

Accelerating Architectural Simulation by Parallel Execution of Trace Samples

Gary Lauterbach

SMLI TR-93-22

December 1993

Abstract:

In order to quickly decide which architectural features are to be included in future processors, we have developed a simulation approach that uses samples of benchmark program instruction traces. Rather than simulating a proposed architecture on the entire SPEC92 program suite of more than 100 billion instructions, we simulate using a set of samples of the SPEC92 suite containing less than 1% of the total instruction trace. Each of our samples contains a relatively short instruction trace that can be simulated quickly.

The technique described can be applied to existing architectural models to produce significant reductions in simulation time. Existing simulation tools can be leveraged to implement the trace sampling technique described.

 *Sun Microsystems
Laboratories, Inc.*

A Sun Microsystems, Inc. Business

M/S 29-01
2550 Garcia Avenue
Mountain View, CA 94043

email address:
gary.lauterbach@eng.sun.com

Accelerating Architectural Simulation by Parallel Execution of Trace Samples

Gary Lauterbach

Sun Microsystems Laboratories, Inc.
2550 Garcia Avenue
Mountain View, CA 94043

1.0 Introduction

The runtime required to simulate new architectures becomes prohibitive with the increasing size of the industry standard benchmarks such as the SPEC92 suite. The SPEC92 suite requires that more than 100 billion instructions be simulated on an architectural model. With simulation speeds on the order of tens of thousands of instructions per second, simulating the entire benchmark suite could take more than 100 days. This long simulation time would severely restrict the number of architectural alternatives that could be examined in a normal processor development project.

The technique we use to reduce architectural simulation times makes use of samples of the benchmark programs so that only a small fraction of the entire instruction trace needs to be simulated. We have found that it is possible to simulate less than 0.5% of the instruction trace of the SPEC92 benchmark suite to obtain accurate performance estimates of future processors. The reduction in the number of instructions simulated results in a 200-to-1 speedup, but this is not the limit of the speedup available from sampling. If each of the samples is simulated on an independent processor, another speedup proportional to the number of samples is available. In our case, with more than 1000 samples of the SPEC92 benchmarks, we can potentially achieve an additional 1000-to-1 speed improvement from parallel processing. The total reduction in simulation time could exceed 200,000-to-1, reducing the turnaround time of architectural simulations from 40 days to just 17 seconds.

Previous architectural studies [1,2,3] have used a form of sampling to reduce the runtime of the simulations. The technique used in these studies was to simulate the first N instructions of each of the benchmark programs for some large value of N. Generally, several tens of millions of instructions from the beginning of each benchmark were simulated. The limitation of this technique is

twofold: 1) a single large sample might not be representative of the program behavior; and 2) with only one sample, there is no additional speedup available from using more than one processor per benchmark by distributing the samples from the benchmark across many processors. The first limitation is of most concern since it calls into question the validity of the architectural simulation results. We know that for some of the SPEC92 programs, a single large sample placed at the beginning of the program's execution is not representative. For instance, the SPEC benchmark program 093.nasa6 is a series of seven kernels that are sequentially executed. Assuming each kernel executed the same number of instructions, then if we were to use the first half of the instruction trace as our one large sample, it would not include any of the execution of the last three kernels.

Our approach uses a moderate number of relatively short instruction trace samples. We start by taking fifty samples of each benchmark program. Each sample contains a trace 250,000 instructions long. The samples are taken at even instruction intervals in the execution of the benchmark. For instance, if the program executes a total of 10^9 instructions, our initial samples would be placed every $2 \cdot 10^7$ instructions. We then compare a number of measurements (described in more detail in Section 6.0) of the sample set to measurements of the benchmark program. If the sample set is not representative, we add to the sample set by taking more samples evenly spaced between the original samples. We continue to add samples to the sample set until it is representative.

The use of short samples could lead to large inaccuracies in the simulation of some parts of a processor. Martonosi et al. [4] discuss the effect that sample length can have on the simulation accuracy of large caches. A major component of the simulation inaccuracy is because the state of the cache is unknown at the start of the sample instruction trace. Since we are interested in observing pipeline behavior in the presence of cache misses, the samples must be representative of the miss behavior of the program. Rather than increasing the sample size until the misses due to the unknown initial cache state become statistically insignificant, we initialize the cache to the correct state at the start of the sample instruction trace. This allows us to keep the sample size small and, consequently, maintain a large speedup. Producing the correct initial cache state at the start of the sample instruction trace means that we must perform cache simulations during the sampling process for the entire execution of the sampled program. Since we don't know *a priori* which cache configuration will be of interest for our architectural simulations, we use a version of stack simulation [5] to simulate a multitude of caches concurrently. This lets us select a single cache state from the multitude saved in the sample and use it as the initial cache state for the architectural simulation.

All of our architectural investigations are based on the SPARCTM instruction set [8], a fairly typical RISC instruction-set architecture. The following descriptions of techniques assume SPARC instruction traces and formats, but should be applicable to other instruction-set architectures.

2.0 Sample Format

Each sample is written to disk as a separate file. The extension of the filename is used to give each sample a unique name. For instance, the first sample of a set will have filename FILE.0, the second FILE.1, etc. We then organize the samples into a directory structure with one directory per SPEC benchmark that contains all of the samples of that program.

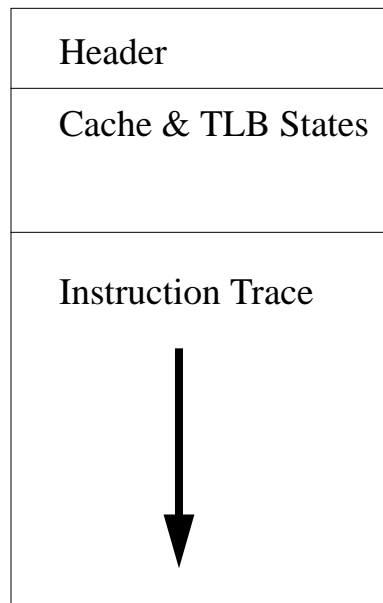


FIGURE 1. Trace Sample Structure

Figure 1 shows the structure of a sample file. Each sample contains three components: a header, a collection of cache and TLB tag states, and the instruction trace. The sample header contains the following information:

- Time and date stamp
- Starting PC of the instruction trace
- Command line used to create the sample
- Instruction count at start of sample

This header information is used for organizational purposes such as being able to unambiguously determine from which benchmark any sample was taken.

After the header, the tag state of the caches and TLBs that were simulated during sampling are stored. We use a version of stack simulation to build the tag state for a large range of caches while sampling the program. For each range of caches being simulated, a description of the range along with the state of the cache tags is written to the sample file. For example, one cache range might be described as 1kb to 64kb, one through four set associative, 16-byte line caches.

The last section of the file contains the instruction trace in an encoded format to minimize the disk space requirements of saving a set of samples. The instruction trace stored in the sample is used to stimulate an architectural model in exactly the same way that the SHADE instruction tracer would stimulate an architectural model.

3.0 Sample Compression

The SPARC instruction tracer (SHADE [6]) produces a 14-byte structure to describe each instruction traced. For each instruction the following information is provided:

- Program counter (4 bytes)
- Instruction (4 bytes)
- Effective memory address (4 bytes)
- Annulment flag (1 byte)
- Branch direction flag (1 byte)

If we were to write this structure to disk, in this format, the instruction trace portion of a 250,000 instruction sample would be 3.5 megabytes. The first step in reducing the space used for a sample is to more efficiently encode the per-instruction trace data. We do this by recognizing that much of the data passed by SHADE for each instruction is unnecessary. The Program Counter can be regenerated from the trace except for computed jumps. The Effective Address is only needed for memory reference instructions. The Annulment flag can be regenerated by looking at the branch direction. We only save the instruction word for each instruction in the trace—the effective address if the instruction references memory—and the branch direction for branch-type instructions. A future optimization would be to eliminate the instruction itself by saving the entire code image once for a set of samples, and referring to it to obtain the instructions. The trace would then just consist of effective addresses, branch directions, and branch target addresses for computed jumps.

After encoding the trace data and writing it to a file, we use the UNIX[®] compress utility to further reduce the sample size. The set of 1200 samples of the SPEC92 programs occupies about 600 Mb. of disk space, on average, 0.5 Mb. per sample.

4.0 Validation

The speed improvement provided by trace sampling would be useless if the samples were not representative of the benchmark programs. We use several metrics to determine when the sample set is representative. The validation procedure is divided into two components: fast checks to determine if the sample set is clearly insufficient; and more time consuming checks that are done as a final check that the sample set is accurate. The fast checks use instruction frequency, function frequency, and cache statistics to determine when we have a sufficient set of samples. The speed of these fast checks is determined by the time required to collect these metrics on the complete instruction trace. The speedup from trace sampling allows us to measure the same metrics very quickly on the sample set.

The slow final validation compares the simulated performance of a set of micro-architectures stimulated by the samples to the simulated performance of the same micro-architectures stimulated by the complete instruction trace. The time required to simulate the micro-architectures on the complete instruction traces qualifies this check as the slow final validation. Several of the SPEC92 benchmark programs require more than one week to perform a micro-architectural simulation on the complete instruction trace.

4.1 Instruction statistics

The first metric we use to test the quality of the samples is the frequency of each instruction type. We compare the dynamic frequency of each instruction type (Ld, St, Add, Sub, Or...) from the complete instruction trace to the frequency from the samples. The SPIX utilities [7] provide the instruction frequency data for the complete trace. A small modification of these utilities enables them to also function on the samples. Figure 2 shows the dynamic instruction frequencies of the seventy-seven instruction types used in the SPEC92 benchmark 085.gcc. The samples very closely reproduce the instruction frequencies of the complete trace. The largest difference in dynamic instruction frequency is less than 0.5%. In the interest of brevity, each metric is illustrated for only one of the benchmarks in the SPEC92 suite.

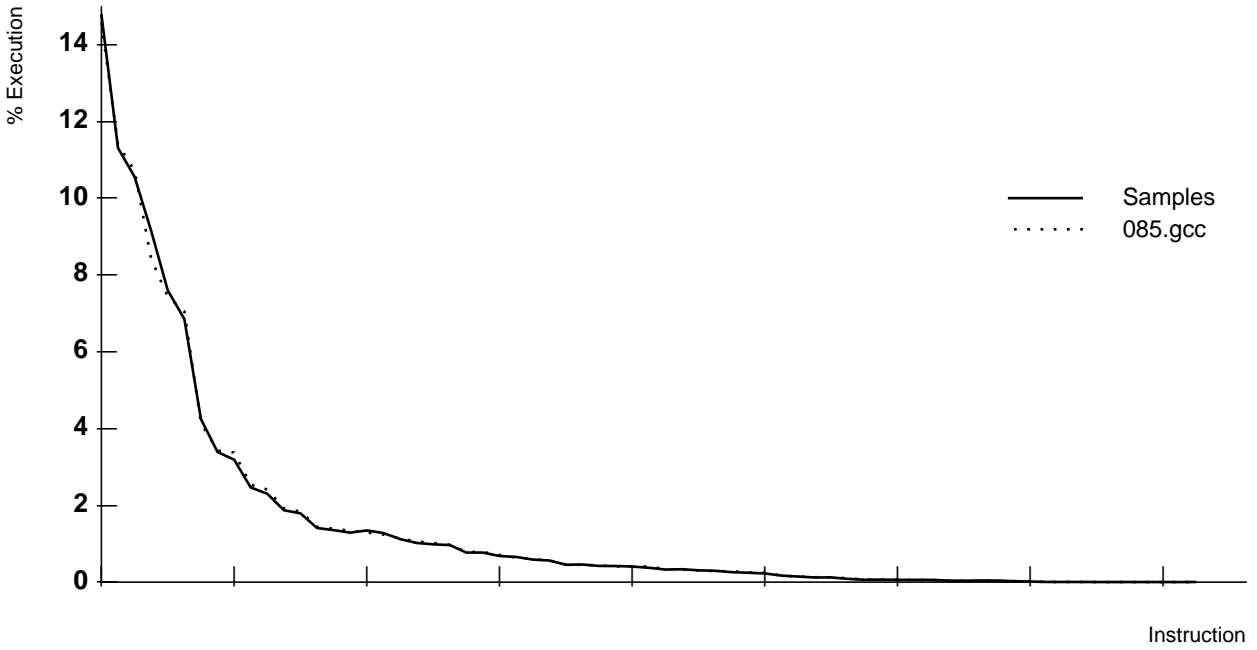


FIGURE 2. Instruction Frequencies — Samples vs. Trace

4.2 Function histogram

The instruction–frequency statistics may indicate that the samples reflect the correct mix of instructions, but the instruction sequences or inter-instruction dependencies can still not be accurately reflected. We compare the percentage of instructions occurring in each sub-routine from the samples to that of the complete program trace to verify that we are getting a representative set of instruction sequences.

Figure 3 compares the percentage of instructions executed from the 200 most significant functions of the SPEC92 benchmark 085.gcc, sorted from most frequent to least frequent function.

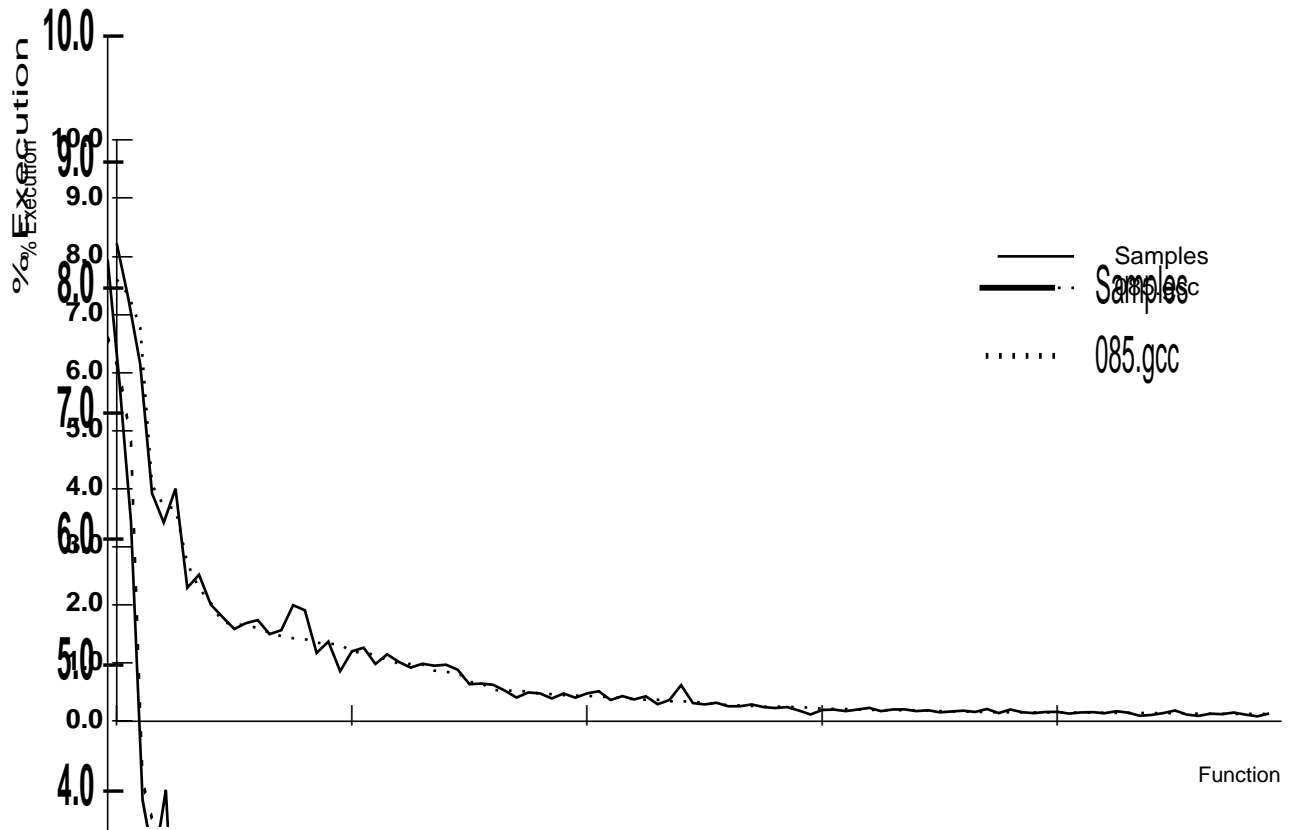


FIGURE 4. Function Frequencies—Samples vs. Complete Trace

4.3 Cache statistics

The only dynamic data contained in our instruction traces are the memory-reference addresses. The architectural models we use to evaluate trade-offs are stimulated from instruction traces and thus are independent of the content of memory or registers. The memory-address trace is required to stimulate the cache / TLB / memory subsystem models of the architectural simulation. To verify that the samples contain representative memory-address traces, we compare the miss rates of a large range of cache simulations between the samples and the complete trace.

Figure 4 shows how the miss rates for caches between 4K bytes and 1M byte in size with two different line sizes, 16 bytes and 32 bytes per line, compare to the same caches simulated on the entire memory address trace. We chose the SPEC92 benchmark 093.nasa6 for the cache-miss rate comparison because it has a significant miss rate for a wide range of caches.

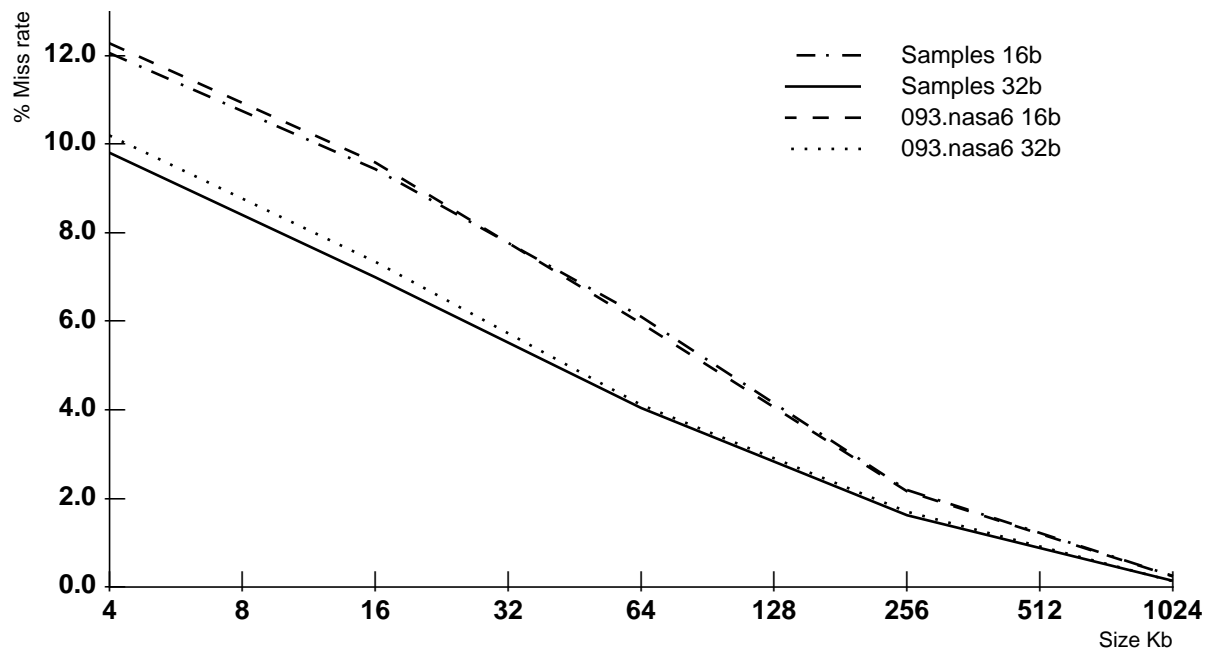


FIGURE 4. Cache Miss Rates — Samples vs. Complete Trace

4.4 Micro-architecture comparison

The final validation of a sample set is to simulate a number of micro-architectures using the samples as stimuli, and compare the results to previous simulations of the same micro-architectures on the complete instruction traces. Figure 5 shows the SPEC92 INT performance of fourteen different micro-architectures on the trace samples and the complete trace. The horizontal axis spans seven micro-architectures, beginning with a simple single-issue pipeline on the left and increasing in complexity to a four-issue superscalar pipeline on the far right. The two curves correspond to two latencies for memory-reference instructions, one cycle and four cycles, for each of the micro-architectures.

The most notable aspect of Figure 5 is that the sample result curves accurately track the complete trace curves. Since we are interested in using the samples to investigate micro-architectural tradeoffs, it is most important that the samples accurately reflect the performance ratio of two different micro-architectures. Even though we are less concerned about the absolute accuracy of the sample simulations than the relative performance, nonetheless, the absolute performance of samples are within 3% of the performance results of the complete trace.

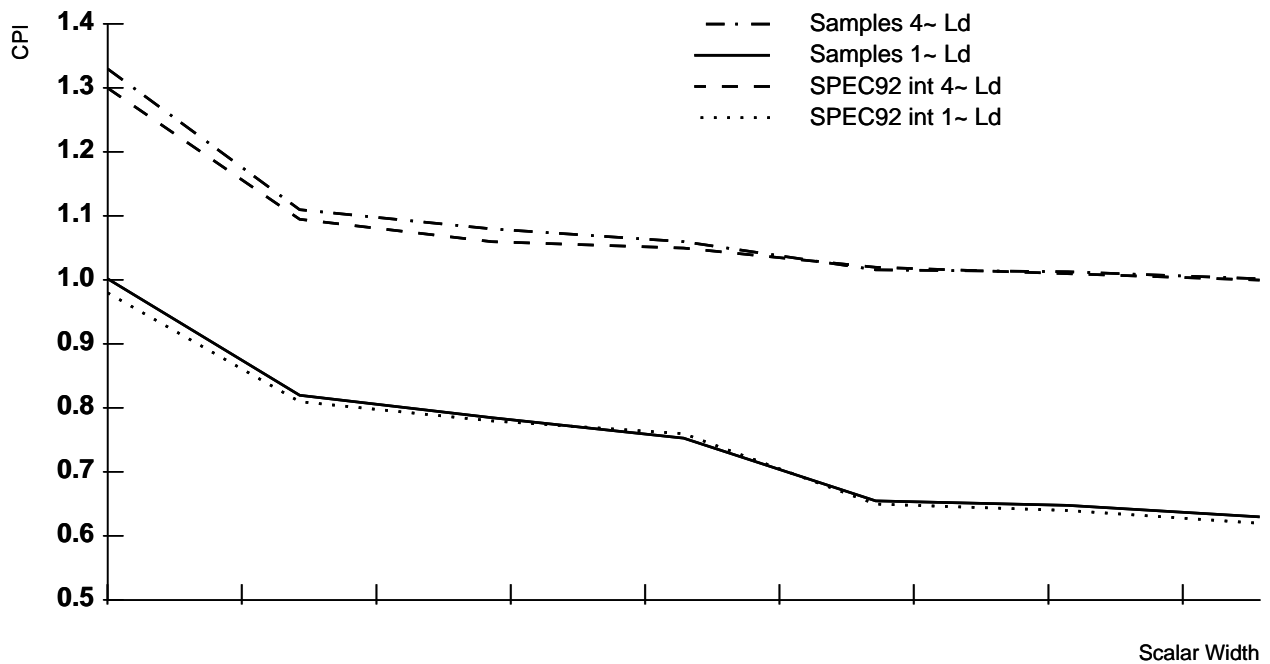


FIGURE 5. Processor SPEC92 INT CPI — Samples vs. Complete Trace

5.0 Parallel Execution

By distributing the simulation of the trace samples across multiple processors, we are able to achieve a speedup that is proportional to the number of processors. We typically distribute the simulations across fifty SPARCstation ELC workstations. Using architectural simulators that simulate 5,000 or more instructions per second, the entire set of 1200 trace samples is completed in less than thirty minutes. Post processing and combining of the results of the simulations of individual samples is done with AWK scripts. The post processing of the simulation results of 1200 samples is accomplished in a matter of a few minutes.

6.0 Limitations

While enabling very fast architectural simulation, our technique does have some accuracy and ease of use limitations. The sampling–verification–resampling process usually takes several days to a couple of weeks of elapsed time to construct a representative set of samples. The total elapsed time depends on the range of caches and TLBs that are simulated while sampling and the number of iterations of the sample–verify process that must be done. The sampler runs at a speed of about 50,000 instructions per second on a SPARCstation ELC; most SPEC92 programs can be sampled in a day or less. If only one or two architectural simulations were required of a particular benchmark, it would be easier and faster to simulate the entire dynamic instruction trace by stimulating the architectural simulator directly from the instruction tracer.

Because the method uses the dynamic instruction trace of a program’s execution to stimulate the architectural simulations, it is difficult to accurately model the behavior of speculative processors. A speculative machine executes some instructions that are not present in the instruction trace because they are on speculative paths that will later be discarded. To approximate the speculative instruction paths, we stimulate the simulation with instructions from the original executable image of the program and use randomly generated effective addresses for the speculative memory reference instructions.

The instruction tracer that we use to generate the trace samples is only able to trace user space instructions. Any activity in system space (system calls, interrupts, traps, etc.) goes unseen, and is therefore not represented in the samples. For the SPEC92 benchmarks, system activity is a fairly small percentage of the execution time; we use analytical methods to adjust the simulation results for the system activity.

7.0 Conclusions

We have developed a viable technique for accelerating architectural simulation to enable a large number of architectural tradeoffs to be investigated in a short period of time. The simulation time is now sufficiently short that the constraining factor becomes the time required to incorporate proposed architectural features into the simulation model.

Trace sampling enables us to accelerate existing architectural models with only very minor modifications. The cache and TLB state in the architectural model must be initialized from the corresponding states in the sample. We have modified more than half a dozen existing architectural models to be compatible with trace sampling. Typically, the changes are completed in less than one half hour.

The method described is able to leverage many existing tools:

- Instruction tracer
- Cache simulator
- Architectural models
- Instruction analyzers

Less than 1000 lines of new “C” code were required to implement the trace sampler, sample compression, sample decompression, and interface library.

8.0 Acknowledgment

Thanks to Valerie Lauterbach and David Poole for comments on drafts of this work. Thanks to David Ditzel for the initial motivation for accelerating architectural simulation, and to Sun Microsystems Laboratories, Inc. for providing an environment in which this could be developed.

9.0 References

- [1] Sohi, G. and M. Franklin. “High-Bandwidth Data Memory Systems for Superscalar Processors.” *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (April 1991): 53–62.
- [2] Lightner, B. “The SPARC Lightning Processor.” *Proceedings of the Hot Chips II Symposium* (August 1990).
- [3] Malik, N., R. Eickemeyer, and S. Vassiliadis. “Interlock Collapsing ALU for Increased Instruction-Level Parallelism.” *Proceedings of the 25th Annual International Symposium on Microarchitecture* (December 1992): 149–157.
- [4] Martonosi, M., A. Gupta, and T. Anderson. “Effectiveness of Trace Sampling for Performance Debugging Tools.” *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (June 1993).
- [5] Hill, M. “Aspects of Cache Memory and Instruction Buffer Performance.” Ph.D. thesis, University of California at Berkeley, Computer Science Division, 1987. Also technical report UCB/CSD 87/381.

[6] Cmelik, R. and D. Keppel. "Shade: A Fast Instruction-Set Simulator for Execution Profiling." *Sun Microsystems Laboratories, Inc. Technical Report SMLI TR 93-12* (May 1993).

[7] Cmelik, R. "SpixTools—Introduction and User's Manual." *Sun Microsystems Laboratories, Inc. Technical Report SMLI TR 93-6* (February 1993).

[8] SPARC International, Inc. *The SPARC Architecture Manual*. version 8. New Jersey: Prentice Hall, 1992.

© Copyright 1993 Sun Microsystems, Inc. The SMLI Technical Report Series is published by Sun Microsystems Laboratories, Inc.
Printed in U.S.A.

Unlimited copying without fee is permitted provided that the copies are not made nor distributed for direct commercial advantage, and credit to the source is given. Otherwise, no part of this work covered by copyright hereon may be reproduced in any form or by any means graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

TRADEMARKS

Sun, Sun Microsystems, and the Sun logo are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX and OPEN LOOK are registered trademarks of UNIX System Laboratories, Inc. All SPARC trademarks, including the SCD Compliant Logo, are trademarks or registered trademarks of SPARC International, Inc. SPARCstation, SPARCserver, SPARCEngine, SPARCworks, and SPARCcompiler are licensed exclusively to Sun Microsystems, Inc. All other product names mentioned herein are the trademarks of their respective owners.