To appear in the IEEE TCCA Newsletter (October 1997). (An earlier version appeared in WCAE-3, February 1997.)

RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors

Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve
Department of Electrical and Computer Engineering
Rice University
Houston, Texas
`http://www-ece.rice.edu/~rsim`

Abstract

This paper describes RSIM – the Rice Simulator *for* ILP Multiprocessors *– Version 1.0. RSIM simulates shared-memory multiprocessors (and uniprocessors) built from processors that aggressively exploit instruction-level parallelism (ILP). RSIM is execution-driven and models state-of-the-art ILP processors, an aggressive memory system, and a multiprocessor coherence protocol and interconnect, including contention at all resources. Although originally designed as a research tool, RSIM is also being used successfully in both undergraduate and graduate computer architecture courses at Rice University. RSIM version 1.0 is publicly available.*

1 Introduction

This paper describes RSIM – the **R**ice **S**imulator for **I**LP **M**ultiprocessors – Version 1.0. RSIM is primarily designed to study shared-memory multiprocessor architectures built from processors that aggressively exploit instruction-level parallelism (ILP). It models state-of-the-art ILP processors, an aggressive memory system, and a multiprocessor coherence protocol and interconnect. It is execution-driven (vs. trace-driven) and models contention at all resources. RSIM provides the user with a number of configuration parameters to simulate a variety of shared-memory multiprocessor and uniprocessor configurations. RSIM, along with a detailed RSIM Reference Manual, is available from `http://www-ece.rice.edu/~rsim/dist.html`.

Compared to other shared-memory simulators publicly available at this time, the key advantage of RSIM is that it supports a processor model that is more representative of current and near-future processors. Current publicly available shared-memory simulators assume a much simpler processor model, which can result in significant inaccuracies when used to study shared-memory multiprocessors built from state-of-the-art ILP processors [12]. A cost of the increased accuracy of RSIM, however, is that it is slower than simulators that do not include a detailed processor model.

RSIM was originally designed for computer architecture research, but has also been used successfully at Rice in undergraduate and graduate courses covering both uniprocessor and multiprocessor architectures.

We next describe the architecture features supported by RSIM, RSIM internals (including supported platforms), the RSIM applications interface, statistics produced by RSIM, our experience with RSIM, related work, and future work.

2 Architecture Features

2.1 The Processor Microarchitecture

RSIM models an aggressive ILP processor, incorporating features from a variety of current processors. Key features include:

- Superscalar – multiple instruction issue per cycle
- Out-of-order (dynamic) scheduling
- Register renaming
- Static and dynamic branch prediction
- Non-blocking memory load and store operations
- Speculative issue of loads before address disambiguation of previous stores
- Support for multiple memory consistency models and various implementations of these models [13]
- Software-controlled non-binding prefetching

*The development of RSIM was funded in part by the National Science Foundation under Grant No. CCR-9410457, CCR-9502500, CDA-9502791, and CDA-9617383, the Texas Advanced Technology Program under Grant No. 003604016, and funds from Rice University. Vijay S. Pai is also supported by a Fannie and John Hertz Foundation Fellowship.

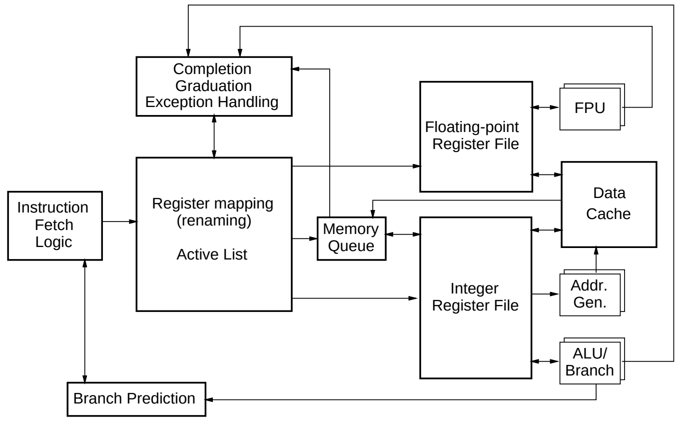

Figure 1: The RSIM processor

The processor microarchitecture modeled by RSIM is closest to the MIPS R10000 [10] and is illustrated in Figure 1. Specifically, RSIM models the R10000's *active list* (which holds the currently active instructions, corresponding to the *reorder buffer* or *instruction window* of other processors), *register map table* (which holds the mapping from the logical to physical registers), and *shadow mappers* (which allow single-cycle state recovery on a mispredicted branch). The pipeline parallels the *Fetch, Decode, Issue, Execute,* and *Complete* stages of the dynamically scheduled R10000 pipeline.[1] Instructions are graduated (i.e., *retired*, *committed*, or *removed*) from the active list after passing through this pipeline. Instructions are fetched, decoded, and graduated in program order; instructions can issue, execute, and complete out-of-order. In-order graduation enables precise interrupts.

The RSIM processor supports static branch prediction, dynamic branch prediction using either a 2-bit history scheme [17] or a 2-bit agree predictor [18], and prediction of return instructions using a return address stack [7]. Each hardware prediction scheme uses only a single level of prediction hardware. The processor may include multiple predicted branches at a time, as long as there is at least one shadow mapper for each outstanding branch. These branches may also be resolved out-of-order.

Most processor parameters are user-configurable, including the number of functional units, the latencies and repeat rates of the functional units, the instruction issue width, the size of the active list, the number of shadow mappers for branch speculation, and the size of the branch prediction structures.

[1] An option for static scheduling is provided with a straightforward modification to the dynamically scheduled pipeline, but is not as thoroughly tested as the dynamic mode.

2.2 The Cache and Memory System

RSIM supports a two-level data cache hierarchy. The first-level cache is multiported and pipelined, and may be writethrough or writeback. The second-level cache is pipelined and writeback. Systems with writethrough first-level caches include a coalescing write buffer. Both caches are lockup-free. They store the state of outstanding requests in miss status holding registers (MSHRs), and coalesce requests to the same line in these MSHRs [8]. Main memory is interleaved and is accessed through a pipelined split-transaction bus.

A variety of user-configurable parameters are provided, including number of ports of the L1 cache, cache line sizes, cache sizes, associativities, number of write buffer entries, number of MSHRs, bus speed, bus width, bus arbitration delay, memory interleaving factor, and latencies of the various modules.

RSIM currently does not support virtual memory and also does not support an instruction cache (assuming a 100% instruction hit rate).

2.3 The Multiprocessor System

RSIM simulates several variations on a base hardware directory-based cache-coherent non-uniform memory access (CC-NUMA) shared-memory multiprocessor. Figure 2 shows the organization of the base system. Each node consists of the processor and cache hierarchy described above, along with a part of the physical memory, its associated directory, and a network interface. A split-transaction bus (mentioned in Section 2.2) connects the secondary cache, the memory and directory module, and the network interface.

RSIM employs a full-mapped invalidation-based directory cache-coherence protocol, and can support either a MESI protocol (with Modified, Exclusive, Shared, and Invalid states) or an MSI protocol (with

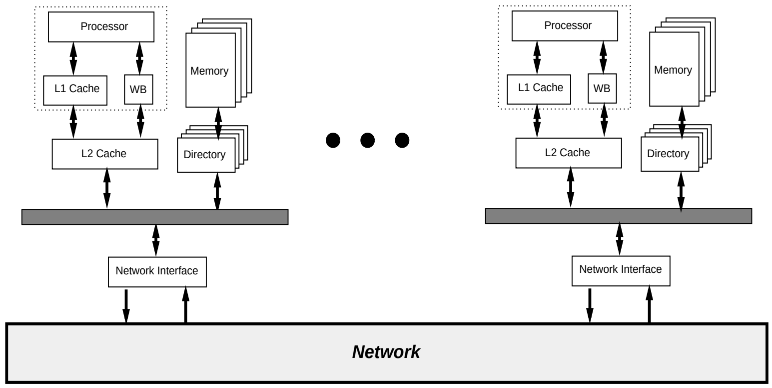

Figure 2: The RSIM multiprocessor system

Modified, Shared, and Invalid states). Both protocols support cache-to-cache transfers for requests for lines held by another processor in Modified state.

For remote communication, RSIM supports a two-dimensional wormhole-routed mesh network. For deadlock avoidance, the system includes separate request and reply networks. The flit delay per network hop, the width of the network, the buffer size at each switch, and the length of each packet's control header are user-configurable parameters.

RSIM supports three memory consistency models – sequential consistency [9], processor consistency [5], and release consistency [5], configurable at compile-time. It also supports optimizations specific to ILP processors for each model, including hardware-controlled prefetching from the instruction window and speculative load execution [4, 13, 16].

3 RSIM Internals

RSIM interprets application executables. The use of application executables rather than traces allows more accurate modeling of the effects of contention and synchronization in simulations of multiprocessors, and more accurate modeling of speculation in simulations of multiprocessors and uniprocessors. We interpret application executables rather than use direct execution because modeling ILP processors accurately with direct execution is currently an open problem.

Internally, RSIM is a discrete event-driven simulator. The event-driven simulation subsystem is based on the YACSIM library from the Rice Parallel Processing Testbed (RPPT) [3, 6]. Other key subsystems include the processor out-of-order engine, the processor memory unit, the cache hierarchy, the memory/directory module, and the interconnection network. Each of these subsystems acts as a largely independent block, interacting with the other units through a small number of predefined mechanisms. Significant parts of the memory and network subsystems are based on code from RPPT [3, 14]. Many of the subsystems within RSIM are activated as separate events only when they have work to perform. However, the processors and caches are simulated using a single event that is scheduled for execution on every cycle, as these units are likely to have activity on nearly every cycle. On every cycle, this event appropriately changes the state of each processor's pipeline and processes outstanding cache requests. The various events faithfully model the processor pipelines, the cache and memory system, and the network, including contention at all resources.

RSIM is written in a modular fashion using C++ and C for extensibility and portability. Currently, it has been tested on Sun systems running Solaris 2.5 or 2.6, a Convex Exemplar running HP-UX version 10, and an SGI Power Challenge running IRIX 6.2. Porting RSIM to the latter two systems from an initial Sun version was straightforward. Porting RSIM to 64-bit platforms or little-endian platforms may require additional effort.

While designing RSIM, we have placed an emphasis on accuracy, a large architecture feature set, code modularity to enable easy addition of new features, and exhaustive statistics collection to better understand performance bottlenecks. This emphasis has often required a sacrifice in simulation speed. Based on our experience with RSIM, we are currently working on various approximations to improve its speed without significantly sacrificing accuracy.

4 Applications Interface

RSIM simulates applications compiled and linked for SPARC V9/Solaris using ordinary SPARC compilers and linkers, with the following exceptions.

First, although RSIM supports most important user-mode SPARC V9 instructions, there are a few unsupported instructions. More specifically, all instructions generated by current C compilers for the UltraSPARC-I or UltraSPARC-II with Solaris 2.5 or 2.6 are supported. Unsupported instructions that may be most important on other SPARC systems include 64-bit integer register instructions and quadruple-precision floating-point instructions.

Second, the system trap convention supported by RSIM differs from that of Solaris. Therefore, standard libraries and functions that rely on such traps cannot be directly used. We provide an RSIM applications library to support such commonly used libraries and functions; all applications must be linked with this library. Nevertheless, there are some unsupported traps and related functions (e.g., `strftime`), and our library has only been tested for C application programs.

For multiprocessor applications, the RSIM applications library includes support for synchronization with locks, flags, and barriers, as well as support for these primitives through PARMACS macros.

For speed and portability, RSIM actually interprets applications in an expanded, loosely encoded instruction set format. A predecoder is provided to convert SPARC application executables into this internal format, which is then fed to the simulator.

5 Statistics in RSIM

RSIM provides a variety of execution statistics. For many metrics, RSIM provides the average value of the metric, the standard deviation, and a histogram showing the distribution of the values of the metric. We also provide scripts that interface with a plotting utility to graphically display statistics related to a run or a set of runs.

Overall performance statistics. RSIM displays the total execution time and the IPC (instructions per cycle) achieved by the program on the system simulated. The total execution time is further categorized into processor busy time and stalls due to various classes of instructions. These classes include ALU, FPU, data reads, data writes, exceptions, branches, synchronization, and up to nine user-defined classes. Data read and write stalls are further split according to the level of the memory hierarchy at which the memory operations were resolved: L1 cache, L2 cache, local memory, or remote memory.

Other processor statistics. RSIM provides statistics on the usage of various functional units in the processor, the branch prediction behavior, and the occupancy of the instruction window. It also displays the metrics of availability, efficiency, and utility [1] related to instruction fetching.

Cache, memory, and network statistics. RSIM classifies memory operations into hits and misses, and further classifies misses into cold, capacity, conflict, and coherence misses. It also collects the average latency of various classes of memory operations, MSHR occupancy, prefetch effectiveness, bus utilization, write-buffer utilization, network contention, traffic, and the usage of the network switch buffers.

6 Experience

We have used RSIM successfully for computer architecture research [12, 13, 15, 16] and education. RSIM is used in two computer architecture courses at Rice – the first is a senior-level course primarily on uniprocessor architecture and the second is a graduate-level course primarily on parallel architecture. RSIM is used for both short assignments and semester-long course projects. The assignments supplement material taught in class. For example, students used RSIM in one assignment to pinpoint performance bottlenecks in ILP processors. Students then designed studies to determine the impact of various processor configuration parameters and evaluated the performance of these modified configurations.

Course projects using RSIM are done in groups of two to four students and typically involve extensions or validations of recent architecture studies in the literature. These projects often require students to implement additional modules to supplement the existing features of RSIM. Course projects in Fall'96 included studies of static vs. dynamic scheduling, aggressive branch prediction strategies for out-of-order processors, tradeoffs between performance and die area, the performance benefits of high-bandwidth DRAM architectures, and use of data stream buffers with out-of-order processors.

7 Related Work

Numerous simulators exist for shared-memory multiprocessor systems, many of which are execution-driven (vs. trace-driven). However, most of these model previous-generation processors with static scheduling and blocking loads. An exception is the SimOS system when used with the MXS processor model [11]. This simulator was developed concurrently with RSIM. Like RSIM, it models advanced processor pipelines, including out-of-order issue and non-blocking loads.

A large number of uniprocessor architecture studies have been based on trace-driven simulation. Execution-driven ILP uniprocessor simulators include the MXS [1] and SimpleScalar simulators [2].

8 Future Work

We are currently engaged in various additions to the features supported by RSIM, including instruction caches, TLBs, aggressive branch prediction strategies, and performance visualization support. We are also working on improving the performance of RSIM by optimizing the current code, parallelization, and using more novel approximations and modeling techniques.

9 Acknowledgments

We thank Hazim Abdel-Shafi, Murthy Durbhakula, Jonathan Hall, and Tracy Harton for assistance in various stages of the development of RSIM. We are also grateful to the Rice Parallel Processing Testbed (RPPT) group; significant parts of the RSIM memory and network system are based on code from RPPT.

References

[1] J. E. Bennett and M. J. Flynn. Performance Factors for Superscalar Processors. Technical Report CSL-TR-95-661, Stanford University, Feb. 1995.

[2] D. Burger, T. Austin, and S. Bennett. Evaluating Future Microprocessors: the SimpleScalar Tool Set. Technical Report CS-TR-96-1308, University of Wisconsin, Madison, July 1996.

[3] R. G. Covington, S. Dwarkadas, J. R. Jump, S. Madala, and J. B. Sinclair. The Efficient Simulation of Parallel Computer Systems. *Intl. Journal of Computer Simulation*, 1:31–58, Jan. 1991.

[4] K. Gharachorloo, A. Gupta, and J. Hennessy. Two Techniques to Enhance the Performance of Memory Consistency Models. In *Proc. Intl. Conf. on Parallel Processing*, 1991.

[5] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proc. 17th Intl. Symp. on Computer Architecture*, 1990.

[6] J. R. Jump. *YACSIM Reference Manual.* Rice University Electrical and Computer Engineering Department, March 1993.

[7] D. R. Kaeli and P. G. Emma. Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns. In *Proc. 18th Intl. Symp. on Computer Architecture*, 1991.

[8] D. Kroft. Lockup-Free Instruction Fetch/ Prefetch Cache Organization. In *Proc. 8th Intl. Symp. on Computer Architecture*, 1981.

[9] L. Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. on Computers*, C-28(9):690–691, Sept. 1979.

[10] MIPS Technologies, Inc. *R10000 Microprocessor User's Manual, Version 2.0*, Dec. 1996.

[11] B. A. Nayfeh, L. Hammond, and K. Olukotun. Evaluation of Design Alternatives for a Multiprocessor Microprocessor. In *Proc. 23rd Intl. Symp. on Computer Architecture*, 1996.

[12] V. S. Pai, P. Ranganathan, and S. V. Adve. The Impact of Instruction Level Parallelism on Multiprocessor Performance and Simulation Methodology. In *Proc. 3rd Intl. Symp. High Performance Computer Architecture*, 1997.

[13] V. S. Pai, P. Ranganathan, S. V. Adve, and T. Harton. An Evaluation of Memory Consistency Models for Shared-Memory Systems with ILP Processors. In *Proc. 7th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1996.

[14] U. Rajagopalan. The Effects of Interconnection Networks on the Performance of Shared-Memory Multiprocessors. Master's thesis, Department of Electrical and Computer Engineering, Rice University, Jan. 1995.

[15] P. Ranganathan, V. S. Pai, H. Abdel-Shafi, and S. V. Adve. The Interaction of Software Prefetching with ILP Processors in Shared-Memory Systems. In *Proc. 24th Intl. Symp. on Computer Architecture*, 1997.

[16] P. Ranganathan, V. S. Pai, and S. V. Adve. Using Speculative Retirement and Larger Instruction Windows to Narrow the Performance Gap between Memory Consistency Models. In *Proc. 9th Symp. on Parallel Algorithms and Architectures*, 1997.

[17] J. E. Smith. A study of branch prediction strategies. In *Proc. 8th Intl. Symp. on Computer Architecture*, 1981.

[18] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt. The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference. In *Proc. 24th Intl. Symp. on Computer Architecture*, 1997.