Available online at www.sciencedirect.com
SCIENCE DIRECT
Journal of Systems Architecture 52 (2006) 283–297
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

The design and utility of the ML-RSIM system simulator

Lambert Schaelicke [a,*], Mike Parker [b]

[a] Intel Corporation, 3400 Harmony Road, Fort Collins, CO 80528, USA
[b] Cray, Inc., PO Box 6000, Chippewa Falls, WI 54729, USA

Received 24 July 2004; accepted 21 July 2005
Available online 25 October 2005

Abstract

Execution-driven simulation has become the primary method for evaluating architectural techniques as it facilitates rapid design space exploration without the cost of building prototype hardware. To date, most simulation systems have either focused on the cycle-accurate modeling of user-level code while ignoring operating system and I/O effects, or have modeled complete systems while abstracting away many cycle-accurate timing details. The ML-RSIM simulation system presented here combines detailed hardware models with the ability to simulate user-level as well as operating system activity, making it particularly suitable for exploring the interaction of applications with the operating system and I/O activity. This paper provides an overview of the design of the simulation infrastructure and discusses its strengths and weaknesses in terms of accuracy, flexibility, and performance. A validation study using LMBench microbenchmarks shows a good correlation for most of the architectural characteristics, while operating system effects show a larger variability. By quantifying the accuracy of the simulation tool in various areas, the validation effort not only helps gauge the validity of simulation results but also allows users to assess the suitability of the tool for a particular purpose.

Keywords: Simulator validation; Execution-driven simulator; Simulator accuracy

1. Introduction

Simulation has become one of the predominant tools for computer architecture research. It allows researchers to rapidly explore novel techniques and architectures without incurring the engineering effort and expense of implementing prototype hardware. In addition, it affords a flexibility normally not provided by real hardware. However, the design and implementation of a suitable and robust simulator is in itself a major undertaking that requires a combination of architectural expertise and software development skills. For this reason, a number of readily available and proven simulation systems such as SimpleScalar [2], SimOS [16], SimICS [12] and RSIM [15] have found widespread use. Understanding the performance, accuracy, and limitations of simulators is critical when selecting a tool to answer a particular research question.

This paper describes the design and validation of ML-RSIM, an execution-driven system simulator

* Corresponding author.
E-mail addresses: l.schaelicke@computer.org (L. Schaelicke), map@cray.com (M. Parker).
URLs: http://www.cs.utah.edu/~lambert (L. Schaelicke), http://www.cs.utah.edu/~map (M. Parker).

1383-7621/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2005.07.001

that combines detailed hardware models of modern workstation and server-class machines with a Unix-like operating system [18]. The ML-RSIM environment, based on RSIM [15], extends the system features normally represented by simulators to include I/O devices and an operating system without sacrificing fidelity of the processor model and with only a small impact on simulation speed. A detailed characterization of the accuracy of the hardware models as well as the simulator operating system highlights key strengths and weaknesses of ML-RSIM and helps determine the tool's suitability for specific research tasks.

Simulator design is a trade-off between development effort, simulation speed, modeling scope, and accuracy. As such, most simulators are targeted towards specific application domains and research agendas. For instance, architectural simulators like SimpleScalar [2] and RSIM [15] focus on the cycle-accurate modeling of user-level instructions by providing highly detailed models of the processor and memory hierarchy. These tools are invaluable for experiments involving compute-intensive applications, but they ignore many system-level effects such as virtual memory, system calls, interrupt handling, and other I/O related effects. System-level simulators like SimOS [16] and SimICS [12], on the other hand, provide complete system models and thus account for application, operating system, and I/O effects. However, to achieve acceptable simulation speed, hardware models are usually simplified and not cycle-accurate. Such system-level simulators are able to model the interaction of hardware, system software, and applications. However, due to the abstract hardware models, only coarse-grain effects can be observed.

In contrast, ML-RSIM combines detailed hardware models of a microprocessor, memory hierarchy, and I/O subsystem with a simulated operating system. As such, it supports detailed explorations of the interaction of applications, operating systems, and I/O devices. Hardware models include a dynamically scheduled processor with a two-level cache hierarchy, coherent system bus, and a multi-bank memory controller. These components are augmented with an I/O subsystem consisting of a PCI bridge, SCSI adapter, SCSI bus, and hard disk. The fully-simulated "Lamix" operating system is based on NetBSD source code and includes a system call layer, multiple file systems with buffer cache, device drivers, and multitasking capabilities [18].

The following section briefly describes the design of ML-RSIM, focusing on the I/O subsystem and operating system infrastructure. Section 3 presents the validation methodology used to gauge the accuracy of ML-RSIM. Section 4 presents the validation results and discusses how important design decisions affect the simulator's utility and accuracy. Section 5 contrasts this work with other simulation infrastructures and validation efforts. Finally, Section 6 summarizes and outlines ongoing and future work.

2. ML-RSIM design

2.1. Machine model

The ML-RSIM processor model is based on RSIM v1.0 [15] with extensions to simulate operating system code and memory-mapped access to I/O devices. Other hardware models include a two-level cache hierarchy, a coherent memory bus, a memory controller, and several I/O devices. Combined, these components model a contemporary workstation or server-class machine, as shown in Fig. 1.

The ML-RSIM CPU model implements the SPARC V8 instruction set [21] on a MIPS R12000-like dynamically scheduled superscalar processor core [14]. It also includes an instruction cache model, TLB models, exception handling capabilities, and support for privileged instructions and registers. Both user-level and privileged instructions are simulated, so simulation sessions account for both application and kernel effects.

The cache hierarchy is modeled as a conventional two-level structure with support for snooping cache coherency. An uncached buffer handles load and store instructions to I/O addresses, and supports store combining to reduce system bus utilization. The system bus implements snooping cache coherency through a MESI protocol. The memory controller supports SDRAM or RAMBUS memory in a variety of configurations and accurately models queueing delays, bank contention, and DRAM precharge and refresh overhead.

The I/O subsystem is based on the PCI bus and consists of a PCI bridge with autoconfiguration support, a real-time clock, and a number of SCSI host adapters with hard disks. The real-time clock is modeled after a MOSTEK 48T02 clock chip and implements common date and time functionality as well as two independent periodic interrupt sources. The SCSI adapter is a variation of an

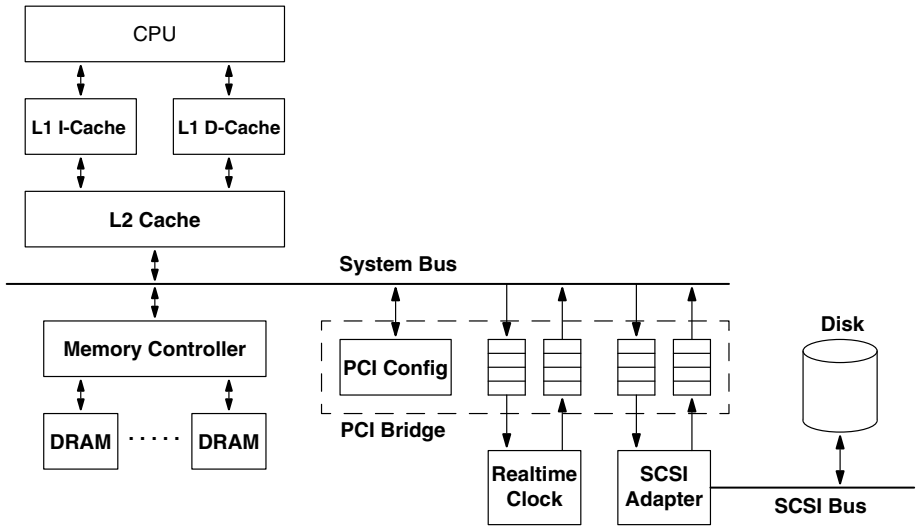


Fig. 1. ML-RSIM system model.

Adaptec AIC 7770-based SCSI host adapter and supports multiple outstanding requests, disconnect and reconnect transactions, and request queueing [1]. Each adapter controls one SCSI bus that accounts for arbitration delays, idle times, and data transfer. Each SCSI bus connects an adapter to a configurable number of SCSI disks. These disks model seek and transfer times as well as an on-board DRAM buffer. Seek times are calculated using one of four algorithms: (a) a fitted curve based on single-track, average, and full-stroke seek times, (b) a three-point linear curve based on the same parameters, (c) a constant seek time, or (d) instantaneous seeks with zero latency [8,11]. The sector-based disk cache is used for data staging, prefetching and optional write buffering. The disk model stores non-empty disk sectors in files on the simulation host, thus maintaining simulated disk contents across successive simulation sessions.

The ML-RSIM hardware models intentionally do not represent any particular system. Instead, various features are modeled after a number of existing prototypes, with the goal of producing a simulator representative of a large class of real machines. The system bus has similarities with SGI systems, while the PCI addressing scheme follows the design of a PowerPC-based workstation. Closely approximating the design of a particular system would have facilitated a relatively straightforward port of an open-source operating system, but would also have imposed many undesirable idiosyncrasies of that system. The approach taken for ML-RSIM balances the need to accurately model a realistic system with the cost of implementing the minute and often obscure details of a particular commercial system.

2.2. Lamix kernel

The Lamix kernel is a Unix-compatible multitasking operating system specifically designed to run on the ML-RSIM simulator. It consists of an independently developed multitasking core and significant portions of NetBSD source code that implement file system and device driver functionality. The system ABI is Solaris-compatible; thus applications compiled for Lamix can, in most cases, execute on native Solaris hosts without modifications. Lamix only requires that applications are statically linked and do not use 64-bit instructions. This flexibility allows users to test and debug applications on a native host at significantly greater speed before simulating them. Since no special libraries or header files are required, applications intended for simulation can be compiled from nearly any programming language with little restriction on the compiler used. However, due to the restriction that applications must be statically linked, arbitrary Solaris binaries may not run on the simulator.

The Lamix kernel supports Unix-style process management, signal handling, basic virtual memory management, file I/O both to simulated disks and to simulation host files, and Unix and Internet domain sockets. All traps and interrupts are dispatched to fully simulated trap handlers. Process management

allows for the dynamic creation and termination of processes, loading of executable files, and passing of signals between processes. Virtual memory management is fully functional, but its implementation is somewhat simplified compared to other Unix variants. Page table management and address translation are accurately modeled, including TLB miss handling via software traps. However, physical memory management does not support paging to disk. Instead it uses a static allocation scheme, resulting in a fixed upper limit on the number of concurrently active processes. In addition, memory-mapped files, and consequently dynamic linking, are currently not supported. These restrictions are partly a result of the incremental design of the Lamix kernel. On the other hand, the static physical memory management scheme helps make results more easily reproducible, since small changes to the order of memory allocation requests from applications can otherwise have a significant impact on overall system behavior due to varying level-2 cache conflicts.

The vnode-based file system layer demultiplexes file I/O system calls to one of three file systems. Both the standard BSD fast file system (FFS) and a log-structured file system (LFS) are fully simulated, including buffer cache and an LFS cleaner daemon [10]. A newly-designed HostFS file system imports the entire file structure of the simulation host into the Lamix environment, thus giving applications access to files in the user's home directory or other locations. While the simulated file systems accurately account for disk access delays, host files are accessed instantaneously. The main purposes of the HostFS are to provide a mechanism to move files between the host and the simulated file system, and to provide file I/O for applications where accurately modeling the I/O overhead is not required, such as accessing configuration files.

The socket layer supports Unix and INET domain sockets. Unix sockets are fully simulated using the BSD mbuf structure, while INET socket operations are performed instantaneously. Consequently, INET sockets are most useful for applications that do not require accurate modeling of networking overheads and perform only a small amount of communication, such as a database server that reads relatively short request messages before performing sizable amounts of disk I/O.

Underlying the file systems are several device drivers, including SCSI disk and host adapter drivers and a PCI bus driver. These device drivers are ported from NetBSD with little modification and accurately account for interrupt handling, synchronization, and request queueing. In addition to these physical devices, Lamix includes the RAIDframe software RAID driver [6] and a pseudo disk device for striping and device concatenation.

2.3. Simulation speed

One of the costs of the level of detail provided by ML-RSIM is simulation speed. Generally, the majority of simulation time is spent executing the processor and cache models, with the instruction decode, register rename and dependency check stages being a significant portion of that time.

The instruction cache model, which is essential for ML-RSIM's support of operating system code, is a considerable source of simulation overhead. Furthermore, changes to the I-cache model parameters have a significant impact on simulation overhead. To improve simulation speed, instructions are decoded into an expanded format while being fetched into the simulated I-cache. Consequently, as the modeled I-cache becomes more effective, simulation speed increases as well. Traditionally, such pre-decoding occurs only once for the entire program at the beginning of a simulation run. However, ML-RSIM requires fully dynamic instruction decoding to support operating system functionality such as the loading of new binaries.

Other hardware components such as the I/O devices, the system bus model, and the memory controller are not significant contributors to the simulation cost, since the event-driven implementation invokes these models only when they are active.

Simulating a detailed operating system does not significantly impact simulation speed beyond the small additional cost of support for exception handling and privileged state. Modeling complete system calls and interrupt handlers requires more simulation time than a simulator that performs instantaneous pseudo-system calls. However, one of the main goals of ML-RSIM is to provide insight into the interactions between user and system code and to allow detailed measurements of operating system costs. Consequently, simulating kernel code is not considered overhead but an important component of the simulation process. The kernel completes initialization and mounts a simulated disk in under 50 million simulated cycles, resulting in a runtime overhead of approximately 1 min on a 900 MHz Sun Blade-1000. Thus, the overhead of

booting Lamix for each simulation run is substantial and does not warrant further optimization.

3. Validation methodology

In our view, simulator validation is the process of quantifying the extent that the simulation tool faithfully models not only the functional behavior but also the performance of real systems. Even though most architectural simulators are used to evaluate new techniques on hypothetical future computer systems, a simulator should be able to approximate the characteristics of existing systems. Such validation helps ensure that conclusions and design trade-offs made with a given simulator are not overly affected by modeling errors. Equally important, validation uncovers the simulator's weaknesses and inaccuracies and lets users judge the suitability of the tool for a particular purpose.

3.1. Methodology

This work uses LMBench [20] to compare various characteristics of a reference system with the corresponding simulated system. LMBench consists of a set of benchmarks that measure microarchitectural performance and basic operating system behavior without relying on hardware performance counters. While such counters are available in nearly all modern systems, their functionality varies considerably and often does not provide enough detail to adequately compare against a simulated system. Portable and well-known microbenchmarks are able to report the same performance metrics for all systems compared here, thus helping to discover the correlation between specific details of the benchmark system and the simulator, such as cache and memory behavior. Such validation establishes confidence that the simulator accurately models the performance of existing systems, and that trends and changes observed on the simulation tool represent real effects and not modeling artifacts.

To this end, two widely different simulator configurations are compared with two representative reference systems: a 175 MHz SGI Octane and a 900 MHz Sun Blade-1000. By quantifying and analyzing the simulators accuracy against such diverse systems, the confidence in the validation process itself is increased, and users of the tool are better able to judge the simulators suitability for a specific purpose. Comparing two diverse systems that differ in instruction set, processor microarchitecture, memory subsystem, and operating system gives further insight into the simulator's strengths and weaknesses in terms of its accuracy and its ability to model a wide range of systems.

3.2. Experimental setup

The validation study presented here is a continuation of work previously reported [17]. It uses the same benchmark system and simulator configuration, augmented with a second set of experiments for a significantly different reference system.

The first reference system is an SGI Octane workstation with one 175 MHz R10000 processor. Although this system no longer represents current high-performance workstation-class machines, it provides a useful comparison point because its architectural characteristics are well documented, and because the R10000 processor's microarchitecture closely matches that modeled by ML-RSIM. Moreover, many architectural aspects of this class of machine have not changed significantly. Combined with the more modern Sun Blade-1000 reference system, this system choice can demonstrate a good correspondence between the simulator and a broad class of real architectures. Table 1 summarizes the key parameters of the reference system and the matching simulator configuration. Other parameters such as the number and latency of various functional units are set to match published data on the MIPS processor.

The second reference system is a 900 MHz Sun Blade-1000 workstation. This machine is based on the superscalar in-order UltraSPARC-III processor. Since its instruction set matches that of the simulation system, and the Lamix system call interface is compatible with Solaris, identical benchmark executables can be used in this validation. Table 2 summarizes the key parameters of both systems. Again, other microarchitectural parameters are set to match published values where appropriate.

Overall, these two reference systems cover a wide range of workstation-class systems, thus providing more meaningful insights through the validation process. While the microarchitecture of the MIPS processor closely matches that modeled by the simulator, the instruction set, operating system and compiler differ, posing some challenges for the validation. On the other hand, the Sun reference system is based on the same instruction set, but the processor microarchitecture differs from the simulator processor model. To approximate the behavior of

Table 1
Architectural parameters of the SGI Octane reference and the corresponding simulator configuration

Parameter	SGI Octane	ML-RSIM
Processor	175 MHz MIPS R10000, MIPS ISA, 4-way superscalar, dynamically scheduled	175 MHz SPARC ISA, 4-way superscalar, dynamically scheduled
L-1 D-Cache	32 kbyte, 2-way set-assoc., 32-byte lines, 1 cycle latency	32 kbyte, 2-way set-assoc., 32-byte lines, 1 cycle latency
L-1 I-Cache	32 kbyte, 2-way set-assoc., 64-byte lines, 1 cycle latency	32 kbyte, 2-way set-assoc., 64-byte lines, 1 cycle latency
L-2 Cache	1 Mbyte, 2-way set-assoc., 128-byte lines, 10 cycles latency	1 Mbyte, 2-way set-assoc., 128-byte lines, 10 cycle latency
System bus	87.5 MHz, 64-bit multiplexed	87.5 MHz, 64-bit multiplexed
Memory controller	SGI proprietary	SDRAM, 4 Banks
Hard disk	IBM UltraStar 9, 9.1 Gbyte 7200 rpm, 10 heads, 8420 cylinders, 1 Mbyte cache with 16 sectors	9.1 Gbyte, 7200 rpm, 10 heads, 8420 cylinders, 1 Mbyte cache with 16 sectors
OS	IRIX 6.5, 64 bit System V, multithreaded	Lamix, 32 bit SysV compatible, BSD-based
Compiler	MIPS Pro 6.2	Sun Workshop 6.0

Table 2
Architectural parameters of the Sun Blade-1000 reference and the corresponding simulator configuration

Parameter	Sun Blade-1000	ML-RSIM
Processor	900 MHz UltraSPARC-III SPARC V9 ISA 4-way superscalar in-order	900 MHz SPARC V8plus ISA 4-way superscalar dynamically scheduled
L-1 D-Cache	64 kbyte, 4-way set-assoc., 32-byte lines, 1 cycle latency	64 kbyte, 4-way set-assoc., 32-byte lines, 1 cycle latency
L-1 I-Cache	32 kbyte, 4-way set-assoc., 32-byte lines, 1 cycle latency	32 kbyte, 4-way set-assoc., 32-byte lines, 1 cycle latency
L-2 Cache	8 Mbyte, 2-way set-assoc., 64-byte lines, 14 cycles latency	8 Mbyte, 2-way set-assoc., 64-byte lines, 14 cycles latency
System bus	150 MHz, 256-bit data path	150 MHz, 256-bit multiplexed address/data
Memory controller	SDRAM, integrated onto CPU die	SDRAM, 2 Banks
Hard disk	Seagate Cheetah, 36 Gbyte, 10,000 rpm, 4 heads, 24,620 cylinders, 4 Mbyte cache with 16 sectors	36 Gbyte, 10,000 rpm, 4 heads, 24,620 cylinders, 4 Mbyte cache with 16 sectors
OS	Solaris 8, 64 bit System V, multithreaded	Lamix 32 bit SysV compatible, BSD-based
Compiler	Sun Workshop 6.0	Sun Workshop 6.0

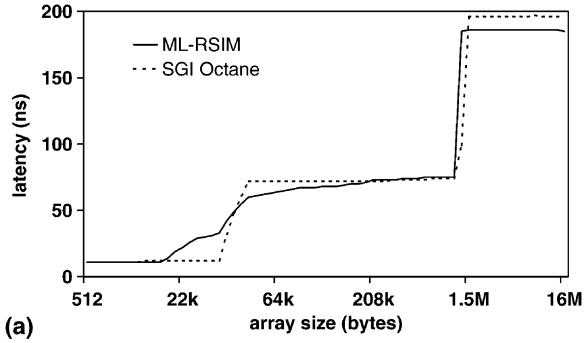
an in-order execution core, the issue window size of the simulated processor is set to 12 entries, with at most two outstanding memory references.

4. Validation results

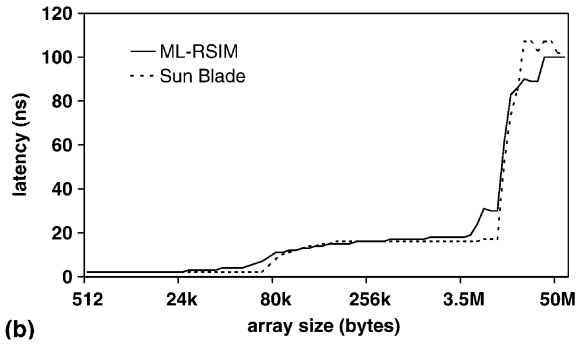
4.1. Memory System Performance

Memory System Performance LMBench measures memory latency by traversing arrays of pointers. Depending on the array size, the observed latency corresponds to different levels of cache or main memory. Fig. 2 shows the resulting graphs for both reference platforms and the simulator models for a 32-byte stride, as well as the squares of the correlation coefficient for all strides.

Generally, the boundaries between the three plateaus correctly identify the level-1 and level-2 cache sizes. Furthermore, the load latencies measured for each memory hierarchy level closely match



Stride	R^2
16k	0.9496
32k	0.9609
64k	0.9232
128k	0.9457
256k	0.9492
512k	0.9446
1M	0.0431



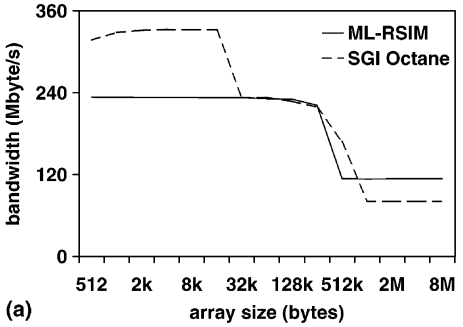
Stride	R^2
16k	0.9461
32k	0.9778
64k	0.9487
128k	0.9840
256k	0.9810
512k	0.9848
1M	0.9837

Fig. 2. Memory load latency for 32-byte stride and r^2 values for all strides: (a) SGI Octane and (b) Sun Blade.

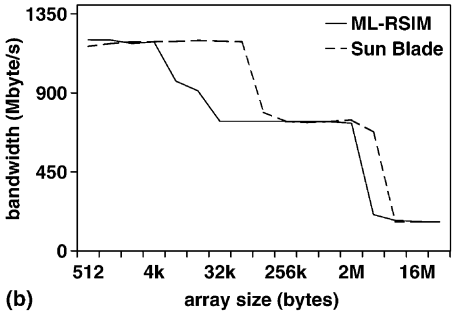
the reference and the simulated systems. The most significant discrepancy between the reference and the simulated systems is generally observed at the transition points between plateaus and is primarily the result of differing interferences of the test array with instructions and TLB entries in the level-2 cache. The r^2 values shown in the tables to the right are also known as the coefficient of determination and are computed as the square of the Pearson correlation coefficient. Generally, r^2 values range from zero to one and indicate the strength of the correlation between the two systems. A value of one represents a perfect correlation, and a value of zero represents no correlation. As the results demonstrate, both ML-RSIM configurations are able to closely track the memory latency of the reference system for all experiments.

In addition to memory latency, LMBench measures memory bandwidth by accessing an array of varying size in a variety of patterns. Similar to the latency measurements, the resulting curves represent bandwidth at different levels of the memory hierarchy. Measurements are taken using the bcopy and bzero routines provided in the native C-library as well as using unrolled copy, read, and write loops. In addition, copy bandwidth is measured for conflicting and non-conflicting source and destination arrays. Fig. 3 shows the bandwidth graphs for the C-library bcopy routine, as well as the r^2 values for all bandwidth measurements.

On the reference platforms, the copy bandwidth shows three distinct regions that correspond to the cache and memory hierarchy levels. For the SGI Octane configuration, ML-RSIM fails to reproduce the steep bandwidth drop at the L-1 cache boundary, but subsequently follows the reference platform more closely. For the Sun Blade configuration, the simulated system tracks bandwidth more closely for small array sizes, but starts observing lower bandwidth at array sizes below the level-1 cache size. Overall, the correlation between the reference system and the simulator is weaker than it is for the latency experiments, especially for tests that include a significant fraction of writes. Part of this discrepancy is the result of differences in the cache controller implementation that affect when and how multiple outstanding requests are coalesced and how writebacks are buffered. It is also worth noting that the implementation of bcopy used on the SGI Octane differs significantly from that used on the Sun Blade and on the simulator. This difference is largely due to the different instruction sets



Method	R^2
libc bcopy unaligned	0.8176
libc bcopy aligned	0.6663
copy unrolled	0.8866
copy unrolled partial	0.7761
read	0.9048
read partial	0.9469
write	0.7667
write partial	0.7154
read/write partial	0.6799
bzero	0.7875



Method	R^2
libc bcopy unaligned	0.7715
libc bcopy aligned	0.7732
copy unrolled	0.8001
copy unrolled partial	0.7827
read	0.7319
read partial	0.6735
write	0.7780
write partial	0.4943
read/write partial	0.4874
bzero	0.7544

Fig. 3. Memory bandwidth of libc bcopy and r^2 values for all bandwidth experiments: (a) SGI Octane and (b) Sun Blade.

that result in different optimization points for each system.

The reference UltraSPARC system uses a 64 kbyte write-through cache accompanied by a 2 kbyte write cache. Earlier experiments suggested that this configuration is best approximated by a 64 kbyte write-back cache rather than by the write-through cache model provided by ML-RSIM. However, significant differences remain, resulting in the bandwidth difference between the two systems for array sizes close to the level-1 cache size. In many cases such discrepancy at the boundary points is not critical. However, care must be taken that applications do not continually operate at these boundary points, or else small perturbations will result in large but meaningless differences in measured application performance. This same effect can be observed on real systems as well as other simulators.

4.2. Disk performance

Complementing the memory system measurements, LMBench characterizes disk seek latency and bandwidth. Seek latency is measured by reading from specific sectors on the raw disk device and hence includes rotational delay. Depending on the initial rotational position of the disk, seek latency varies considerably between seeks of similar distance. The resulting graphs are scatter plots with a large variation between adjacent data points. To improve readability, Fig. 4 shows seek latency for the two systems summarized by polynomial trend lines.

In both cases, ML-RSIM is able to approximate the seek latencies of the reference disk through a fitted curve based on published single cylinder, average, and full-stroke seek times. Generally, the simulated disk model overestimates seek latency for small distances, perhaps because there is insufficient detail in modeling SCSI bus and controller overhead. Note that in both cases, the disk model is parameterized from published seek latencies and it not tuned to match the reference disk.

The disk bandwidth experiments reveal a shortcoming of the simulated disk models. The reference disks are divided into multiple zones of varying sector densities to maximize capacity. Consequently, read bandwidth varies from nearly 15 Mbyte/s on the outside of the platter to 7.5 Mbyte/s for the innermost cylinders for the IBM UltraStar drive found in the SGI Octane, and varies between 48

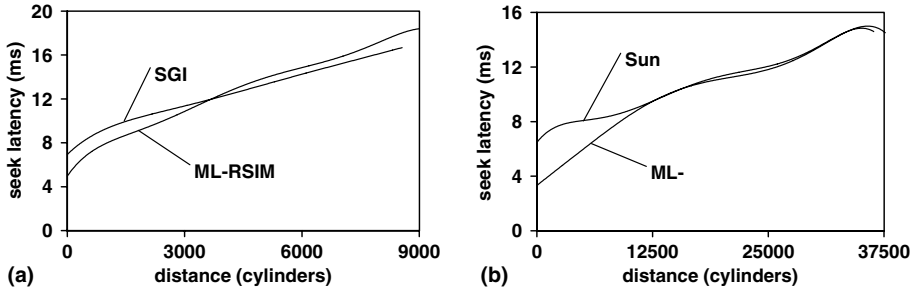


Fig. 4. Hard disk seek latency: (a) SGI Octane and (b) Sun Blade.

and 28 Mbyte/s for the Seagate Cheetah drive in the Sun Blade. ML-RSIM currently does not model multiple zones, but rather assumes a fixed sector density for all cylinders. As a result, disk bandwidth is constant at 10 and 35 Mbyte/s respectively, independent of the cylinder. These bandwidths closely match the average bandwidth of both reference disks. Unfortunately, detailed zoning information is often not available for commercial disks, though an approximation could be formulated through experimentation. Such an approximation would lead to a more accurate disk model, in regard to bandwidth. However, it would place additional burden on the user to correctly parameterize the model. Furthermore, adding zoning details to the disk model makes experimentation results sensitive to the placement of files on the simulated disks. Files on an otherwise empty simulated disk will be placed on the outside, whereas in real systems with more utilized disks, the test files may be placed elsewhere, resulting in the simulator overestimating effective bandwidth.

4.3. System call latency

In addition to microarchitectural characteristics, LMBench measures the latencies of various system calls. Results for both systems are shown in Table 3. Relative error is calculated as latency difference divided by the latency of the reference system.

Overall, system call latencies do not match the reference and simulated systems as well as do the architectural characteristics. In most other cases, ML-RSIM/Lamix underestimates the cost of system calls compared to the reference kernel. This error leads to conservative results for studies that attempt to optimize operating system performance, since such evaluation uses a more efficient baseline system. The reason for the large discrepancies is found in the different structures of the two kernels. While Lamix is a traditional non-preemptive 32-bit uniprocessor kernel, both Irix and Solaris are internally multithreaded, support application-level threads and 64-bit code, and are designed to scale to large numbers of processors. The resulting synchronization

Table 3
System call latencies

Parameter	Octane (μs)	ML-RSIM (μs)	Error (%)	Blade (μs)	ML-RSIM (μs)	Error (%)
getpid	2.61	2.57	1.53	0.75	0.67	11.33
read/dev/null	10.38	6.88	33.71	5.13	1.55	69.84
read file	10.38	7.76	70.98	4.40	2.18	50.42
write	12.24	6.21	49.27	1.53	1.54	−1.12
stat	49.45	51.12	−3.38	5.60	15.34	−173.94
fstat	8.00	4.08	49.00	2.22	1.09	50.76
open/close	73.76	59.61	19.18	7.45	17.06	−128.89
select, 10 FDs	11.78	8.69	26.23	8.97	2.06	77.07
select, 60 fds	43.16	37.80	12.42	44.42	8.29	81.33
signal install	8.16	2.78	65.88	1.42	0.55	61.50
signal handler	31.95	8.25	74.18	16.52	2.11	87.24
pipe	64.94	52.98	18.42	16.37	20.69	−26.43
socket	74.24	55.68	25.00	26.82	20.39	23.96
fork/exit	1592.75	12785.00	−702.70	416.14	7566.00	−17181.30
fork/exec	4894.00	13484.00	−106.53	577.80	7898.00	−12669.10

overhead and the cost of supporting thread-specific signals increases system call latencies on the reference systems.

However, in both cases the latency of the simple getpid( ) system call closely matches that of the reference platform. This system call performs very little actual work as it retrieves a single value from the process control block, hence this experiment essentially measures the cost of entering and leaving the kernel. In many Unix kernels the relevant code sequence is nearly identical for system calls, external interrupts, and traps. Consequently, the good match between the systems confirms that Lamix implements a realistic model for saving and restoring process state.

Process creation is several times more expensive on Lamix than it is on the reference platforms, which indicates a deficiency in the simulator kernel. Lamix's simple memory management scheme does not support copy-on-write, hence the fork( ) system call copies the entire address space of the parent process to the child. Given the small code and data size of the microbenchmarks, the stack is a major contributor to the memory footprint of the process. Simulated measurements with a smaller stack lowers process creation costs to within a factor of two of the reference systems, confirming that the lack of copy-on-write functionality is the main source of this discrepancy.

This difference is a direct result of the original design goal of ML-RSIM, and shows how simulator design is a trade-off between accuracy and complexity. One of ML-RSIM's original design targets was to provide a platform to investigate and optimize the cost of I/O operations. As such, process creation is a necessary feature but does not need to be modeled accurately. Implementing a fully-featured virtual memory subsystem with demand-paging and copy-on-write functionality would have provided no additional benefit towards the original goals, but would have added considerably to the overall complexity of the simulation system.

Overall, the system call latency results bound the application domain that ML-RSIM/Lamix is able to simulate accurately. The cost of most system calls is underestimated under Lamix, compared to the reference platforms. Consequently, studies concerned with system call performance find in ML-RSIM an aggressive baseline model and yield conservative performance gains. However, care must be taken that the unrealistically expensive fork( ) system call does not overly influence results. Applications that frequently spawn new processes are not well served by the current implementation of this service. The Lamix kernel itself uses this system call during boot time to start the simulated applications, but gathering of simulation statistics begins only after the application has been loaded and thus does not include the fork( ) system call.

4.4. Context switch cost

Context switch overhead is another area of large discrepancy between the two systems, but at the same time also shows large variation between systems and between successive experiments. LMBench measures context switch overhead for different numbers of processes of differing working set sizes. In addition to measuring the cost of saving and restoring the process state, the benchmark also includes the cost of cache and TLB miss due to the context switch. Fig. 5 shows the context switch latency for the reference and simulated systems, for a 16 kbyte working set size. Note the different scale on the y-axis between the two graphs.

In the default configuration, ML-RSIM overestimates the context switch cost by a factor of two to six, especially for small numbers of processes or large working sets. Only for large numbers of

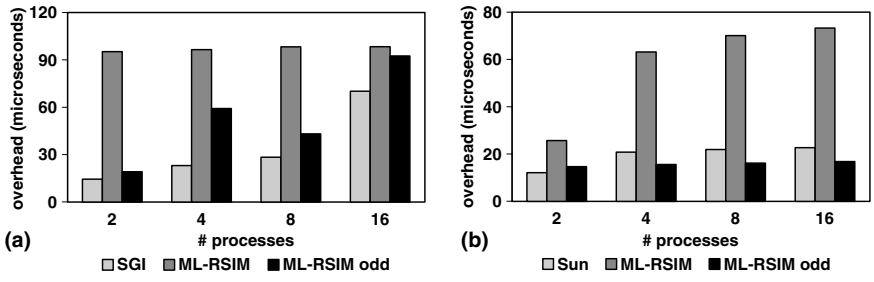


Fig. 5. Context switch latency: (a) SGI Octane and (b) Sun Blade.

processes is ML-RSIM able to approach the switch latency of the reference system.

The main source of this discrepancy is the variation in physical memory layout that affects the number of level-2 cache conflicts. By default, ML-RSIM's static memory management scheme aligns the physical address spaces of processes at powers of two, thus maximizing L2-cache conflicts between processes, resulting in worst-case context switch costs. The third bar in each group in Fig. 5 shows context switch costs for Lamix recompiled so as to not align physical address spaces at powers of two. Due to the reduction in L2-cache conflicts, Lamix exhibits context switch costs that approach that of the reference platform to within a factor of two.

In practice, the impact and significance of this difference will vary between experiments. In multiprogramming environments consisting of different processes, level-2 cache conflicts are less likely since different processes exhibit different memory layouts and operate in different code and data regions. On the other hand, multiple instances of the same program will suffer from the worst-case level-2 cache conflicts and thus overestimate context switch costs. In such cases, changing the default alignment of physical address spaces in Lamix will mitigate this effect.

4.5. System call bandwidth

LMBench measures the bandwidth of several operating system services, including reading of regular files from the file cache and interprocess communication with Unix domain sockets and pipes. Since these operations do not involve any I/O transactions, the experiments essentially measure the cost of copying data between protected address spaces.

When reading from an open file, ML-RSIM overestimates the bandwidth of the SGI Octane reference platform by 20–40%, resulting in a determination coefficient (r^2) of 0.606. Since the micro-architectural bandwidth measurements show a stronger correlation, some of this discrepancy is attributed to different implementations of the copy routines in the two kernels. In addition, differences in physical memory layout and in the resulting L2-cache conflict misses can lead to significant discrepancies, especially for small requests. Indeed, the modeling error is largest for transfers of less than 16 kbytes. When including opening and closing the file in the overall data transfer, ML-RSIM approximates the performance of the reference platform more closely with a determination coefficient of 0.925.

For the Sun Blade platform, ML-RSIM similarly overestimates effective bandwidth, resulting in determination coefficients of 0.67 and 0.64. For this platform, the overall shape of the bandwidth curves matches better than that for the SGI Octane, but both curves are slightly offset thus leading to a similar overall error. File system differences play an insignificant role, as the data transfer occurs only between the file cache and a user buffer.

On the other hand, ML-RSIM generally underestimates interprocess communication bandwidth by approximately 30%. Both Unix domain socket and pipe bandwidth is measured between multiple processes; consequently, results are very sensitive to the context switch cost and the physical memory layout. The same effect that increases context switch cost in Lamix degrades interprocess copy bandwidth. Both the source and destination buffers in the two processes map to the same locations in the level-2 cache, increasing the number of conflict cache misses and reducing bandwidth. Tests on the reference platforms also show a large variation by nearly a factor of two between successive measurements due to varying alignments of physical memory in subsequent runs. A similar but smaller variation can be reproduced in the simulator by compiling Lamix with different parameters for the physical memory allocation scheme.

Interestingly, for the Sun Blade system bandwidth differs by a factor of two between Unix sockets and pipes. Lamix, following the BSD design, treats pipes as a special case of Unix sockets and achieves nearly the same bandwidth in both cases, whereas Solaris achieves significantly higher bandwidth for pipes. In effect, the bandwidth observed on the simulated system is close to the average of the bandwidths measured on the reference system. Again, these observations confirm that different kernel organizations can lead to significantly different behavior, and highlight the challenge of validating a system simulator.

4.6. File system performance

LMBench measures file system performance as the number of file creations and deletions performed per second. The different file system structures of the systems leads to a large discrepancy, as shown in Table 4.

Table 4
File system performance in creations/deletions per second

File size (kbyte)	SGI Octane Irix 6.5	ML-RSIM Lamix	Sun Blade Solaris 8	ML-RSIM Lamix	Sun Ultra 1/140 Solaris 6	Sun Ultra 60/450 Solaris 7
0	528/350	63/120	5489/11,959	162/167	38/87	56/119
1	210/247	35/64	3780/11,882	54/167	37/35	63/59
4	207/229	33/60	4012/11,409	56/167	28/37	52/63
10	218/227	31/56	718/9864	49/89	27/38	48/68

Irix's XFS is a modern journaling file system that achieves significantly higher performance for these metadata operations than does the BSD fast file system. Similarly, the Solaris 8 implementation of UFS achieves extremely high rates of metadata updates, indicating that the original BSD consistency requirement has been relaxed. For a more realistic comparison with other FFS-based commercial file systems, Table 4 also includes results for two SPARC platforms running previous versions of Solaris. These systems exhibit performance similar to that of the simulator and confirm that while Lamix's FFS cannot be compared with XFS or the latest Solaris UFS, it is comparable to previous-generation commercial FFS implementations. A journaling file system as well as the recent soft-updates optimization of UFS could be ported to Lamix if desired. Note, however, that this performance discrepancy only holds for metadata updates such as creating and deleting files. Results from previous sections demonstrate that file read and write bandwidth matches more closely that of the reference systems.

4.7. Validation conclusions

Simulator validation using microbenchmarks provides important insights into the accuracy of various components. Overall modeling error determines the degree of confidence in simulation results. Equally important, the validation results define the application domain that the simulation tool can model with high accuracy. ML-RSIM/Lamix shows good correlation with the reference systems in most architectural aspects. These results are encouraging, since they confirm that compute-intensive applications without a major I/O component can be modeled with high accuracy. The fact that such correlation can be achieved for two widely different reference systems demonstrates that the main design goal of ML-RSIM, to be representative of an entire class of systems, has been achieved.

ML-RSIM/Lamix underestimates the cost of most system calls. This leads to results that are conservative for studies concerned with the performance impact of operating system activity. Process creation is unrealistically expensive due to the simple memory management scheme employed in Lamix. Consequently, the simulation system is currently not suitable for workloads that rely on the frequent creation of processes. A redesign of Lamix's virtual memory subsystem would be necessary if ML-RSIM were to be used in an environment in which these effects are important.

File I/O and interprocess communication bandwidth show a varying error compared to the reference systems. The main source of this error is the differing physical memory management schemes of the two kernels. On the reference platforms, demand-paging, copy-on-write, and interactions with other processes cause considerable variations in physical memory layout between successive measurements, with a resulting difference in copy bandwidth due to varying level-2 cache conflicts. Lamix's simple and deterministic scheme can lead to unrealistically high or low numbers of cache conflicts that do not change between successive simulation runs. This difference is as much a result of the design goals of ML-RSIM as it is inherent to simulators in general.

ML-RSIM is designed to model basic I/O functionality such as interrupt handling, DMA transfers, and system calls. As such it does not emphasize accurate memory management. While implementing or porting a more complex memory management subsystem is feasible, the resulting kernel would be subject to much of the same non-determinism that can be observed in real operating systems. Small changes in the order of memory allocation requests can lead to very different physical memory layouts, thus making it more difficult to reproduce and compare simulation results. Due to the run-time cost of simulations, it is not usually possible to repeat experiments multiple times to

account for variability; thus, a deterministic simulator is essential.

5. Related work

Existing execution-driven simulation tools fall broadly into two categories. The first category, composed of commonly-used architectural simulators such as SimpleScalar [2] and RSIM [15], provides detailed processor and memory hierarchy models but focuses on user-level instructions only. These tools are invaluable for studying compute-intensive workloads, but the lack of operating system and I/O device models makes them unsuitable for I/O intensive applications. The SimpleScalar tool set targets the detailed simulation of single-program user-level code and as such is not able to account for operating system and I/O effects. RSIM is designed to simulate scientific shared-memory multiprocessor workloads and consequently includes a simple network model, but also lacks support for operating system and I/O effects. ML-RSIM, on the other hand, extends the modeling scope to include several I/O devices, virtual memory, and a fully-simulated operating system.

The second category of simulation tools includes full-system simulators, such as SimOS [16], SimOS-PPC [19], Pharmsim [5], SimICS [12], and M5 [3]. These system simulators are able to boot complete operating systems and can thus simulate a much broader set of workloads, including I/O and operating-system intensive codes. SimOS and its derivatives as well as SimICS trade model fidelity for simulation speed. Furthermore, SimOS and M5 currently boot commercial operating systems for which source code is not readily available, restricting modifications that can be made by users to the hardware models. The M5 system simulator focuses on accurately modeling network effects, thus complementing ML-RSIM's emphasis on disk I/O. It currently also runs only a commercial OS. ML-RSIM, on the other hand, does not provide a complete machine model and thus requires a custom OS. However, the operating system kernel is freely available in source code form and the smaller scope of the simulation system reduces its complexity and development and maintenance effort.

Timing-first simulation lowers the development and execution cost of accurate full-system simulators by combining a timing-accurate but functionally incomplete simulator with a functionally-correct full-system simulator [13]. Since the usually more complex timing simulator does not need to be functionally correct, implementation of advanced features can proceed incrementally, thus reducing implementation effort. ML-RSIM takes a different approach to reduce implementation cost and does not provide a complete machine model. Consequently, it is not able to fully leverage existing operating systems, but the reduced complexity of hardware models and system software also results in a system that is relatively easier to maintain.

Validation is an important part of developing a simulation system, as it provides insight as to how conclusions and design trade-offs based on simulation may be aliased by modeling errors. In addition, it can help establish a level of significance and confidence in the final conclusions.

Several existing simulators have undergone a similar validation process. A version of SimpleScalar has been validated against a Compaq Alpha workstation [7]. Performance counter measurements and statistics collected by the simulator are compared to provide a detailed view of simulator inaccuracies. While this approach is able to pinpoint sources of modeling errors more clearly than the observation of microbenchmark results, it relies on the availability of suitable performance counters. In some cases, the exact specification of these counters is not clear. Using LMBench to characterize microarchitectural characteristics can eliminate or complement the use of performance counters. LMBench's capabilities and scope exceed those of most performance counters, providing statistics not only related to isolated events, but also related to whole system behavior. In fact, this study shows that accurate microarchitectural models are not sufficient to ensure that overall system behavior will be accounted for correctly.

The FLASH group validated the simulator used before and during prototype construction against the final hardware [9]. Unlike other validation efforts, this work investigates how accurately the simulator predicts trends, rather than focusing on absolute metrics such as execution time. Given that many simulators are in fact used to predict the relative impact of new techniques, the FLASH approach more closely measures the desired characteristics of the tool. While parallel architecture speedup and scalability trends are relatively easy to recreate on actual hardware, microarchitectural details such as instruction window size and memory system organization cannot be varied easily in isolation on real hardware. As a result, validation studies

that compare absolute metrics are necessary when investigating changes to microarchitectural details.

Calibration, the process of refining simulator models until they match a hardware prototype to a desired degree [4], is similar to validation but has a different objective. Unless the hardware models are general enough to be applicable to a variety of systems, this calibration can result in a simulation tool that contains too many idiosyncrasies of one specific system to be broadly applicable. The validation work presented here intentionally does not curve fit or specialize hardware models since the simulator is designed to represent a large class of systems.

6. Conclusions

This article presents a high-fidelity simulation system designed for I/O and operating system intensive workloads. ML-RSIM combines accurate models of modern workstation hardware with a fully-functional Unix-compatible operating system. ML-RSIM is based on RSIM v1.0 and extends the original processor model with an I/O subsystem, detailed bus and memory controller models and support for privileged instructions and virtual memory. These extensions allow ML-RSIM to simulate a Unix-like operating system. This unique combination of features makes the simulation environment well-suited for exploring the interaction of system architecture, system software and applications.

A validation study comparing the simulator to two diverse reference systems demonstrates the level of accuracy achieved by the system, but more importantly highlights its strengths and weaknesses. The validation process presented here uses LMBench to compare microarchitectural characteristics and operating system performance with an SGI Octane and Sun Blade-1000 workstation. Microarchitectural parameters generally show a close correlation between ML-RSIM and the reference platforms, indicating that all major hardware components are modeled accurately. System call latencies and other operating system performance parameters show considerable discrepancies between the systems, largely as a result of the different internal structures of the operating system kernels. This understanding and the detailed results presented here help researchers to evaluate the suitability of the tool for specific research questions and to judge the validity of results. In particular, the simulator is appropriate and useful for simulating workloads in which kernel interaction is an important part of the application, and where underestimating the cost of various system calls would not invalidate the experimental results. Currently ML-RSIM is less appropriate for evaluating workloads that frequently create processes, unless improvements are first made to the memory management subsystem of Lamix.

ML-RSIM is freely available for researchers to use and modify. In addition to several refinements and functionality enhancements, a simultaneous multithreading version is under development. The ML-RSIM tools, like most available simulators, provide the basis on which researchers implement and test their own techniques and enhancements. Ultimately it is the responsibility of those researchers to ensure that the simulator being used is the appropriate tool for the task and that the simulator provides sufficient fidelity and accuracy in the areas that impact the conclusions drawn from its use.

Acknowledgements

We would like to thank the RSIM project at the University of Illinois at Urbana-Champaign for developing and providing their simulation environment on which ML-RSIM is built. Many people have contributed to the development of ML-RSIM. In particular, we would like to acknowledge the significant contributions of Lixin Zhang and Branden Moore. This work was in part supported by Defense Advanced Research Projects Agency under agreement number N0003995C0018 and F306029810101, by an NSF Research Infrastructure Grant Number CDA9623614, and by a Graduate Research Fellowship from the University of Utah Graduate School for the academic year 2000/2001.

References

[1] Adaptec, Inc. AIC-7770 Data Book, 1992.
[2] T. Austin, E. Larson, D. Ernst, SimpleScalar: an infrastructure for computer system modeling, IEEE Computer 35 (2) (1998) 59–67.
[3] N. Binkert, E. Hallnor, and S. Reinhardt, Network-oriented full-system simulation using M5, in: Proc. Sixth Workshop Computer Architecture Evaluation using Commercial Workloads, February 2003.
[4] B. Black, J. Shen, Calibration of microprocessor performance models, IEEE Computer 31 (5) (1998) 59–65.
[5] H. Cain et al., Precise and accurate processor simulation, in: Proc. Fifth Workshop Computer Architecture Evaluation using Commercial Workloads, 2002.

[6] W. Courtright II, G. Gibson, M. Holland, L. Neal Reilly, J. Zelenka, RAIDFrame: A Rapid Prototyping Tool for RAID Systems, Version 1.0, tech. report, Carnegie Mellon University, Pittsburgh, PA, 1996.
[7] R. Desikan, D. Burger, S. Keckler, Measuring experimental error in microprocessor simulation, in: Proc. 28th Int'l Symposium Computer Architecture, ACM Press, New York, NY, 2001, pp. 266–277.
[8] G. Ganger, B. Worthington, Y. Patt, The DiskSim Simulation Environment Version 1.0 Reference Manual, tech. report CSE-TR-358-98, Department of Electrical Engineering and Computer Science, University of Michigan, Michigan, 1998.
[9] J. Gibson et al., FLASH vs. (simulated) FLASH: closing the simulation loop, in: Proc. 9th Int'l Conf. Architectural Support Programming Languages and Operating Systems, ACM Press, New York, NY, 2000, pp. 49–58.
[10] M. McKusick, K. Bostic, M. Karels, J. Quarterman, The Design and Implementation of the 4.4. BSD Operating System, Addison Wesley Longman Inc., 1996.
[11] E. Lee, Performance Modeling and Analysis of Disk Arrays, Ph.D. Dissertation, tech. report CSD-93-770, University of California, Berkeley, CA, 1993.
[12] P. Magnusson et al., SimICS: a full system simulation platform, IEEE Computer 35 (2) (1998) 50–58.
[13] C. Mauer, M. Hill, D. Wood, Full system timing-first simulation, in: Proc. ACM SIGMETRICS Conf. Measurement Modeling Computer Systems, ACM Press, New York, NY, 2002, pp. 108–116.
[14] MIPS Technologies, MIPS R10000 Microprocessor User's Manual, Version 2.0, 1996.
[15] V.S. Pai, P. Rangnathan, S.V. Adve, RSIM Reference Manual, Version 1.0, tech. report 9705, Department of Electrical Computer Engineering, Rice University, Houston, TX, 1997.
[16] M. Rosenblum et al., Using the SimOS machine simulator to study complex computer systems, ACM TOMACS Special Issue on Computer Simulation (1997) 79–103.
[17] L. Schaelicke, L-RSIM: A simulation environment for I/O intensive workloads, in: Proc. 3rd Annual IEEE Workshop Workload Characterization, 2000, pp. 83–89.
[18] L. Schaelicke, M. Parker, ML-RSIM Reference Manual, tech. report TR 02-10, Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, In., 2002.
[19] The SimOS-PPC Project, http://www.cs.utexas.edu/users/cart/simOS/.
[20] L. McVoy, C. Staelin, lmbench: portable tools for performance analysis, in: Proc. USENIX Annual Technical Conference, Usenix Assoc., Berkeley, CA, 1996, pp. 279–294.
[21] D. Weaver, T. Germond, The SPARC Architecture Manual Version 9, PTR Prentice Hall Inc., 1994.

Lambert Schaelicke's research interest include high-performance computer architecture and performance modeling and evaluation. As an assistant professor at the University of Notre Dame, he has directed a research project that developed and implemented a high-speed network intrusion detection system. In addition, he was involved in the design and evaluation of a novel high-end computer architecture based on merged logic-DRAM technology. He is now working as a performance engineer in the Intel Itanium group. He received a Diploma in Computer Science from the Technical University of Berlin, Germany, and a Ph.D. in Computer Science from the University of Utah.

Mike Parker is a system and processor architect at Cray. His research interests include processor and scalable parallel computer architecture, performance analysis, and VLSI design. Past work includes improving I/O and message-passing performance through direct user-level event notifications; investigating memory system, processor, and synchronization improvements on scalable parallel shared-memory systems; architecture and hardware development of advanced memory systems; and reducing communication latency and overhead in clusters of workstations by using efficient protocols and closely coupling the communication hardware with the CPU. He received his BSEE from the University of Oklahoma in 1995 and is expecting his Ph.D. from the University of Utah in 2005.