

Toward Reducing Processor Simulation Time via Dynamic Reduction of Microarchitecture Complexity*

Jeanine Cook
Department of Electrical and
Computer Engineering
New Mexico State University
jcook@nmsu.edu

Richard L. Oliver
Department of Accounting and
Business Computer Systems
New Mexico State University
roliver@nmsu.edu

Eric E. Johnson
Department of Electrical and
Computer Engineering
New Mexico State University
ejohnson@nmsu.edu

ABSTRACT

As processor microarchitectures continue to increase in complexity, so does the time required to explore the design space. Performing cycle-accurate, detailed timing simulation of a realistic workload on a proposed processor microarchitecture often incurs a prohibitively large time cost. We propose a method to reduce the time cost of simulation by dynamically varying the complexity of the processor model throughout the simulation. In this paper, we give first evidence of the feasibility of this approach. We demonstrate that there are significant amounts of time during a simulation where a reduced processor model accurately tracks important behavior of a full model, and that by simulating the reduced model during these times the total simulation time can be reduced. Finally, we discuss metrics for detecting areas where the two processor models track each other, which is crucial for dynamically deciding when to use a reduced rather than a full model.

1. INTRODUCTION

Cycle-accurate simulation of current generation processor microarchitectures is characterized by a time dilation on the order of 10^2 – 10^4 . A large amount of research has been directed toward reducing this dilation while maintaining simulation accuracy [5, 3]. However, this dilation will continue to grow as the complexity of future generation microarchitectures and software application sizes continues to increase. More importantly, it forces researchers and designers to use faster but less accurate methods of simulation.

We propose to reduce the simulation time of processor microarchitectures by dynamically varying (reducing and subsequently increasing, when appropriate) the complexity of the simulated processor model throughout the simulation. The underlying idea of our method is that during certain execution intervals of a simulated workload, the “answer” produced by simulating the full processor microarchitecture is equivalent to that produced by the faster simulation of a reduced microarchitecture. Therefore, during these intervals we can simulate a reduced processor model at increased

Memory Configuration 1	
L1 D-cache	Direct-mapped, 16KB, 32 byte lines
L1 I-cache	2-way set associative, 16KB, 32 byte lines
L2 Cache	Direct-mapped, 512KB, 64 byte lines
Memory Configuration 2	
L1 D-cache	8-way set associative, 32KB, 32 byte lines
L1 I-cache	8-way set associative, 32KB, 32 byte lines
L2 Cache	Direct-mapped, 1MB, 128 byte lines

Table 1: Simulated Memory Configurations

speed with an insignificant loss in accuracy.

We define the “answer” as the level one data cache miss rate. If the per interval L1 D-cache miss rates produced by the full and reduced processor model are within 5% of one another, we call them equivalent. We specify per interval L1 D-cache miss rates to clarify that we are not interested in cumulative miss rate values, but rather are interested in miss rate behavior within individual execution intervals containing 100,000 instructions each. We demonstrate that this equivalence of results is observable for a particular full and reduced microarchitecture model over a representative class of workloads and set of memory hierarchies. Because previous work has shown that it is possible to develop an accurate mathematical model to estimate out-of-order IPC (instructions committed per cycle) during an in-order simulation [2], we chose to first look at L1 D-cache miss rate as the answer of interest.

We begin a simulation with a full processor model. This model is a cycle-accurate, superscalar, out-of-order, speculative execution processor model that also models the behavior and timing of the cache hierarchy. Then during simulation we employ a metric or *predictor* that indicates when a full model simulation is not required, thus effecting a switch to a reduced-model simulation. Similarly, we use a *predictor* to switch from a reduced-model back to a full-model simulation. The reduced model simulates a scalar, in-order functional execution processor that includes a model of only the behavior of the cache hierarchy.

2. EXPERIMENTAL METHODOLOGY

To investigate the feasibility of this approach, we simulate the two processor models using a particular workload as input. From the results of the two simulations, we identify intervals where L1 D-cache miss rates are equivalent. Using this information we compute speedup, which determines whether or not our method affords any improvement for that particular workload. For workloads that

*

Memory Configuration 1			Memory Configuration 2	
Workload	Potential Speedup	Maximum Potential Speedup	Potential Speedup	Maximum Potential Speedup
INT: go	1.52	4.51	1.28	4.55
li	1.99	4.19	2.51	4.13
perl	2.18	4.19	3.64	4.11
vortex	1.40	3.88	1.75	3.71
FP: su2cor	1.0	2.7	1.02	3.63
sweep3d	1.01	3.39	1.0	2.53
swim	2.24	3.57	2.31	3.78

Table 2: Potential Speedups

demonstrate speedup, the simulation data is processed appropriately and input to a tool called *C5.0* [4]. *C5.0* extracts informative patterns from data. From these patterns, it constructs a classifier that predicts to which class the data belongs. Our data is comprised of performance metrics that are output by the simulator. A single set of metrics belong to one of two classes: reduced or full. *C5.0* expresses the classifier as a decision tree or as a set of rules. This set of rules is analyzed to identify possible predictors. This process is repeated for a set of workloads, memory configurations, and microarchitectures (when applicable).

We use six workloads from the SPEC95 benchmark suite as input to a simulator. All benchmarks are executed to completion and all SPEC benchmarks use the reference input. We also use a Los Alamos National Laboratory benchmark, called Sweep3D. It is a compute-intensive application that solves a 3D Cartesian geometry neutron transport problem.

The SimpleScalar [1] tool set is our simulation platform. We use a modified version of *sim-outorder* as our full, out-of-order, speculative, cycle-accurate processor model and a modified version of *sim-cache* as our reduced, in-order model. We name these models FullPipeLine and SimFunct, respectively. The FullPipeLine processor is four-way superscalar (decode, issue, and commit width of four), with an 8-entry load/store queue and a 16-entry register update unit. It has four integer and four floating-point ALUs, and one integer and one floating-point multiplier/divider. The memory configurations simulated by both FullPipeLine and SimFunct are shown in Table 1.

3. RESULTS

Table 2 shows potential speedups using our *adaptive-model* method. We refer to the speedup as potential because we have not yet implemented our technique into a traditional simulator. Potential speedup does not include the overhead cost. Speedup varies widely, with integer workloads characterized by larger speedups than those shown by floating-point applications. *Swim* is the only floating-point workload that shows any speedup. These results and those of experimentation with very large microarchitectures (not presented here) give evidence that the lack of speedup shown by floating-point workloads may be due to the high utilization rates of several of the microarchitecture components. In contrast, the utilization rates of the same components during simulated execution of the integer workloads are generally significantly lower. The *Maximum Potential Speedup* shown in Table 2 is the maximum possible speedup given

Benchmark	Full Model \implies Reduced Model
go	IPB \leq 6
li	LSQ occupancy \leq 70%
perl	IFQ percent full $>$ 60
vortex	IFQ percent full $>$ 77
swim	IPB $>$ 50

Table 3: Predictors of Model Complexity

the reduced model we have chosen. This number represents the speedup if the given workload could be accurately simulated with the reduced model during its entire execution.

Table 3 shows some of the predictors for dynamically detecting when to switch to from a full-model to a reduced-model simulation. Although not presented here, we have a similar set of predictors for switching from a reduced-model to a full-model simulation. These primarily consist of metrics such as the number of loads and stores executed and distance between loads and stores, respectively. Because hundreds, even thousands, of rules may be generated by *C5.0*, analysis is required to reduce the rule set to a small number that provides the largest classification accuracy and generalization across memory configurations. We are currently assessing the accuracy of these predictors as dynamic detectors of the required model complexity within an adaptive-model virtual simulator. This simulator concatenates previously generated simulation results based on the chosen predictors. We can test for accuracy of the predictors and simulation results, but we can not account for overhead in this simulator.

4. CONCLUSION

We have shown that for some workloads, simulating a simple, in-order processor model produces L1 D-cache miss rates within 5% of those produced by a full, out-of-order execution processor model. These workloads demonstrate effectively equivalent behavior during enough of the simulated execution to result in overall potential speedups ranging from approximately 1.5–4.

General predictors are necessary to employ this adaptive-model methodology to the initial simulation of new workloads. Although currently a single predictor is specific to a particular workload, we are confident that in the future, by refining our current technique and/or exploring new methods, we will discover more general predictors. Once we have identified accurate, general predictors, we plan to implement our adaptive-model technique into a new simulator within the SimpleScalar framework.

5. REFERENCES

- [1] Doug Burger and Todd M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, University of Wisconsin Computer Sciences Department, 1997.
- [2] Gabriel Loh. A Time-Stamping Algorithm for Efficient Performance Estimation of Superscalar Processors. In *Joint International Conference on Measurement and Modeling of Computer Systems*, pages 72–81, June 2001.
- [3] Mark Oskin, Frederic T. Chong, and Matthew Farrens. HLS: Combining Statistical and Symbolic Simulation to Guide Microprocessor Designs. In *The 27th International Symposium on Computer Architecture*, pages 71–82, 2000.
- [4] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.
- [5] Eric Schnarr and James R. Larus. Fast Out-Of-Order Processor Simulation Using Memoization. In *ASPLOS VIII*, pages 283–294, October 1998.