling strategies to delay scheduling the execution of events to reduce time warp overheads. However, the work by Tay, Teo and Kong is especially interesting in that a method for accelerating the critical path is proposed [21]. Acceleration of the critical path is obtained by dynamically adjusting the number of events executed in each LP cycle. Although the experimental results presented in this work are promising, the acceleration of LPs on the critical path is constrained by the speed of the machine on which the simulation is executing. We hope to show that further acceleration is possible using DVFS.

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.

This manuscript presents some preliminary results for dynamically adjusting core frequencies to optimize time warp synchronized parallel simulation. Lacking access to an SCC chip, this demonstration is achieved using an existing multi-core platform with frequency control that is far more granular and limiting than that provided by the Intel SCC platform. Complicating this study is the fact that not all contemporary multi-core platforms support full and independent control of each core's frequency setting. In fact, all of the Intel chips and most of the AMD chips to not strictly follow the software settings. Only the AMD Phenom II X6 platform supports the full range of independence in core frequency that is required to complete this experiments.

III. Background

A. 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 [4], [5], [6]. 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. 1). Each of the 24 tiles contains a dual-core processor, L1 and L2 caches, and a router for 2-D message passing over the mesh network connecting the tiles. There are four memory controllers on the board, supporting a total of 64GB of addressable DDR3 memory. One very interesting feature of the SCC platform is a fine grained, software controllable, dynamic power and frequency management capability.

The Intel SCC platform offers an interesting set of features. Of relevance to this study are the voltage and frequency throttling features [10], [11], [22]. The chip is divided into 7 voltage domains (called *voltage islands*); 6 for each 2x2 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 controllers, and one for the voltage regulator controller.

As shown in Fig. 1, each dual core tile is an independently controllable frequency island. The range of operating frequencies of each tile scales from 300 MHz at 700 mV to 1.3 GHz at 1.3 V [10]. The chip is designed to target a normal operating frequency of 1 GHz. Voltage changes have relatively high latencies (on the order of milliseconds) whereas frequency changes are much faster (on the order of 20 cycles). Of course the actual operating frequencies are a function of both voltage and frequency and therefore the full range of operating frequencies are not always achievable with low latency. Furthermore, since each voltage island covers 4 tiles, the range of selectable frequencies in the frequency islands within a voltage island is limited.

B. DVFS in Multi-Core x86 Chips

DVFS is a computer architecture techniques where the processor voltage and/or frequency can by adjusted to better compensate for the processing needs of the system. Typically these techniques are used for reducing the power consumption of the system. Both voltage and frequency adjustment can reduce the dynamic power, or *switching power*, consumed by a CMOS gate.

The Advanced Configuration and Power Interface (or ACPI) specification [23] provides industry-standard interfaces for Operating System directed configuration and Power Management of devices. 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. ACPI 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 DFS technologies above control the P-states. The key advantage of P-states is that switching between states is low latency. The ACPI standard also defines *Throttling States*, or T-states that control processor frequency throttling. However, because these states do not generally reduce power consumption, they 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 [24] provides access to devices, device drivers, and device configurations through the `sysfs` virtual file system. In `sysfs` there is a subsystem called CPUfreq [25], [26], [27] 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

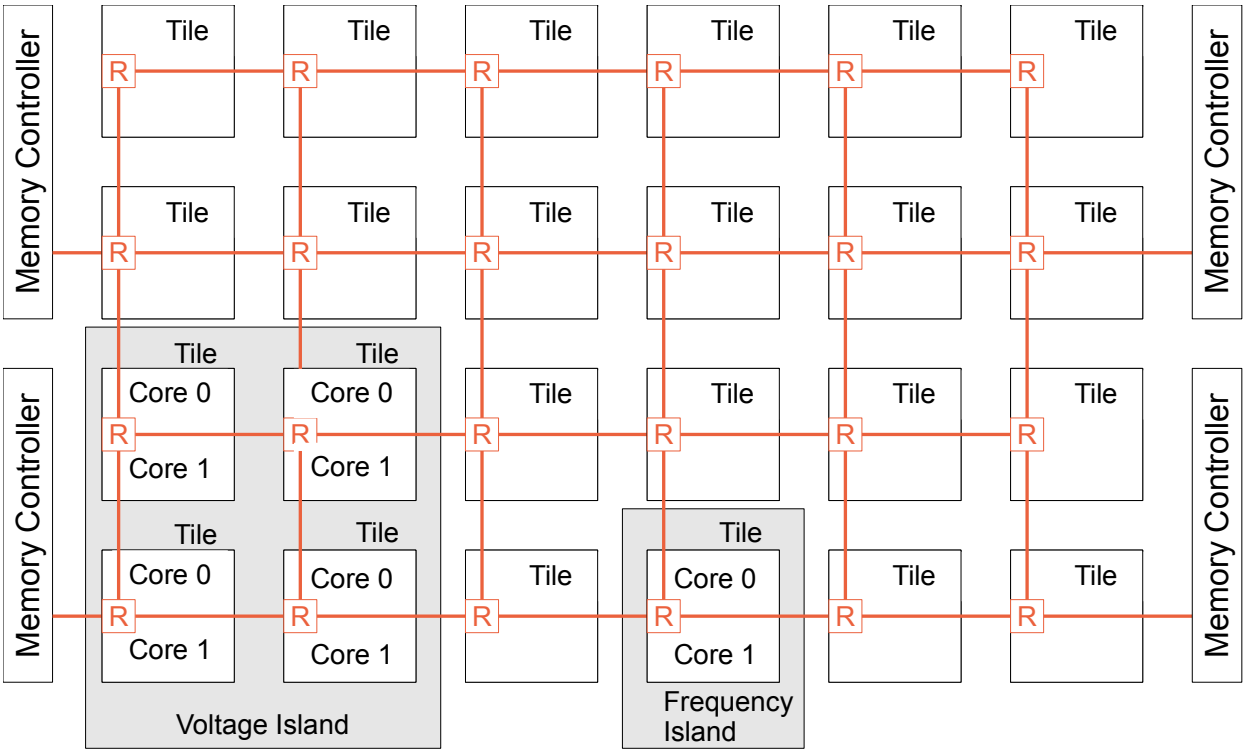

Figure 1. The Intel SCC chip architecture

to us is the `Userspace` governor which gives the user the ability to manually select the processor frequency.

IV. DVFS and Time Warp

In this study, we investigate DVFS using a multi-core platform. While not a perfect match to the Intel SCC many-core platform, we believe that the hardware and software systems can be configured to permit us to draw reasonable inferences from the study. In particular, the system is setup as follows:

1) The simulation LPs are grouped together into N heavyweight threads that are each bound to a specific core on the many core platform. Each thread has a global event queue for the LPs assigned to it and a least timestamp first scheduling policy is followed. In the remainder of this paper, we will use the term LP to denote these collections of LPs on each core. Furthermore, we will refer to "over(under)clocking an LP" as shorthand for "over(under)clocking the core to which an LP is bound."
2) Event information is communicated, using MPI calls, between the threads and event information within a thread is inserted directly into the event queue.

Given the above characterized configuration, we propose that it is possible to achieve shorter simulation run-times by over-clocking those LPs on the critical path and underclocking those off the critical path. However in order for this to occur, the following basic conditions must be met:

1) It must be possible to implement software control of the clock frequency of each core (and furthermore, the hardware must adhere to the software settings),
2) The processor must allow frequencies to be set (by software) above the standard operating maximum (the processor cores may be overclocked),
3) The steps between the available clock frequencies must be small enough to accelerate LPs on the critical path without immediately driving them off the critical path, and
4) The critical path must be sufficiently biased; that is, on average throughout the simulation the critical path will lie more on some LPs than others.

The first condition is already met by certain multi-core processors currently on the market; however, the second is not (except for the Intel SCC platform). To satisfy the second condition, we will conduct experiments with a test platform that is initially configured below its normal operating frequency and use the next higher and lower frequencies as, respectively, overclock and underclock states. The third condition is out of our control and it does cause some complications for this study. In particular, the Intel SCC has 15 frequency settings at each of the 8 voltage settings [11]. An example of these settings and their percent differences

are shown in Table I (the default system operating frequency setting is 4). By contrast, the AMD Phenom II X6 platform frequency settings are shown in Fig. II. The X6 clock frequencies are more coarsely grained and can (and do) present some complications for our experiments. Fortunately this condition is related to the fourth condition and we will exploit that relationship to mitigate the complication of coarse frequency control in the experimental platform. In particular, we will modify the LP partitioning to more heavily skew the load among the executing threads. By starting with a simulation model with a large amount of parallelism and then skewing the partitioning of objects to LPs such that the critical path is sufficiently biased, we can fabricate an environment in which even large frequency steps can accelerate the critical path without introducing instability. Speedups obtained in this environment should then in theory be realizable with smaller critical path bias given sufficiently small frequency steps.

Since run-time tuning of Time Warp has been successfully used to dynamically discover optimal settings for simulation parameters such as the checkpointing interval, selection of cancellation strategy, and event scheduling parameters, it is natural to apply run-time tuning of the core clock frequencies. Furthermore, as mentioned in previous sections, rollback behavior can be used as an indirect measure for estimating the critical path of execution in a Time Warp simulation. A rollback counter can be easily maintained by each LP and made available for the purposes of critical path estimation. The current state of the rollback counter can then be passed to the control mechanism at any given time with very little overhead.

Table I
EXAMPLE FREQUENCY SETTINGS IN THE INTEL SCC

Setting	Frequency (MHz)	% change
2	800	
3	533	33.33%
4	400	25.00%
5	320	20.00%
6	267	16.67%
7	229	14.29%
8	200	12.50%
9	178	11.11%
10	160	10.00%
11	145	9.09%
12	133	8.33%
13	123	7.69%
14	114	7.14%
15	107	6.67%
16	100	6.25%

Table II
EXAMPLE FREQUENCY SETTINGS IN THE AMD PHENOM II X6

Setting	Frequency (MHz)	% change
P0	2,800	
P1	2,200	21.43%
P2	1,500	31.82%
P3	800	46.67%

The frequency control model used in this study is a centralized model that (i) measures the number of rollbacks in each LP and (ii) determines and assigns clock frequencies to each core. Both of these tasks are performed every measurement cycle, which is defined as M iterations through the simulation loop. The model is centralized in that only a single "master" LP performs these tasks. The other LPs are responsible only for providing their rollback numbers. The master LP measures rollbacks by initiating a round of messages between all LPs. It begins by sending a "rollback vector" message to the next LP. The next LP adds the value of its rollback counter to the rollback vector message and passes it on to the next LP. This process is repeated until all of the LPs have added their rollback counters to the rollback vector and the message is sent back to the master LP. At the end of the process, the master LP has a vector with the number of rollbacks of each LP for the current measurement cycle. Using this data, we develop algorithms (described next) to calculate and assign a new distribution of clock frequencies for the LPs.

To map rollback behaviors onto core frequencies, we have experimentally developed and tested several algorithms. The two most successful of these are shown in Figs. 2 and 3. The first algorithm guarantees an even distribution of underclocked and overclocked frequencies. In practice, this should result in a fairly safe thermal profile across the processor, so we call this the "safe algorithm." The safe algorithm falls short in that it does not consider the relation of each individual LP to the overall average. For example, if during a given measurement cycle LP_0, LP_1, and LP_2 roll back 10 times and LP_3 rolls back 100 times, one of LP_0, LP_1 or LP_2 will actually be underclocked instead of overclocked.

To obtain a more useful distribution of clock frequencies, the safe algorithm was modified to adjust the clock frequency of LPs based on each LPs distance to the average rollbacks for each measurement cycle. Fig. 3 illustrates the greedy algorithm. In this algorithm, each LP that significantly deviates from the average rollbacks will be adjusted in the correct direction, without regard to the clock frequencies of the other LPs, hence the name "greedy." Unlike the safe algorithm, there is no guarantee that cores will be equally over/underclocked, which may lead to dangerous thermal profiles. Before being used in an environment with actual overclocking, it is likely that another layer would be needed in hardware or software that could map the desired clock frequencies determined by this algorithm onto a more thermally safe distribution.

```
1. Collect rollback counts of each LP.
2. Sort LPs in order of increasing
   rollbacks.
3. Set the frequencies of the cores
   containing the first n/2 LPs to
   the overclocked frequency.
4. Set the frequencies of the cores
   containing the last n/2 LPs to
   the underclocked frequency.
```

Figure 2. Safe algorithm

```
1. Collect rollback counts of each LP.
2. Set each core frequency using
   equation (1).
```

Figure 3. Greedy algorithm

For the greedy algorithm, we use the following equation to assign clock frequencies to the LPs:

$$f_i = \begin{cases} f_{overclock}, & r_i < r_{avg} - \frac{h}{2} \\ f_{underclock}, & r_i > r_{avg} + \frac{h}{2} \\ f_i, & r_{avg} - \frac{h}{2} \leq r_i \leq r_{avg} + \frac{h}{2} \end{cases} \quad (1)$$

where the variable f is clock frequency, r is the number of rollbacks for the current measurement cycle (r_i denotes a single LP and r_{avg} denotes the average of all the LPs), and h is the size of the hysteresis zone that dampens the response rate of the control algorithm to help prevent instability of the frequency settings. This is discussed more fully in the next section.

Stability

Instability can occur wherever there is a feedback loop and it is a concern in the design of any closed-loop systems feedback control system [28]. With the introduction of a DVFS control mechanism in a Time Warp simulation, a feedback path exists in every LP. For example, let LP_0 and LP_1 be two LPs in a simulation. At some point during the simulation, LP_0 may go off the critical path and roll back many times in a short period of time. The DVFS control mechanism will detect this and underclock LP_0. If the underclocked frequency is too low, it may overcompensate for the increase in rollbacks and actually cause the rollbacks to drop to virtually zero. This will be detected by the DVFS control mechanism which will set the clock frequency to overclocked, which will again result in increased rollbacks.

Feedback loops with paths through multiple LPs may also exist. Tay, Teo and Kong showed that unrestrained acceleration of the critical path can lead to a racing effect in which recursive rollback occurs due to LPs constantly surpassing each other [21]. To mitigate this racing effect, Tay *et al* defined a hysteresis zone in which LPs are not accelerated, even if they are behind relative to the global progress of the simulation. Defining a hysteresis zone based on LP rollbacks for a measurement cycle between the minimum and maximum should be able to reduce instability by not overclocking or underclocking LPs whose rollbacks for the current measurement cycle are close to the average.

The stability or instability of a Time Warp simulator equipped with a DVFS control mechanism is a function of many parameters. Some of these parameters are currently controllable and some are not. The frequency steps, for example, on the AMD Phenom II X6 are limited to only four frequencies. The range of frequency steps available for a set of cores together with the load distribution may cause a simulation under DVFS to become unstable. As will be shown in the following section, a large frequency step combined with a mostly even load distribution results in extreme oscillation. With a more unbalanced load, however, a larger frequency step adequately balances the simulation without becoming unstable. Given a choice of frequency steps, a DVFS control mechanism might be able to analyze the load distribution across all LPs and choose an appropriate frequency step. The length of the measurement cycle is also related to stability: If oscillation occurs, a larger measurement cycle length will result in oscillations of greater magnitude, while a smaller measurement cycle will result in smaller oscillations and more overhead in computation and communication.

V. Experimental Results

Experiments were performed on a PC workstation with an AMD Phenom II X6 and 4GB RAM, running Linux kernel 3.0.0. The WARPED [29] parallel simulation kernel, was extended to include two different DVFS control mechanisms and additional parameters for selecting and configuring the mechanism. Fujimoto's parallel hold, or PHold [30] model was used as the simulation model. In all experiments, four LPs were used, each of which were bound to one of the cores of the X6.

Rollback measurements were filtered with a Finite Impulse Response (FIR) filter for smoothing before sending to the control mechanism [28]. Experimental data suggested that varying the FIR filter size did not have a significant effect on the overall simulation time. The measurement cycle for all experiments was chosen to be 1000, based on the length of the simulations. Finally, a hysteresis zone of 10% of the distance between minimum and maximum rollbacks, centered on the average, was used.

Our initial attempts at dynamic frequency control resulted in extreme swings in rollbacks and wide oscillations as discussed in the previous section. This was due to the X6 having frequency steps much too large ($\pm$ $\sim$600MHz) for a load that was already fairly balanced. The extreme rollback swings can be seen in the trace of a simulation run, as in Fig. 4. Measurement cycles are shown along the x-axis and the y-axis shows the number of rollbacks in the current

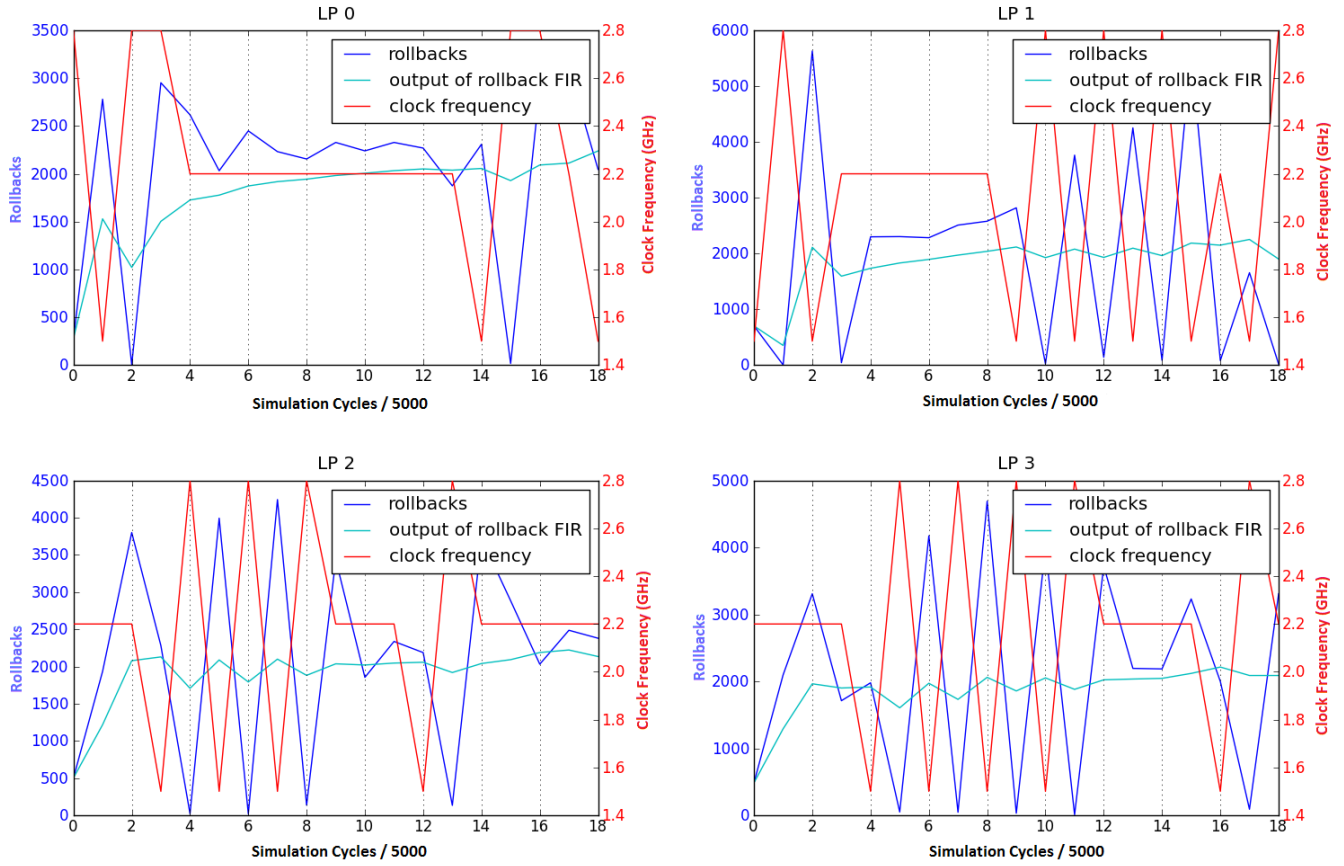

Figure 4. Rollback Histories.

Table III
RESULTS WITH A PHOLD SIMULATION MODEL

N	Runtime (s)					Rollbacks				
	With DVFS		Without DVFS		Speedup	With DVFS		Without DVFS		Decrease
	Mean	Stdev	Mean	Stdev		Mean	Stdev	Mean	Stdev	
∞	19.848	0.165	18.293	0.065	-7.83%	1797.6	129.382	968.1	71.346	-46.14%
16	19.634	0.092	17.685	0.077	-9.93%	1638.6	63.861	1109.1	49.979	-32.31%
8	18.497	0.146	18.501	0.082	0.02%	1451.3	126.314	1760.4	94.394	21.30%
4	19.748	0.061	20.453	0.045	3.57%	2716.1	236.710	3833.8	84.315	41.15%
2	23.547	0.071	24.828	0.080	5.44%	7510.9	96.225	9036.1	122.477	20.31%

measurement cycle in dark blue, the output of the FIR in light blue, and the assigned clock frequencies in red. The measurement cycle for this particular trace was 5000. It can clearly be seen from the figure that over(under)clocking causes an immediate and extreme decrease(increase) in rollbacks, which triggers wide oscillation. Obviously, the frequency steps available on our experimental platform are simply too large.

As mentioned in the previous section, we cannot control the clock frequency steps, but we can control the critical path bias. WARPED creates a `SimulationObject` for each physical process being modeled and partitions them among each LP. To artificially bias the critical path towards a particular LP, we experimented with skewing the static partitioning of `SimulationObjects` to LPs. Every N rounds, two simulation objects were assigned to LP_0 and LP_1 was skipped. We refer to partitionings with small N as "heavily skewed" and partitionings with large N as "lightly skewed." $N = \infty$ indicates no skew. Table III shows simulation run-times and rollback numbers for the PHold model with various degrees of skew. For each amount of skew tested, data was taken from 10 different simulations,

and the FIR filter size was 128. The PHold model parameters were 200 processes, a message density of 20, a state size of 1024, and a grain of 100 [30].

Experimental data shows that speedup generally increased with the skew of the load, as expected. The number of rollbacks also generally decreased as the simulation times increased, which suggests that the number of rollbacks is indeed related to simulation efficiency. Because speedup is possible with large frequency steps for heavily skewed loads, we infer that speedup of more normally loaded simulations might be attainable given smaller frequency steps. The 5% speedup observed in the most imbalanced case was not as dramatic as the authors had expected. In that regard, we faced many unexpected challenges during this study. First, the number of rollbacks sometimes varied significantly between measurement cycles. This may possibly be offset by running extremely lengthy simulations, decreasing the measurement cycle, and using a larger FIR filter size. Difficulty in avoiding oscillation even for relatively heavily skewed loads also leads us to suspect that controlling the clock frequency of one LP in the simulation has a greater effect on the rollback behavior of the other LPs than we originally believed. The benefits of balancing the rollbacks across all LPs is also easily overstated. We assume an LP with few rollbacks is spending more time on the critical path than the others. Accelerating it quickly leads to an increase in rollbacks which pushes it more off the critical path. The greatest gains in speedup resulted from simulations in which overclocking the LP with the least rollbacks had the least effect on the rollback behavior of that LP. In other words, DVFS control is most effective when it can accelerate the LP on the critical path without moving the critical path.

These findings present many opportunities for future work. We have presented data for a fixed set of simulation parameters, however the effects of individual parameters (specifically the DVFS control parameters we have proposed in this study, namely the measurement cycle, FIR filter size, and hysteresis zone) on simulation run-time have yet to be examined in depth. Certain configurations of the PHold model, or any other simulation model, may lend additional insight on the viability of DVFS control in Time Warp simulations. The parameters we chose for our experiments are highly dependent on the simulation length and amount of parallelism in the simulation model. Adaptive techniques may be used to dynamically select optimal DVFS control parameters for any simulation model. Finally, these experiments could be repeated on a platform with smaller frequency steps such as the Intel SCC to verify that faster simulation times are possible for more balanced loads.

VI. Conclusion

We have given an overview of DVFS features available in modern multi-core and many-core processors. We have developed an algorithm to measure the rollback behavior of all LPs and set core frequencies such that the critical path is accelerated and other paths are decelerated. While demonstrated on an AMD Phenom II X6 at less-than-maximum clock frequencies, the proposed DVFS control method would be most promising given a platform that allows overclocking. Such a platform would not necessarily suffer from overheating effects as our DVFS control algorithm guarantees that some cores are underclocked whenever others are overclocked. Our experimental data shows that the frequency steps available on the X6 are too coarse to be applied directly to existing Time Warp simulators without skewing the partitioning of simulation objects. However, data also shows that speedup is possible using coarse-grain frequency steps for more imbalanced loads. This suggests that speedup may be possible with a more normal partitioning if only fine-grain frequency steps such as those on the Intel SCC were available.

Acknowledgment

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

References

[1] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009.

[2] A. Ghuloum, "Face the inevitable, embrace parallelism," *Communications of the ACM*, vol. 52, no. 9, pp. 36–38, Sep. 2009.

[3] R. Kalla, "Power7: Ibm's next generation power microprocessor," in *Hot Chips 21*, Aug. 2009.

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

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

[6] I. Labs, "The SCC platform," Intel Corporation, Tech. Rep., May 2010. [Online]. Available: http://techresearch.intel.com/spaw2/uploads/files/SCC_Platform_Overview.pdf

[7] ——, "Intel insights at SC11," Intel Corporation, Tech. Rep., Nov. 2011. [Online]. Available: http://newsroom.intel.com/community/intel_newsroom/blog/2011/11/15/intel-reveals-details-of-next-generation-high-performance-computing-platforms

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

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

[10] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, Jan. 2011.

[11] I. Labs, "The SCC programmer's guide," Intel Corporation, Tech. Rep., May 2010.

[12] D. Jefferson and P. L. Reiher, "Supercritical speedup," in *Proceedings of the 24^{th} Annual Simulation Symposium*, A. H. Rutan, Ed. IEEE Computer Society Press, Apr. 1991, pp. 159–168.

[13] A. Palaniswamy and P. A. Wilsey, "Parameterized Time Warp: An integrated adaptive solution to optimistic pdes," *Journal of Parallel and Distributed Computing*, vol. 37, no. 2, pp. 134–145, Sep. 1996.

[14] L. Auriche, F. Quaglia, and B. Ciciani, "Run-time selection of the checkpoint interval in time warp based simulations," *Simulation Practice and Theory*, vol. 6, no. 5, pp. 461–478, 1998.

[15] J. Fleischmann and 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)*, Jun. 1995, pp. 50–58.

[16] R. Rönngren and R. Ayani, "Adaptive checkpointing in Time Warp," in *Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94)*. Society for Computer Simulation, Jul. 1994, pp. 110–117.

[17] R. Rajan, R. Radhakrishnan, and P. A. Wilsey, "Dynamic cancellation: Selecting Time Warp cancellation strategies at runtime," *VLSI Design*, vol. 9, no. 3, pp. 237–251, 1999.

[18] A. Palaniswamy and P. A. Wilsey, "Scheduling Time Warp processes using adaptive control techniques," in *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Eds., Dec. 1994, pp. 731–738.

[19] F. Quaglia and 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)*, May 2000.

[20] T. Som and R. Sargent, "A probabilistic event scheduling policy for optimistic parallel discrete event simulation," in *Proc. of 12th Workshop on Parallel and Distributed Simulation (PADS98)*, May 1998, pp. 56–63.

[21] S. Tay, Y. Teo, and S. Kong, "Speculative parallel simulation with an adaptive throttle scheme," in *Proc. of 11th Workshop on Parallel and Distributed Simulation (PADS97)*, Jun. 1997, pp. 116–123.

[22] I. Labs, "How to read the voltage and frequency on SCC," Intel Corporation, Tech. Rep., Jun. 2010.

[23] *Advanced Configuration and Power Interface Specification*, 4th ed., Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation, 2010. [Online]. Available: http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf

[24] L. K. O. Inc, "The linux kernel archives," Linux Kernel Organization Inc, Tech. Rep., 2011. [Online]. Available: http://www.kernel.org

[25] D. Brodowski and N. Golde, "Linux CPUFreq – CPUFreq governors," Linux Kernel, Tech. Rep. [Online]. Available: http://www.mjmwired.net/kernel/Documentation/cpu-freq/governors.txt

[26] J. Hopper, "Reduce Linux power consumption, part 1: The CPUfreq Subsystem," IBM, Tech. Rep. [Online]. Available: http://www.ibm.com/developerworks/linux/library/l-cpufreq-1/index.html

[27] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor: Past, present, and future," in *Proceedings of the Linux Symposium*, 2006, pp. 223–238. [Online]. Available: http://www.linuxinsight.com/ols2006_the_ondemand_governor.html

[28] R. C. Dorf, *Modern Control Systems*, 6th ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1991.

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

[30] R. M. Fujimoto, "Performance of time warp under synthetic workloads," in *Proceedings of the SCS Multiconference on Distributed Simulation, vol. 22*. Society for Computer Simulation, Jan. 1990, pp. 23–28.