Recent Extensions to the SimpleScalar Tool Suite

Doug Burger [1] Todd M. Austin [1] Stephen W. Keckler [2]

Abstract

Over the past eight years, the SimpleScalar Tool suite has become the most widely used set of simulation tools in the computer architecture research community. The authors have recently completed an NSF-funded project to extend and improve the SimpleScalar tools. In this paper, we describe the extensions and improvements to the tools, which include the capability to simulate more instruction sets, graphical support for performance viewing, and more simulators that model different types of machines, including embedded systems, ISA-specific systems, systems with operating systems, and multiprocessing systems.

1 Introduction

The SimpleScalar Tools [3] are currently one of the most widely used tool sets in the computer architecture research community. First released in 1996, their use has expanded widely, benefiting from the open-source release of the tools [1]. The tools are used for approximately half of the papers in current-day architecture conferences (for example, SimpleScalar was used in 58% of the papers appearing in the 29th International Symposium on Computer Architecture) and have been used to publish well over 500 papers during the past four years. The tools are free to academic researchers.

SimpleScalar was never intended as a complete, "out of the box" software package to be used as-is by researchers. Rather, it is a simulation *environment*, with many features useful for constructing simulators. These features include a simulated target debugger, I/O checkpoints for repeatable simulation from any point in a simulated program's execution, a pipeline viewer, configuration files, toolchains ported to the PISA architecture [1], a statistics registration and handling package, and a command-line argument handling package.

Undoubtedly, one of the most popular features of the tools package is the set of architectural simulators that accompany the tools, which researchers can use and modify to simulate new ideas, so long as those ideas are compatible with at least one of the simulators. The simulators model targets from a very abstract level to a fairly detailed timing level.

[1]Dept. of EECS, University of Michigan.

[2]Dept. of Computer Sciences, The University of Texas at Austin.

[1]PISA stands for Portable Instruction Set Architecture. It closely resembles MIPS with a few additions.

The most abstract of the simulators are (*sim-fast* and *sim-safe*, which are both simply functional architectural simulators. They execute the semantics of an instruction set, and can thus simulate binaries, but have no notion of simulated time, and do not model any underlying hardware. In a sense, they are instruction-set emulators. The difference between the two is that *sim-safe* performs various safety checks (e.g., alignment on loads) at the cost of some speed, whereas *sim-fast* dispenses with the checks to run as fast as possible.

The next more complex set of simulators gather microarchitectural statistics, but still do not model time *per se*. The two most widely used are *sim-cache* and *sim-bpred*, which use the instruction-set emulators to generate an on-the-fly instruction trace, which is then fed into cache and branch prediction models, respectively. A common misconception about the SimpleScalar simulators is that they are execution-driven, when in fact all of them are dynamic trace driven. They use the instruction-set emulators to generate a trace, which is then fed into various microarchitectural modelers (ranging from simple cache models all the way up to full-blown timing models of an entire processor). No traces written to disk or read from disk, as they are always generated on the fly for each simulation.

Finally, the *sim-outorder* simulator provides a detailed timing model of an out-of-order microprocessor, based on Sohi's Register Update Unit, or RUU [8]. Due to the difficulty of building out-of-order processor simulators, many researchers modify this simulator to evaluate their ideas. This simulator is good for uniprocessor simulations, but is less ideal for multiprocessor simulations, which must be truly execution driven to order the communication operations correctly. The issue width of *sim-outorder* can be adjusted, as can the issue restrictions (i.e., in-order vs. out-of-order issue). Thus, this single simulator can approximate a range of systems, from a single-issue in-order scalar core to an aggressive out-of-order 8-wide superscalar core.

While these tools have been beneficial to the community, there were many possible extensions that would benefit researchers but that the maintainers of the tools did not have the resources to add. In 1999, the National Science Foundation funded a grant to the authors of this paper to extend the utility, functionality, and portability of the SimpleScalar tools. The grant came from NSF's CADRE program (CISE Advanced and Distributed Resources for Experimentation), and it ran from 1999-2003. The grant was split across the authors' groups at the University of Michigan and The University of Texas at Austin. The goal was to make a number of additions to the tools that would make them more useful to the research community. Below, we describe the major additions to the tools that were funded under this program.

2 Instruction Set Extensions

SimpleScalar version 3.0 supported two instruction-set front ends. The first is named PISA (for Portable Instruction Set Architecture). It is a close derivative of MIPS, with a few addressing modes removed, and extended to 64 bits to free up space for instruction additions. Sim-outorder can model the instructions as consuming 32 or 64 bits in memory, to approximate how a conventional RISC ISA would behave while not sacrificing the flexibility in the larger instructions.

The second supported ISA is the Alpha AXP instruction set. The advantage to using Alpha is that optimized binaries can be generated via Alpha compilers, and the performance of simulated Alpha binaries on modified versions of sim-outorder can be compared to actual performance of the same binaries running on an Alpha workstation. Unfortunately, since future processors implementing the Alpha instruction set have been discontinued, this capability has been decreasing in importance. For the investment people have made in porting and modifying the tool set to be long-lived, it is important for the tools to emulate a subset of currently active instruction sets. Thus, as a part of this work, we added two major and active ISAs to the SimpleScalar framework:

2.1 The PowerPC ISA

One of the major remaining instruction sets, the PowerPC ISA is used in many products, from embedded systems to Apple desktop computers. We have implemented the complete PowerPC user-level instruction set and released it to external researchers [7]. Like the other instruction sets, it consists of a definition file that implements the ISA semantics, tools for unpacking and loading PowerPC binaries, as well as interfaces to the functional and timing simulators. Since the floating-point support (with the condition registers) is expensive to simulate, we provided for both native, direct execution when simulating PowerPC binaries on a PowerPC host, as well as slower simulation when simulating PowerPC binaries on a different type of architecture.

2.2 The ARM ISA

ARM is extensively used in low-end, embedded systems. We have added functional simulation for the ARM 7 instruction set architecture, as well as the Floating Point Accelerator (FPA) extensions. In addition to the ARM ISA emulation, we have added some timing simulator support to accompany this ISA. This support includes a validated SA-1 pipeline core model, which permits researchers to simulate many popular embedded system targets, such at Intel's StrongARM SA-1110 processor. We have also included a prototype of Intel's Xscale ARM processor in the distribution.

3 Graphical Viewers

Given the quantity of in-flight state in modern processors, it is difficult to reason about the sources of bottlenecks in the microarchitecture. This difficulty is most pronounced in an educational setting, when trying to explain the concept of overlapped operations. Consequently, we developed two separate visualization engines that address this challenge at different levels.

3.1 ss-viz

ss-viz is a high-level microarchitectural pipeline visualizer, and is available with documentation at *www.cs.utexas.edu/users/cart/code/ss-viz.tgz*. ss-viz allows the user to view on-line architectural and microarchitectural state as the program runs, as well as dynamically generated graphs that measure quantities like instructions per cycle, cache miss rates, etc., as a function of time. We have used this tool in a junior-level undergraduate architecture class, to further their understanding of the dynamic flow of microarchitectural state.

3.2 Graphical Pipeline Viewer

The other graphical viewer that we developed is GPV, the Graphical Pipeline Viewer for SimpleScalar [9]. GPV permits users to visualize the performance of programs on arbitrary pipeline configurations. The tool is based on the Perl/TK graphical programming language, permitting the tools to run on most popular platforms. GPV can display instruction pipelining, resource usage, and memory access patterns. Researchers at University of Michigan have successfully used GPV to optimize cryptographic kernels for the Alpha 21264 microarchitecture, and researchers elsewhere have incorporated GPV into their computer architecture courses.

4 New Simulators

The simulator development features (such as statistic database and I/O trace checkpointing), the instruction set front ends, and the graphical visualizers are all "support" tools that facilitate custom development of new simulators. However, since developing complete timing simulators is prohibitively expensive for many researchers, and the timing simulators see such wide use, it is important to keep them up to date. Consequently, we have augmented the old simulators and provided new simulators and simulation environments, described below.

4.1 MASE

The first major simulation extension to the tools that we describe is MASE, the MicroArchitectural Simulation Engine [6]. MASE is a novel performance modeling infrastructure for SimpleScalar that addresses a number of deficiencies in the current detailed performance modeling simulator (*sim-outorder*). The new modeling infrastructure permits arbitrary mispeculation/recovery suitable for analysis of novel speculation mechanisms. In addition, the new simulation infrastructure accurately models microarchitectural operation through RTL-level (micro)functional simulation. This support permits more accurate modeling of mispeculation and data-dependent optimizations. In addition, this methodology permits accurate modeling of multiprocessor race conditions. Finally, the new modeling infrastructure incorporates an execution checker component that simplifies the validation and debugging of complex microarchitectural mechanisms. This new environment is already being used successfully by academic researchers.

4.2 Sim-Alpha

One of the advantages of performance simulators such as *sim-outorder* is that they model the microarchitecture at a level low enough to permit exploration of new microarchitectural innovations, but at a level high enough that makes it relatively easy for researchers to extend the simulators. The simulations are also relatively fast (tens to hundreds of thousands of instructions per second), compared to detailed chip-level, RTL, or circuit simulations. However, modeling at this level does introduce some error compared to a simulator that precisely models a chip.

For those researchers that wanted a more detailed (but more difficult to extend) simulator, we built a simulator (creatively called 'sim-alpha') modeled on an Alpha 21264 processor running in a real workstation (a Compaq DS-10L) [5]. Because it is modeled on an actual chip, it is possible to quantify the error the simulator incurs on benchmarks compared to those workloads running on an actual machine.

The validation against a real Alpha workstation (a DS-10L) showed that, for a large suite of cache-resident microbenchmarks, the simulator was within an average of 2% of the actual alpha workstation. For SPEC2000 benchmarks, the error was larger due to memory and TLB effects: an average of 6.6% on the SPECINT2000 benchmarks and 21.5% on the SPECFP2000 benchmarks [4]. While the error is quantifiable, the simulator has many more features specific to the 21264 microarchitecture that make it less easily extensible for new ideas or modeling. However, implementing new ideas in both simulators can permit researchers to see if performance benefits from new ideas track across two quite distinct microarchitectures that support the same ISA (using the version of sim-outorder with the Alpha ISA). The code is available at *www.cs.utexas.edu/users/cart/code/alphasim-1.0.tgz*.

4.3 Memory Extensions

The original release of the SimpleScalar timing simulators contained simple cache and TLB models that gave good estimates of the effects of cache and TLB miss latencies. However, the memory hierarchies that could be simulated were fairly rigid, and they did not model much of the lower-level, underlying detail and complexity found in modern memory systems.

To compensate, we developed a set of memory extensions, that model both existing structures in more detail as well as adding new structures not modeled in the original release [2]. The structures modeled in more detail include the DRAM subsystem, which includes DRDRAM and SDRAM models, as well as more detailed simulation of bus traffic and contention. The code also supports an arbitrary topology of caches, buses, and memories to be constructed from command-line arguments.

The new structures include finite miss status holding registers (MSHRs), permitting the user to specify how many misses may be overlapped at each level of the memory hierarchy. In addition, these are combining MSHRs; the number of combined requests for a single cache line is also user-definable. The extensions also implement a simple multi-level page table, entries for which can be cached. The user can specify whether caches are virtually or physically indexed or tagged, and can have TLB accesses go in parallel with cache accesses if necessary. The code supports hardware traversal of the page table upon a TLB miss or page fault.

4.4 Operating System and Multiprocessor Simulation

Many benchmark suites such as SPECCPU2000 spend little time in the operating system, consequently restricting simulation to user-level only (as SimpleScalar does) adds little error. However, to simulate more OS-intensive workloads, such as commercial transaction processing, web serving, database accesses, and many other multiprocessor workloads, full-system simulation is necessary to obtain accurate results.

To provide full-system simulation capability using SimpleScalar, we have merged an older version of the IBM Austin Research Lab's SimOS full-system simulator with the PowerPC port of SimpleScalar. This version of SimOS predates the Mambo simulator described later in this special issue. This merged tool enable users to simulate a full-blown operating system (AIX version 4.1.3) running on SimpleScalar timing simulators. We also adapted and merged the SimpleMP tool, a shared-memory multiprocessor version of SimpleScalar originally built by Ravi Rajwar at Wisconsin, with SimOS. This tool permits simulation of full-blown small-scale multiprocessors (in particular, chip multiprocessors), including both explicitly parallel (e.g. MPI) and multithreaded programs, running on an operating system, simulating I/O, etc. Currently the tool runs only on AIX/PowerPC and Linux/x86 systems.

5 Summary

SimpleScalar has proven to be a boon for many researchers, particularly those that do not have the resources or staffpower to develop complex simulation infrastructures internally. In addition to research, many educators are now using it in coursework.

It is our hope that the extensions and improvements described in this paper will be useful to the community. The relentless pace of technological change makes research tools difficult to keep current. However, combining federally-funded efforts with a collaborative, open source model and an easily extensible environment makes it possible for many researchers to perform widely varying studies from a single code base.

Acknowledgments

First, we gratefully acknowledge the National Science Foundation's support through grant number EIA-9975286. Without that support, these extensions and improvements to SimpleScalar would not have been possible. Also, much of the hard development work was done by our graduate students, many of them listed in the technical references, whose excellent efforts we happily acknowledge. Finally, we thank the SimpleScalar user base, whose productive use of the tools has made our efforts worthwhile.

References

[1] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2), February 2002.

[2] D. Burger, A. Kägi, and M. Hrishikesh. Memory hierarchy extensions to the simplescalar tool set. Technical Report TR-99-25, Department of Computer Sciences, University of Texas at Austin, Austin, TX, December 1999.

[3] D. Burger and T. Austin. The simplescalar tool set version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin, Madison, WI, June 1997.

[4] R. Desikan, D. Burger, and S. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 266–277, July 2001.

[5] R. Desikan, D. Burger, S. Keckler, and T. Austin. Sim-alpha: a validated, execution-driven alpha 21264 simulator. Technical Report TR-00-04, Department of Computer Sciences, University of Texas at Austin, Austin, TX, February 2000.

[6] E. Larson, S. Chatterjee, and T. Austin. The mase microarchitecture simulation environment. In *Proceedings of the 2001 International Symposium on Performance Analysis of Systems and Software*, June 2001.

[7] K. Sankaralingam, R. Nagarajan, S. Keckler, and D. Burger. Simplescalar simulation of the powerpc instruction set architecture. Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin, Austin, TX, February 2001.

[8] G. S. Sohi. Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. *IEEE Trans. Comput.*, 39(3):349–359, March 1990.

[9] C. Weaver, K. C. Barr, E. D. Marsman, D. Ernst, and T. Austin. Performance analysis using pipeline visualization. In *Proceedings of the 2001 International Symposium on Performance Analysis of Systems and Software*, June 2001.