

CoGS-Sim - CoPhase-Guided Small-Sample Simulation of Multithreaded and Multicore Architectures

Joshua L. Kihm and Daniel A. Connors
University of Colorado at Boulder
Department of Electrical and Computer Engineering
UCB 425, Boulder, CO, 80309
{kih, dconnors}@colorado.edu

Abstract

The emergence and ubiquity of multithreaded and multicore architectures in modern processors presents many new challenges in processor design. These architectures present significantly larger design and benchmark spaces than traditional single-context architectures. Such characteristics put ever-higher strain on efficient cycle-accurate simulation which remains a vital tool in the design process.

In many cases in multicontext architectures, independent processes are running simultaneously. By tracking the program phase of each of the running processes and using that information to guide small-sample simulation, it is possible to perform very fast and accurate sampled simulation of a multicontext architecture. As with any sampled simulation, speed-up is achieved by minimizing the amount of detailed, cycle-accurate simulation. By tracking the variance of performance in each co-phase, it is possible to minimize the amount of simulation in each co-phase and of the programs as a whole, further accelerating the simulation process and providing an automatic adaptive mechanism to adjust the amount of detailed simulation as needed. Finally, by tracking phase behavior, it is possible to quickly adapt to co-phase changes on the fly during simulation without prior knowledge of the running programs. This paper presents this technique, which is called CoGS-Sim for CoPhase-Guided, Small-Sample Simulation.

1. Introduction

Cycle-accurate architectural simulation remains a vital component of the processor design cycle.

Simulation allows the expedient exploration of design alternatives which is crucial in design space exploration and finding the optimal design. However, the number of designs which can be explored, and therefore, to what degree the design space can be covered, is dependent on the speed of simulation. Thus, highly-efficient simulation methods which do not reduce accuracy are indispensable in the design process. Additionally, the sudden proliferation of various types of multithreaded and multicore architectures has caused a prodigious expansion of both the design and the benchmarking space.

A very effective method for dealing with this expansion is small-sample simulation ([1, 18]). Small-sample simulation relies on the observation that in most systems a very short detailed simulation period on the order of just a few thousand instructions is typically representative of a much larger simulation period on the order of a million instructions. Because of their short length, a very large number samples can be taken across the entire test program. By combining this information with phase behavior, it is possible to greatly reduce the total number of samples required, as was shown in *Phase-Guided, Small-Sample Simulation* or *PGSS-Sim* ([4]). Applying small-sample simulation to multicontext architectures presents unique challenges which were explored in [3] and will be explained in Section 3. A multicontext architecture is simply any architecture capable of running multiple threads or processes simultaneously or very near simultaneously, which is to say any multithreaded or multicore architecture. The greatest challenge in simulating independent processes in multithreaded architectures is that the relative position of the processes determines the performance of the system to a very large degree. Simulation must take into account that

any relative position is possible and that any deviation from the path of a full, detailed simulation can greatly change the outcome of the simulation. By combining the techniques of phase-guided simulation and multithreaded, small-sample simulation, it is possible to meet these challenges in a technique called *Co-phase-Guided, Small-sample Simulation* or *CoGS-Sim*.

The remainder of this paper is organized as follows. Section 2 explains the motivation for efficient simulation in multithreaded and multicore architectures. Next, Section 3 explains CoGS-Sim in detail. Finally, initial results are presented in Section 4 and Section 5 concludes and outlines future work.

2. Motivation

Although accurate and efficient architectural simulation has always been a challenge for processor designers, current trends in processor design have greatly increased the difficulty of the problem. Due to a variety of restraints including power, heat, and design complexity, traditional methods of achieving higher performance through higher clock speeds on single-context processors are no longer viable. As such, virtually all modern processors achieve higher process throughput by running multiple processes at the same time either through various types of multithreading or multicore architectures. In fact, many modern processors and future processor designs consist of multicore designs which incorporate some type of multithreading on each core. The tremendous variety of possible configurations of multicontext systems in which resources are shared and how they are shared creates a much larger design space to be explored versus an equivalent single-threaded processor. This expansion of the design space puts even higher demand on efficient simulation to explore this ever expanding design space.

The limitation of multicontext architectures is that the contexts share finite resources and as such interference detracts from the ideal performance of each co-scheduled thread on the system. A large range exists in which resources are shared and how. At one end, certain multicore designs amount to little more than multiple single-context processors in the same package which only share off-chip bandwidth. On the other end, simultaneous multithreading (SMT) designs ([12]) share all processor re-

sources between threads. The effectiveness of such systems is almost primarily dependent on the degree of interference, creating a challenge to simulation as this contention for resources and its resultant performance degradation must be modeled in addition to the individual behaviors of each running processes.

One fortunate characteristic of almost all programs is that they exhibit repetitive behaviors called *phases* which can be exploited for sampled simulation ([8]). When independent processes run concurrently on a multicontext system, their phases combine into what is called a *co-phase* ([14]). Although multicontext systems are ideal for multithreaded programs and multithreaded programming is the only way to achieve higher performance of a single program on these systems, single-threaded programs make up the vast majority of existing code and because of their simplicity and the difficulty of parallelization of many algorithms, they will continue to make up a large part of programs for the foreseeable future. The downside of this lack of synchronization is that no restrictions exist on the relative position between co-scheduled programs. As a result, any co-phase may be encountered in a real system and the overall performance of the system will be widely variant depending on the offset between threads ([5]). As the number of contexts increases, the number of co-phases grows at a sub-factorial rate in a system with symmetric contexts. Additionally, as the benchmarks get larger and more complex, like the *Spec 2006* benchmark suite ([11]), the number of phases increases, causing a further sub-factorial scaling of the number of co-phases. This problem is only exacerbated in designs where the contexts are not identical and program placement becomes important. In such cases, the growth becomes exponential with the number of contexts. This uncontrolled growth in benchmark space puts even further impetus on fast simulation.

3. Methodology

CoGS-Sim combines the ideas of phase-based simulation ([8]) and small-sample simulation ([18]). Additionally, it extends each idea to a multithreaded domain. This section will first explain how small sample simulation can be extended to a multithreaded domain, then how small-sample simula-

tion can be combined with phase-based simulation, and finally how these ideas are combined to create *CoGS-Sim*.

3.1. Small-Sample Simulation on Single and Multi-context Architectures

Small sample simulation was first proposed in [1] and then extended and applied to execution-driven simulation in [18]. The process is relatively simple. A sampling period consists of three parts. First a detailed warming period consisting of approximately five thousand instructions. Detailed warming is identical to full, cycle-accurate simulation, but no statistics are kept. The next phase is detailed simulation on the order of a thousand instructions. The preceding detailed warming eliminates cold start effects, this very short sample typically represents the larger fast-forwarding period very accurately. The final simulation phase is fast-forwarding. Fast-forwarding is functional simulation where no timing information is kept and consists mainly of tracking program state. Additionally, the long-lifetime structures of the cache and the branch prediction models are kept warm, typically using a quick simulation mode. The advantage is that fast-forwarding can be as much as a two or three orders of magnitude faster than detailed simulation, depending on the implementation. These three steps are then simply repeated until the program completes. Because the detailed simulation is so short, it is possible to take a very large number of samples throughout the program execution. This has the coupled advantages of averaging out much of the sampling error and correctly representing each of the programs behaviors without any additional calculation or monitoring.

Although one of the great advantages of small-sample simulation is its simplicity, certain complications arise when it is extended to multicontext architectures. First, the detailed warming and simulation phases must be long enough that each thread is properly measured which is accomplished by simply lengthening the detailed simulation periods such that each thread executes a minimum number of instructions. Extending the SMARTS methodology to transaction based multithreaded programs was explored in [17] where the greatest modifications needed are the extension of the warming and simulation periods. However, in the case of independent programs, the largest modifications are made to the

fast-forwarding period. First, each of the running programs execute at a different speed which is easily taken care of by proportionally fast-forwarding each thread based on the IPC measurements from each thread. Another complication is that the cache and branch predictor affinities, or how much data from each running thread is resident, must be maintained during the fast forwarding period if they are shared between the threads. Since there is no timing information being kept, it is impossible to know exactly how the requests to shared resources would be interleaved in detailed simulation. To model this interleaving, a technique called *MonteCarlo Fast-Forwarding* is used. Essentially, the instructions from each thread are interleaved by randomly picking a thread to execute at each iteration of the fast-forward mechanism. The probability that a given thread will be picked is based on it's IPC from the last detailed sample. Since the fast-forwarding period is on the order of a million instructions, this has the added benefit of handling proportional fast-forwarding as the large sample means that the number of instructions picked from each thread will likely very tracked by its probability. This combination of techniques is discussed in much more detail in [3] and is called *Multithreaded Architecture Small-Sample Simulation* or *MASS-Sim*.

3.2. Phase-Guided Small-Sample Simulation

In [19], it was shown that basic small-sample simulation takes on the order of forty times more detailed samples than are needed to characterize a given program because it does not take into account phase information. In the SimPoint methodology, program phase is determined based on the frequencies of execution of each of the basic blocks which are combined into a *Basic Block Vector (BBV)*. The BBVs are clustered to group together periods of similar execution. However, the k-means clustering which is used in SimPoint limits the granularity at which samples can be taken because it scales poorly. A simple way to avoid this pitfall is to simply use a different clustering algorithm, such as pivot clustering. Pivot clustering works by setting the first point in a cluster as the center of that cluster. Any following point that is within some maximum distance of that center is considered part of that cluster. If a point is outside of the maximum distance from all existing clusters, a new cluster is created

$$n \geq \left(\frac{\sigma t_{a,n-1}}{r \bar{x}} \right)^2$$

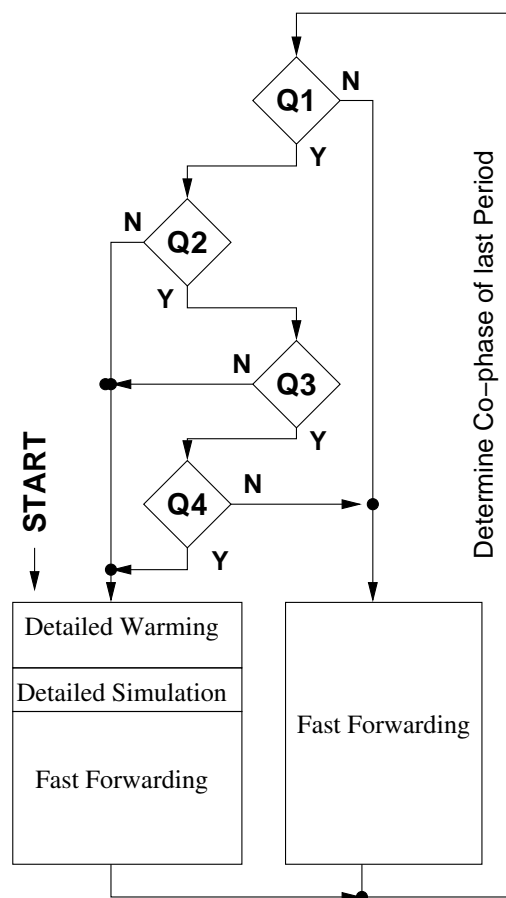
n = the number of samples
 t = The t-distribution value
 a = The level of confidence
 σ = the standard deviation of the data
 r = the desired confidence interval as a fraction of \bar{x}
 \bar{x} = the mean value of the samples

Figure 1. The minimum number of samples required to fall within a confidence interval.

with that point in the center. Contrastingly, if a point falls within the maximum distance of more than one cluster center, it is simply assigned to the closest one. The advantage of using pivot clustering in this case is that it scales linearly with dataset size and since it is a single pass algorithm, can be performed on the fly as the data is collected. By using pivot clustering on the basic block information as it is simulated, phase characterization can be performed on-line, circumventing the road blocks to combining phase-based and small-sample simulation. An added advantage is that there is no off line analysis required to perform the phase characterization. The combination for single-threaded architectures is called *Phase-Guided, Small-Sample Simulation* or *PGSS-Sim* ([4]).

Generating the BBV on-line is a very simple process which was first proposed in [9]. Although that proposal was for a small hardware module, it is very easy to implement in software. Essentially, the address of any taken branch is fed into a hash to reduce it to just a few bits. The hash is randomly generated at the start of execution and is implemented as a bit mask. The number of bits that the address is reduced to is the base-2 logarithm of the desired length of the BBV. This turns each branch address into an index into a BBV. The number of completed instructions since the last taken branch is added to that entry of the BBV. At some specified interval, the BBV is normalized and fed into the clustering algorithm to determine the phase of that interval. The BBV is cleared and the next interval is started.

Determining the number of samples to take in a given phase relies on basic sampling theory. The minimum number of samples required to achieve a



- Q1 – Is the total number of ops since the last sample above the threshold?
- Q2 – Is the number of Samples taken in this co-phase above the minimum?
- Q3 – Does each thread meet the sample Variance Requirement?
- Q4 – Is the number of occurrences of this co-phase since it was last sampled above the maximum?

Figure 2. The process flow of CoGS-Sim. Sampling decisions are made based upon the amount of time since the last detailed sample of a co-phase and the variance in co-phase performance.

given confidence interval can be easily calculated from the standard deviation of the samples, their average, the t-distribution value for the given number of samples and the desired confidence level, and the

confidence interval given as a percentage of the average. This relation is shown in Figure 1. The limitation of this theory is that the average and standard deviation need to be stable. This can be achieved by setting a minimum number of samples to take before testing the inequality. In the simulation realm it is important that the samples are not necessarily the first occurrences of a phase because the first occurrences often suffer cold start performance penalties which are not representative of the phase as a whole. Finally, a threshold is set on the maximum number of occurrences of a phase which can occur between detailed simulation samples. This further spreads the samples across occurrences of very common threads and sets a minimum on the portion of a given phase which must be simulated. A flow diagram of the sampling process is shown in Figure 2.

Extending PGSS-Sim to multicontext environments is simply a matter of combining it with the techniques of MASS-Sim. The only change to the PGSS-Sim technique is that each thread is checked for performance variation individually in determining the number of samples required in each co-phase. CoGS-Sim is the combination of these techniques. CoGS-Sim has two major advantages over simple small-sample simulation. First, the amount of detailed simulation is greatly reduced. Secondly, CoGS-Sim can react more quickly to phase changes. MASS-Sim takes a sample at each and every interval meaning that the intervals must be of a larger length so that the number of detailed samples does not overwhelm the time savings from sampling. Conversely, CoGS-Sim is selective and only performs detailed subset of the simulation intervals, it can use shorter intervals than MASS-Sim. This effect is illustrated in Figure 3. In the graph, each axis represents the number of instructions completed in each phase. The dotted lines illustrate the phase change points in each thread. Each box created by these lines is a co-phase. As execution moves between co-phases, the relative speeds of the threads, and therefore the direction of the line changes. As MASS-Sim overshoots phase boundaries, it diverges from the path which would be followed in full detailed simulation. Conversely, CoGS-Sim quickly reacts to phase changes and more closely follows the correct path of execution.

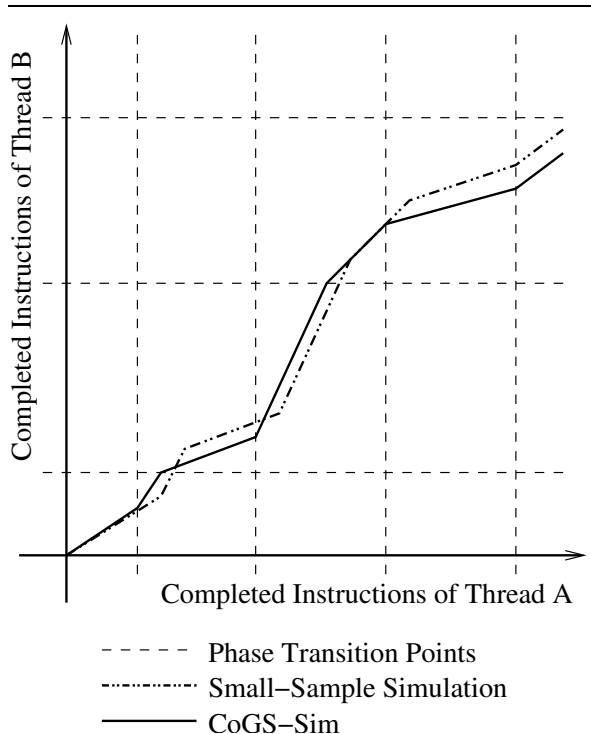


Figure 3. An illustration of the advantage of CoGS-Sim versus simple small-sample simulation. By tracking phase behavior on-line, CoGS-Sim is able to quickly react to changes in IPC as either thread changes phase. Simple small-sample simulation only reacts to changes when a scheduled simulation period occurs, which can lead to errors.

4. Preliminary Results

4.1. Experimental Set-Up

CoGS-Sim has been implemented in our simulator Xsim ([3, 6, 2, 7]). Xsim models any number a simple, in-order cores which can have Simultaneous Multithreading (SMT) ([12], Course-Grained Multithreading (CGMT) ([16]) or both. Xsim also has very flexible sampling, warming, and fast-forwarding for testing sampling techniques. The experiments shown in this section used a simple, single core with two-way SMT. The cache system consists of a 64kB instruction and 16kB data cache with a unified 1MB unified level two cache. All fifty-five combinations of ten of the *Spec2000* ([10]) bench-

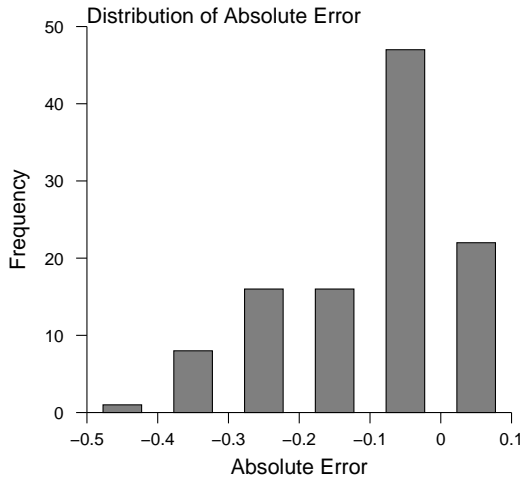


Figure 4. Distribution of total IPC sampling error of CoGS-Sim for various all combinations of ten Spec2000 benchmarks.

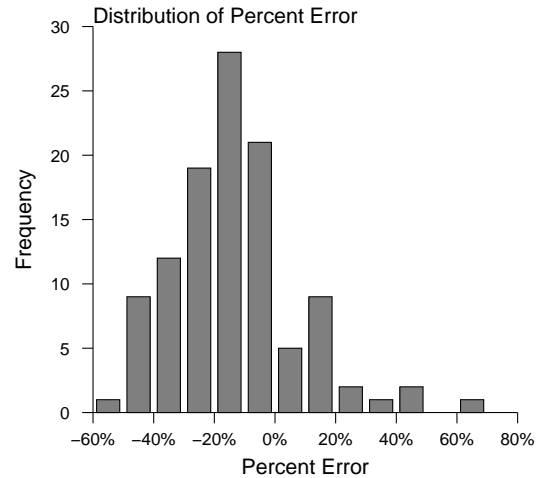


Figure 5. Distribution of total percent IPC sampling error of CoGS-Sim for various all combinations of ten Spec2000 benchmarks.

marks were tested with test inputs. The CoGS-Sim parameters were a minimum of 30 samples per co-phase, a minimum one million instructions between samples in a co-phase, a $\frac{t}{r}$ value of 100, a maximum of 100 occurrences between samples in a co-phase. The sampling parameters were a sample size of one thousand total ops across the threads, five thousand total ops of detailed warming, and one million total ops per fast forwarding period. The pivot clustering threshold was set at $\frac{\pi}{20}$ radians in line with the suggestions of [4].

4.2. Results

Shown in Figures 4 and 5 is the distribution of error and percent error in IPC for the fifty-five benchmark combinations tested. Since there are two program running for each test, there are 110 total points. The error is simply the difference between the total IPC measured by CoGS-Sim and that measured by full detailed simulation. One obvious trend to the data is that the sampling error tends to be negative, that is, the CoGS-Sim is underestimated the data far more often than is it over-estimating it. This is likely due to using too short of a warming period. Since features are not warm, it takes longer for operations to begin to retire as they need to propa-

gate through the pipeline. Further experimentation will be used to test this hypothesis. If this negative trend to the data can be rectified, almost all of the data points will be brought into an acceptable range of sampling error.

Another important metric on multithreaded systems is the ratio of performance between the threads in the system. In addition to simple throughput, a multicontext system must insure some level of fairness between the threads in almost all situations. To measure the ratio error is simply the difference in proportion of one of the program's IPC to the total IPC of the system. For example, if the sampled IPC of thread zero is 40% of the total sampled IPC and in the full simulation it made up 50%, the error would be 10%. In a two context system, obviously either thread can be used to make this measurement. The distribution of this error among the fifty-five tests run is shown in Figure 6. Even with light warm-up used, the ratio was within 4% error in thirty five of the cases and within 10% in forty-seven. Even though most of the IPC measurements were lower than their targets, each thread was off proportionally, and CoGS-Sim still gave a very accurate measurement of IPC ratio and is tracking co-phase behaviors accurately.

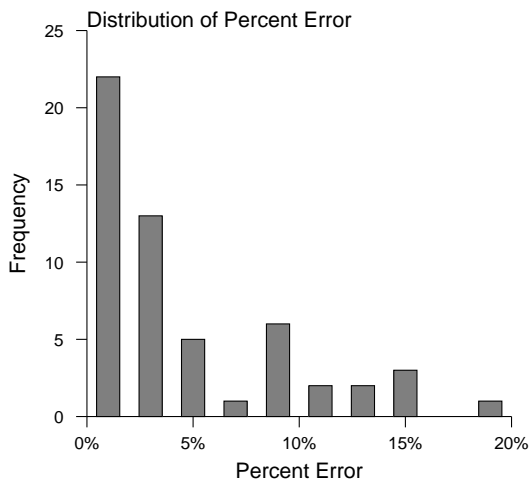


Figure 6. Distribution of error in IPC ratio between the two contexts of CoGS-Sim for all combinations of 10 Spec2000 benchmarks.

A final way to look at the data is through a relative progress graph as was illustrated in Figure 3. This gives an idea of how well the sampling is tracking the actual execution of the program. Shown in Figure 7 is the combination of *177.mesa* and *188.ammp*. The dark line represents the progress in full detailed simulation, which is what the sampling is trying to match. The lighter-colored line is the progress as measured by CoGS-Sim. Although the full simulation was terminated early for time constraints, it is easy to show that CoGS-Sim does a good job of tracking Co-Phase changes in this case.

5. Conclusions and Future Work

The initial work on CoGS-Sim has opened up many avenues for future research. The obvious first step is a thorough exploration of the parameter space using various thresholds and sampling sizes. Another important area of on-going research involves more accurately tracking the required number of samples in a co-phase. Instead of simply having a minimum number of samples before the variance parameters are checked, it would be better to take samples until the average and standard deviation stabilized before formally testing the confidence interval threshold. Also, a better way to calculating

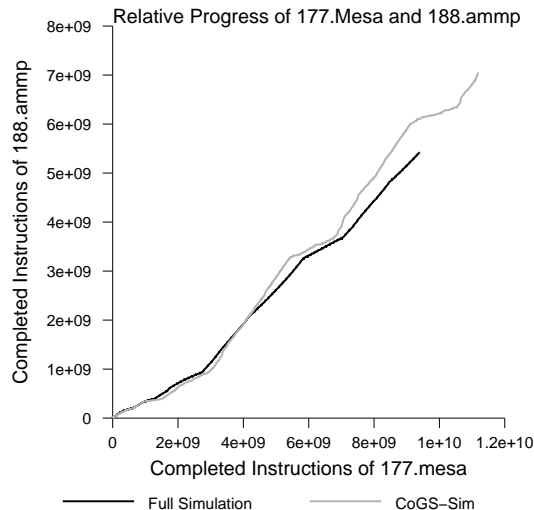


Figure 7. Relative Progress of the Spec2000 benchmarks 177.mesa and 188.ammp when simulated with full detailed simulation and sampled with CoGS-Sim.

the t-distribution value from the confidence interval and the number of samples should be found. CoGS-Sim does not take into account program offset but is equally applicable to any program offset position, so it can be easily combined with methodologies such as FAME ([15]) or by Van Beisbrouck et al. in [13] which both essentially use random program offsets to get a more complete picture of overall performance. Finally, although there are many challenges unique to multithreaded programs, such as synchronization, which must be addressed, CoGS-Sim could be applied to multithreaded programs as well.

Although there is room for improvement and much exploration yet to be done, the initial tests on CoGS-Sim are very promising. By combining the strengths of small-sample and phase-based simulation, it is possible to greatly reduce the requisite amount of detailed simulation while still obtaining accurate results. As multicontext architectures continue to grow in size, complexity, and market share, effective simulation techniques like CoGS-Sim will only become more important.

6. Acknowledgments

The authors would like to thank the anonymous reviewers for their useful comments and Intel Corporation for sponsoring this work. Computer time was provided by NSF ARI Grant #CDA-9601817, NSF MRI Grant #CNS-0420873, NASA AIST grant #NAG2-1646, DOE SciDAC grant #DE-FG02-04ER63870, NSF sponsorship of the National Center for Atmospheric Research, and a grant from the IBM Shared University Research (SUR) program.

References

- [1] T. Conte, M. A. Hirsch, and K. Menezes. Reducing state loss for effective sampling of superscalar processors. In *Proceedings of the 1996 International Conference on Computer Design (ICCD)*. October, 1996, 1996.
- [2] J. Kihm and D. Connors. Implementation of fine-grained cache monitoring for improved smt scheduling. In *Proceedings of The 22nd International Conference on Computer Design*, 2004.
- [3] J. Kihm and D. Connors. Statistical simulation of multithreaded architectures. In *Proceedings of the 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MAS-COTS)*, 2005.
- [4] J. Kihm and D. Connors. Pgss-sim: Phase-guided, small-sample simulation. In *Proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-07)*, Washington, DC, USA, 2007. IEEE Computer Society.
- [5] J. Kihm, T. Moseley, and D. Connors. A mathematical model for balancing co-phase effects in simulated multithreaded systems. In *Proceedings of the 1st Workshop on Modeling, Benchmarking, and Simulation (MoBS)*, 2005.
- [6] J. L. Kihm, A. Settle, A. Janiszewski, and D. A. Connors. Understanding the impact of inter-thread cache interference on ILP in modern SMT processors. *Journal of Instruction Level Parallelism*, 7(2), 2005.
- [7] A. Settle, J. Kihm, , A. Janiszewski, and D. Connors. Architectural support for enhanced smt job scheduling. In *Proceedings of the International Conference on Parallel Architectures and Compiler Techniques*, October 2004.
- [8] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 45–57, 2002.
- [9] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 336–349, New York, NY, USA, 2003. ACM Press.
- [10] Standard Performance Evaluation Corporation. The SPEC CPU 2000 benchmark suite, 2000.
- [11] Standard Performance Evaluation Corporation. The SPEC CPU 2006 benchmark suite, 2006.
- [12] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *The Proceedings of the 30th International Symposium on Computer Architecture (ISCA)*, pages 392–403, 1995.
- [13] M. VanBeisbrouk, L. Eeckhout, and B. Calder. Considering all starting points for simultaneous multithreading simulation. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2006.
- [14] M. VanBeisbrouk, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2004.
- [15] J. Vera, F. J. Cazorla, A. Ramirez, and M. Valero. A novel evaluation methodology to obtain fair measurements in multithreaded architectures. In *Proceedings of the 2nd Workshop on Modeling, Benchmarking, and Simulation (MoBS)*, 2006.
- [16] W. Weber and A. Gupta. Exploring the benefits of multiple hardware contexts in a multiprocessor architecture: Preliminary results. In *Proceedings of the 16th International Symposium on Computer Architecture (ISCA)*, pages 273–280, June 1989.
- [17] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. Simflex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, 2006.
- [18] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *The Proceedings of the 30th International Symposium on Computer Architecture (ISCA 2003)*, 9-11 June 2003, San Diego, California, USA, pages 84–95, 2003.
- [19] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. An evaluation of stratified sampling of microarchitecture simulations. In *The Proceedings of the Third Annual Workshop on Duplicating, Deconstructing, and Debunking (held in conjunction with ISCA-31)*, 2004.