Low Cost Trace-driven Memory Simulation Using SimPoint

Michael Laurenzano Beth Simon Allan Snavely Meghan Gunn
{michaell, bsimon, allans, mgunn}@sdsc.edu
Performance Modeling and Characterization Lab
San Diego Supercomputer Center, University of California, San Diego

Abstract

Trace-driven memory simulation tools such as MetaSim Tracer [1] capture the address stream of an application during an instrumented program run. Various statistics can be measured using the in-flight address stream including anticipated cache hit rates. This paper reports on performance improvements of MetaSim Tracer gained by using techniques developed in SimPoint [2]. Concurrent research [3] addresses techniques that can be used to reduce the instrumentation overhead involved in memory tracing, and this work addresses a technique that can be used to decrease the amount of cache simulation that is required on top of this. The result is a tool for trace driven cache simulation that is practical to use for memory performance studies of full sized scientific applications.

1 Introduction

In [1, 4] a system, called the MetaSim Tracer, is described that is capable of predicting the performance of an HPC scientific application on a target machine by combining information describing the behavior of an application (called the application signature) and the behavior of the given target machine (called the machine profile). While the machine profile is derived by simply benchmarking the target machine's memory hierarchy (a quick and easy process), the gathering of an application signature uses the costly process of binary instrumentation in order to simulate the application's use of a target machine's memory hierarchy. Statistics like expected cache hit rates and memory operation counts are generated by simulating the interaction of the address stream of the application against the memory subsystem of a desired target machine based on user-supplied inputs about the target machine such as cache sizes, associativities, line sizes, etc.

The tool used for instrumentation is the ATOM toolkit [5], where the full simulation of an application incurred a penalty of three orders-of-magnitude slowdown versus uninstrumented code. In [6], cache simulation was performed only at regularly spaced intervals thereby exploiting inherent program stability to decrease the number of instructions that must be simulated to maintain an accurate signature. This technique has reduced the number of instructions that must be simulated to just below 10% while maintaining almost full accuracy, still costing two orders-of-magnitude slowdown over regular execution.

Analysis of well-optimized memory tracing tools indicates that an appreciable amount of the slowdown comes from actually performing cache simulations as opposed to instrumentation overhead. This is shown to be true for ATOM-based MetaSim Tracer in Table 1, where the first column shows uninstrumented runtime, the second column shows instrumented runtime with full cache simulation, and the third columns shows instrumented runtime without cache simulation for the NAS Parallel Benchmarks. The slowdowns given just by instrumentation overhead can be considered a lower bound on the performance of trace driven simulation (if the simulation were infinitely fast) with a certain amount of instrumentation overhead.

This leads to the observation that reduction of the number of instructions that are simulated can dramatically reduce the slowdown that MetaSim Tracer has on an application. The benefits of decreasing slowdown are obvious: one would be able to profile longer running applications and be able to profile currently studied applications more quickly. Simulating one or two orders of magnitude fewer instructions than a full simulation would facilitate the approach of a lower bound on the slowdown of an instrumentation-based trace-driven simulator. In this work, cache simulation is done selectively based on techniques developed in SimPoint, which reduces the number of instructions that must be simulated by an order or two of magnitude while maintaining most of the accuracy of total simulation.

2 Simulation Using Periodic Intervals

In order to show the effect that intelligent selection of simulation intervals has on the accuracy of a trace driven simulation, we wish to compare it to profiles gathered with regular periodic selection of simulation intervals. Consider

Benchmark	Code E.T.	Full Trace Execution Time (Slowdown)	Full Instrumentation Execution Time (Slowdown)
bt.W.4	3.23s	16715s (5175x)	49.6s (15.3x)
bt.W.9	1.79s	7687s (4294x)	23.2s (13.0x)
bt.W.16	1.43s	4392s (3071x)	14.0s (9.8x)
cg.A.4	1.39s	8121s (5842x)	20.2s (14.5x)
cg.A.8	0.89s	4064s (4566x)	13.5s (15.2x)
cg.A.16	0.75s	2167s (2889x)	6.6s (8.8x)
ep.A.4	9.54s	9651s (1012x)	35.9s (3.8x)
ep.A.8	4.75s	4841s (1019x)	18.2s (3.8x)
ep.A.16	2.38s	2428s (1020x)	9.0s (3.8x)
is.A.4	3.38s	4399s (1301x)	16.2s (4.8x)
is.A.8	2.09s	2239s (1071x)	10.6s (5.1x)
is.A.16	1.23s	1170s (951x)	6.0s (4.9x)
lu.W.8	4.62s	19481s (4217x)	60.9s (13.2x)
lu.W.16	3.67s	9969s (2716x)	34.9s (9.5x)
mg.A.4	3.19s	12378s (3880x)	35.0s (11.0x)
mg.A.8	1.71s	5995s (3506x)	17.7s (10.4x)
mg.A.16	0.97s	3086s (3181x)	9.3s (9.6x)
sp.W.4	9.39s	40162s (4277x)	110.7s (11.8x)
sp.W.9	5.75s	18182s (3162x)	55.1s (9.6x)
sp.W.16	4.45s	10207s (2294x)	32.3s (7.3x)

Table 1. *Relative cost of instrumentation and trace analysis using ATOM, showing that speedup can be expected by reducing the amount of cache simulation.*

a program with a dynamic instruction count of n for which we wish to obtain an application signature. A periodic trace simulates $X\%$ of the instructions of this program and does so over p regularly spaced intervals. We do not simulate any of the first 10% of the instructions of the program in order to avoid a cold-start bias in our results. To calculate how many instructions to fast forward (fwd_amt), how many instructions should be simulated in each period (on_amt), and how many instructions should not be simulated in each period (off_amt), we use the following equations:

$$fwd\_amt = n/10 \tag{1}$$

$$on\_amt = \frac{(n - fwd\_amt)\frac{X}{100}}{p} \tag{2}$$

$$off\_amt = \frac{(n - fwd\_amt)\frac{100-X}{100}}{p} \tag{3}$$

From (1), (2) and (3), we can see that (fwd_amt), (on_amt) and (off_amt) can be calculated from n, X and p, the latter of which are provided to MetaSim so that the former can be calculated automatically.

3 Simulation Using SimPoint Intervals

Many programs exhibit structure and repetition in their dynamic execution behavior stemming from static code structures such as loops. For this reason, blind sampling, regularly spaced and at high frequency, often leads to a high probability of capturing salient program behavior including cache behavior. However, more advanced analysis of static and dynamic code structure can allow for informed selection of specific intervals of execution which are representative of entire program behavior.

SimPoint [2] is a tool which utilizes light-weight dynamic basic block traces to feed an offline phase classification algorithm which determines representative simulation points for a program. The intervals defined by these simulation points can be relatively small and not necessarily regularly spaced. Moreover, information from offline analysis includes weights for each simulation point, indicating the extent to which behavior from that interval of execution is representative of overall program behavior.

The SimPoint-guided version of MetaSim Tracer runs cache simulation only on the memory references which are in one of the application's selected simulation intervals. After execution of the program, MetaSim has the memory-use statistics for each basic block during each SimPoint interval. Let n be the number of simulation points chosen for a given program, and $weight_i$ be the relevance of interval i to the execution of the program as given by SimPoint (the fraction of the program that the i^{th} SimPoint interval represents). Also, let $L1hit_{ij}$ and $L1miss_{ij}$ be the number of L1 cache hits and L1 cache misses respectively for basic block j that occurred in simulation point interval i. We can calculate the L1 cache hit rate of basic block j by the formula

$$\sum_{i=1}^{n} \left(weight_i \frac{L1hit_{ij}}{L1hit_{ij} + L1miss_{ij}} \right) \tag{4}$$

4 Methodology

Results for MetaSim are gathered on Lemiuex [7], located at the the Pittsburgh Supercomputing Center. Lemieux is comprised of 750 Compaq Alphaserver ES45 nodes, each of which has 4 1 GHz processors and 4 GB of memory. The IBM Power3 architecture is our target architecture (the machine for which we are predicting cache hit rates). The Power3 being modelled has a 64 KB 128-way set associative L1 cache and a 4 MB 4-way set associative L2 cache, both with line sizes of 16 words (128 bytes).

We report benchmark results from the NAS Parallel Benchmark suite, version 2.3 [8]. The kernel benchmarks (CG, EP, IS and MG) are tested at size A on 4, 8 and 16 pro-

cessors and the application benchmarks (BT, LU and SP) are tested at size W on 4, 9[1] and 16 processors.

We use Simpoint version 1.0 to gather simulation points for each program. Interval size is set to 1×10^6 instructions, because it most closely approximates the proportion of interval size to program size used in [2]. We set the number of simulation points to a fixed number, 8, simply because we found by experimentation that there was typically no noticeable gain in accuracy when using more intervals, and some loss in accuracy could occur when using fewer intervals.

5 Results

There are two primary findings. The first is that the number of instructions that must be simulated is greatly reduced by guiding cache simulation with SimPoint-produced intervals, resulting in much faster simulation. The second is that these same simulations have results that are comparable in accuracy to periodic sampling-based simulations. In other words, SimPoint-guided simulation is faster than 1% periodic sampling-based simulation and as accurate as 10% periodic sampling-based simulation, a clear case of win-win.

5.1 Simulation Speed

Earlier, we stated that usefully accurate SimPoint-guided simulations simulated two orders-of-magnitude fewer instructions than a full trace. Such a large factor is possible because one is able to control the amount of simulation one wishes to perform according to SimPoint's two parameters: interval size and number of simulation intervals. Though a similar decision can be made when using periodic sampling-based simulation, it is likely that accuracy will decay as the amount of simulation is reduced.

Table 2 shows three things: dynamic instructions traced, time to perform tracing, and per-basic-block cache hit rates (discussed below). In the second major column we report on total number of dynamic instructions that were simulated for each of the benchmarks. Note that there is a drastic decrease in the number of instructions simulated by SimPoint-guided version of MetaSim. For most benchmarks, this is below the percentage of instructions simulated by periodic 1% sampling and is significantly lower than the number of instructions simulated by periodic 10% sampling.

In the third major column we report the execution time of our three sampling-based techniques as a percentage of the full simulation execution time for that benchmark. Simpoint-guided tracing shows reduced execution time of up to four-fold over 1% sampling with much larger reductions over 10% sampling (an order of magnitude). Importantly, the baseline comparison is to full tracing execution time – so reductions of even 5% are meaningful in real time but the ATOM-implemented SimPoint-guided tracer typically saves two orders-of-magnitude over a full simulation.

[1] LU Benchmark uses 8 instead of 9 processors

5.2 Simulation Accuracy: L1 Cache Hit Rate

The standard definition of an L1 cache hit rate involves dividing the number of L1 cache hits that occur in a program by the total number of memory accesses for that program. This definition can be impacted greatly by sampling, as the unpredictable nature (unpredictable with respect to which parts of the code will be sampled) of sampling can cause regions of lesser importance to contribute more heavily to the overall sampled cache hit rate.

However, this is not the L1 cache hit rate used by the MetaSim Convolver[2] to perform performance predictions. Instead, MetaSim Convolver uses a more fine-grained calculation that combines a hit rate for each basic block and that basic block's frequency in the actual program as opposed to its frequency in some subset of the program (e.g., the subset of the program that is examined in the sampled portions a partial program trace). We examine the accuracy of various types of sampling for the MetaSim definition of L1 cache hit rate, a definition which is less likely to produce misleading results on sampled traces. We then show more detailed results of cache hit rates on the basic block level.

5.2.1 Overall Program Cache Hit Rates

In Table 2 we show the accuracy of sampled overall cache hit rates for each type of sampling (as compared to full tracing). Note that, overall, the difference from full trace cache hit rates is very small (usually less than 1%) for all types of sampling. In general, SimPoint sampling produces more accurate hit rates than 1% sampling. There are only 2 cases for which the SimPoint-guided simulation is notably worse than the 10% sampling simulation (by notably worse we mean a cache hit rate that differs by greater than 1%), cases which are discussed in a later section.

For full traces, 1%, and 10% sampling traces, the Metasim-style program cache hit rates are calculated by summing up the cache hit rates of each basic block, each multiplied by a weight for each basic block. This weight (not a SimPoint weight) is determined by the frequency of execution of the basic block in a full trace (gathered in a separate, non-cache simulating run for Metasim). For SimPoint sampled runs, the same process is followed, but the hit rate for each basic block is calculated using Equation 2.

5.2.2 Finer Granularity: Basic Block Cache Hit Rates

MetaSim-style overall L1 cache hit rate is designed to be more accurate when used with sampled tracing, but the finer

[2] MetaSim Convolver[9] is the part of the MetaSim performance model that combines the application signature with the machine profile to aid in performance prediction.

	# of Instructions Simulated		Execution Time (% of Full)			Per BB L1 Cache Hit Rate (Difference from Full)		
Benchmark	Full Trace	SimPoint	SimPoint	Per 10%	Per 1%	SimPoint	Per 10%	Per 1%
bt.W.4	4324626467	0.18%	0.46%	10.69%	1.62%	−0.50%	−0.02%	−0.04%
bt.W.9	1967188101	0.41%	0.64%	14.43%	2.39%	−0.21%	0.06%	0.29%
bt.W.16	1131606549	0.71%	0.82%	16.50%	0.83%	0.01%	0.13%	0.55%
cg.A.4	1774258202	0.45%	0.54%	13.43%	1.90%	−0.36%	−0.07%	0.26%
cg.A.8	975634405	0.82%	1.03%	15.28%	2.20%	−0.22%	−0.02%	0.38%
cg.A.16	536630503	1.49%	1.84%	16.66%	2.34%	−0.18%	−0.04%	0.68%
ep.A.4	3950107069	0.20%	0.92%	10.85%	1.99%	−0.02%	0.00%	−0.01%
ep.A.8	1975224512	0.41%	0.96%	11.15%	2.19%	−0.02%	−0.03%	−0.02%
ep.A.16	987732588	0.81%	1.03%	11.78%	2.67%	−0.03%	−0.02%	−0.03%
is.A.4	2480817023	0.29%	0.98%	12.0%	2.02%	−0.11%	0.08%	0.61%
is.A.8	1242395861	0.52%	1.03%	12.1%	2.08%	−0.09%	0.15%	0.92%
is.A.16	620853957	1.30%	1.38%	12.25%	2.17%	0.15%	0.27%	1.30%
lu.W.8	4908689632	0.16%	0.56%	18.49%	2.78%	−0.01%	0.02%	0.05%
lu.W.16	2164713698	0.37%	0.75%	20.93%	3.54%	−0.09%	0.27%	0.62%
mg.A.4	4191963949	0.17%	0.46%	11.7%	1.57%	−0.09%	0.03%	0.31%
mg.A.8	2088488215	0.37%	0.47%	11.8%	1.60%	−0.38%	0.04%	0.40%
mg.A.16	1071537525	0.71%	0.53%	18.8%	1.73%	−0.86%	0.08%	0.69%
sp.W.4	10122113343	0.08%	0.47%	11.4%	1.66%	−0.83%	0.02%	−0.51%
sp.W.9	4539711740	0.17%	0.61%	16.1%	2.49%	−0.71%	0.02%	0.28%
sp.W.16	2538859116	0.32%	0.72%	19.1%	3.22%	−0.11%	0.08%	0.59%

Table 2. *Amount of simulation required (second main column), execution cost of each simulation variant (third main column) and per-basic-block cache hit rates (fourth main column).*

granularity of per-basic-block cache hit rates is important both in making performance predictions in the MetaSim framework and for evaluating the impact of sampling.

Figures 1-3 show L1 cache hit rate statistics gathered on a per-basic-block basis. In each figure, the width of a particular segment of the graph represents the weight (frequency of execution in a full trace) of the basic block being represented by that particular segment of the line. In other words, the wider a given line is, the more important/frequent that particular basic block is. For convenience, the basic blocks are also sorted by frequency.

The difference in the estimated cache hit rate between a particular trace and a full trace can be seen by calculating the area underneath the line for a particular trace. For example, in Figure 2, the estimated L1 cache hit rate of the 3rd highest weighted basic block in the 1% periodic trace differs from the estimation for that rate in the full trace by about 0.65%. In comparison, the inaccuracy for this block has more effect than the second highest weighted basic block in Figure 2, which only exhibits a 0.27% inaccuracy. Moreover, the fact that some basic blocks of relatively little weight/importance have high levels of inaccuracy (off of the Y-axis of the chart) is quite unimportant, given their extremely small contribution to the overall area.

Here we see 3 such comparisons based on a loose classification of the accuracy of the partial traces. Figure 1 is representative of benchmarks for which the SimPoint-guided trace is more accurate with respect to L1 cache hit rate than both 10% and 1% periodic traces; 47.8% of the NPBs fall into this category. Figure 2 is representative of benchmarks for which the SimPoint-guided simulation is more accurate than the 1% sampling simulation but not as accurate as the 10% sampling simulation; 8.7% of the NPBs studied belong to this class. Figure 3 is representative of the benchmarks for which the SimPoint-guided simulation is more accurate than the 1% simulation but neither more nor less accurate than the 10% simulation; such was the case with 17.4% of the benchmarks studied. We consider all of the above cases, which account for 73.9% of the benchmarks studied, to be cases which show the SimPoint-guided traces to be of high accuracy.

Not shown is a representative from the benchmarks for which all three non-full simulations are nearly equal in accuracy; this case accounts for 17.4% of the benchmarks studied and does not necessarily contradict the notion that SimPoint is accurate. For this case, which is accounted for by the EP (Embarrassingly Parallel) benchmark, the code is simple enough so that any reasonably well-conceived simulation scheme will accurately capture program behavior. It comes as no surprise then that all of the simulation schemes have similarly high levels of accuracy on this code.

Also not shown is a representative from the benchmarks for which SimPoint-guided simulations are less accurate than both the 1% and 10% simulations. In the SimPoint-

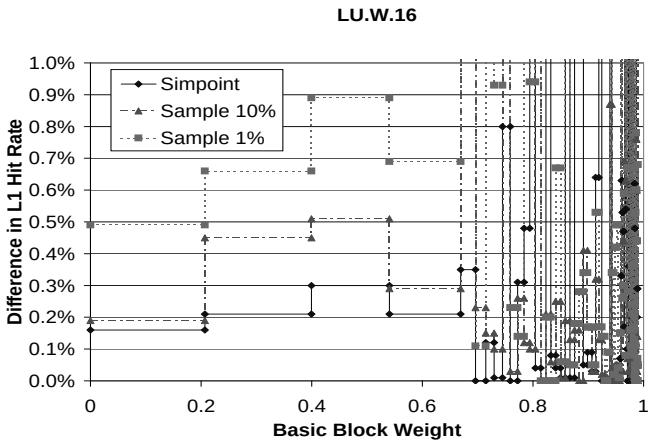

Figure 1. *SimPoint-guided simulation is more accurate than both* 10% *and* 1% *sampling-based simulations.*

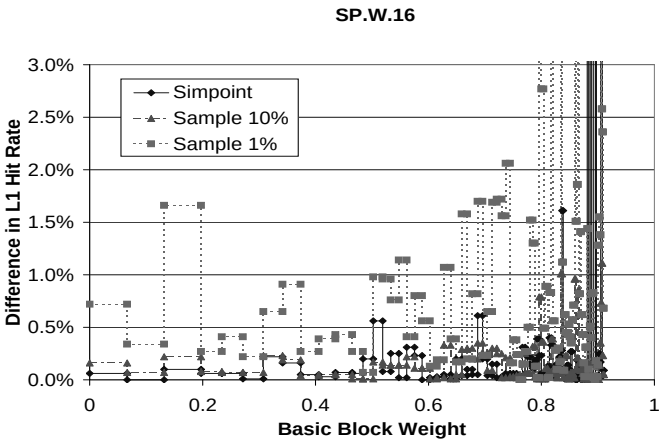

Figure 3. *SimPoint-guided simulation is more accurate than* 1% *sampling-based simulation about as accurate as* 10% *sampling-based simulation .*

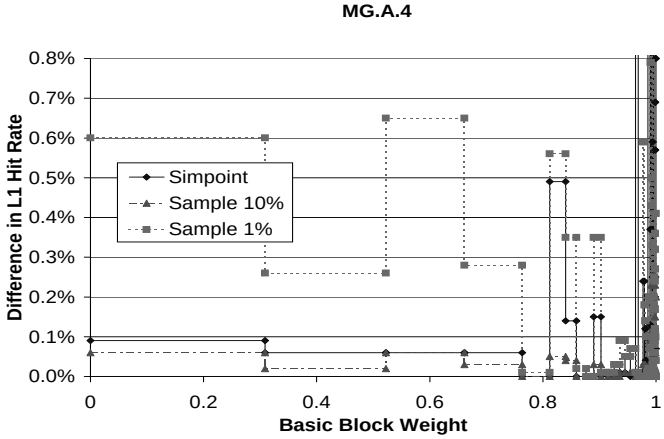

Figure 2. *SimPoint-guided simulation is more accurate than* 1% *sampling-based simulation and is less accurate than* 10% *sampling-based simulation.*

guided simulations, an L1 cache hit rate of 100% is assumed for any basic block for which there is no data – that is a basic block that SimPoint did not select to be part of any traced interval. With the lack of any other evidence, we assume that this basic block will not result in any cache misses. This can clearly be incorrect, but for our benchmarks, was more accurate than making the alternate assumption – that the basic block has a 100% cache miss rate. This case is accounted for by the SP (Scalar Pentadiagonal) benchmark for reasons that are currently under investigation. We speculate that this has occurred because there are too many sections of the code during which different important basic blocks are being executed (i.e., there are a lot of different relatively smaller loops that each perform a significant amount of computation). If this is the case, using only 8 SimPoint intervals for simulation may not properly characterize the execution of the entire code. While only 3 graphs are presented here, the full complement of graphs can be viewed at `http://users.sdsc.edu/~michaell/SimpointStudy/simpointGraphs.html`.

In summary, SimPoint-guided cache simulations are about as accurate as 10% periodic trace for L1 cache hit rate. We can observe that the overall cache hit rate of SimPoint-guided traces differ from that of a full simulation by an average of 0.25%, while the 1% and 10% periodic traces differ from the full trace by an average of 0.43% and 0.07% respectively.

5.3 AVUS: Why Performance Matters

The goal of this work is to enable trace driven simulations of real applications; here we demonstrate that full and and 10% sampling-based simulations are far too computationally expensive for even modestly sized applications. A small test version of the Air Vehicles Unstructured Solver (AVUS) [10] , without any sort of instrumentation, executes for 385 seconds. This small version iterates for only 5 time steps, where the full version iterates 100 times. We are able to perform a SimPoint-guided and 1% sampling-based simulations on this code, which require 8435 seconds and 17098 seconds for execution, for respective slowdowns of approximately 22x and 44x. An attempt to complete a 10% sampling based simulation failed due to a 12-hour queue limit. To estimate the slowdowns of performing a full simulation and a 10% sampling-based simulation, let us use the average slowdowns (from the NPBs) of 2792-fold and 432-

fold respectively. In order to perform a full simulation of this small AVUS case, it would require 1074920 seconds (or about 298 hours) of execution. In order to perform a 10% sampling-based simulation, it would require 166320 seconds (or about 46 hours) of execution. Obviously it is imperative to utilize a technique which reduces these times. Based on the levels of accuracy shown previously and the fact that our only realistic options are the 1% sampling-based simulation and the SimPoint-guided simulation, it is preferable to use a SimPoint-guided simulation because of the likelihood of having higher accuracy than the 1% periodic sampling-based simulation. Also, this halves execution time.

6 Future Work

A binary instrumentation tool called Pin has been developed by researchers at Intel [11]. Pin supports Linux binaries on Xscale, IA-32 and Itanium families of processors and aims to provide similar but more flexible functionality to the ATOM toolkit. Preliminary testing of Pin has given us optimism in its speed, which may make it a suitable tool for instrumentation-based memory simulation. This is useful to us because it could allow us to run instrumented programs on some Intel platforms instead of being limited to the Alpha platform, a limitation imposed by ATOM. Another instrumentation tool called Dyninst [12] is supported on Alpha, MIPS, Power/PowerPC, SPARC, x86, and ia64 platforms, which provides a lot of portability. The speed of Dyninst is a large inhibitor of its utility at this point, but we anticipate that this issue will be addressed in the near future.

For each of the benchmarks used to test SimPoint-guided profiling tool, each process of those benchmarks exhibits similar behavior. This assumption of homogeneity reduces the complexity of obtaining a SimPoint-guided profile because it allows the use of a single set of SimPoint intervals and weights on all processes of a parallel application. To perform a SimPoint-guided trace on an application which has heterogeneous behavior across its processes, we must be able to identify a process before it begins to execute (or at the very beginning of its execution) in order to correlate it with a set of SimPoints. This is something which must be done to make a SimPoint-guided profiling tool useful for a larger and more useful class of applications.

7 Conclusions

In this paper, we have shown that the SimPoint phase analysis tool can be used to guide the MetaSim Tracer to trace as few as 0.08% of the instructions of a program while still retaining the accuracy of the details of the profile that are necessary to make a precise performance prediction. As an important example, program-wide L1 cache hit rate deviates from the rate predicted by a full simulation by an average of 0.25%. This can have a great impact on the performance of an application profiling tool because it can dramatically decrease the slowdown that such a tool has on an application, giving the potential to profile larger applications and to more quickly profile current applications.

This work was supported in part by NSF award CNS-0406312, the DOE Office of Science through the award entitled HPCS Execution Time Evaluation, by NSF NGS Award 0406312: Performance Measurement and Modeling of Deep Hierarchy Systems, and by the Department of Energy Office of Science through SciDAC award High-End Computer System Performance: Science and Engineering. Computer time partly provided by an NSF NRAC award.

References

[1] Snavely, A., Wolter, N. and Carrington, L. *Modeling Application Performance by Convolving Machine Signatures with Application Profiles.* IEEE 4th Workshop on Workload Characterization, December 2001.

[2] Sherwood, T., Perelman, E., Hamerly, G. and Calder, B. *Automatically Characterizing Large Scale Program Behavior.* International Conference on Architectural Support for Programming Languages and Operating Systems, October 2002.

[3] Gao, X., Simon, B. and Snavely, A. *ALITER: An Asynchronous Lightweight Instrumentation Tool for Event Recording.* Workshop on Binary Instrumentation and Applications, September 2005.

[4] Carrington, L., Snavely, A., Gao, X. and Wolter, N. *A Performance Prediction Framework for Scientific Applications.* Workshop on Performance Modeling, June 2003.

[5] Srivastava, A. and Eustace, A. *ATOM: A Flexible Interface for Building High Performance Program Analysis Tools.* USENIX Winter Conference, January 1995.

[6] Gao, X. and Snavely, A. *Exploiting Stability to Reduce Time-Space Cost for Memory Tracing.* Workshop on Performance Modeling - ICCS, June 2003.

[7] Lemieux User Information Page. http://www.psc.edu/machines/tcs/lemieux.html, July 2004.

[8] Bailey, D. *The NAS Parallel Benchmarks.* Intl. Journal of Supercomputer Applications, vol. 5, no. 3, Fall 1991.

[9] Snavely, A., Carrington, L., Wolter, N., Labarta, J., Badia, R. and Purkayastha, A. *A Framework for Application Performance Modeling and Prediction.* Supercomputing, November 2002.

[10] Ramamurti, R. and Lohner, R. *A Parallel Implicit Incompressible Flow Solver Using Unstructured Meshes.* Computers and Fluids , Vol. 25, No. 2, pp. 119-132, 1996.

[11] Pin User Manual. http://rogue.colorado.edu/Pin/, November 2003.

[12] Buck, B. and Hollingsworth, J. *An API for Runtime Code Patching.* Journal of High Performance Computing Applications 14 (4), Winter 2000.