Measuring Experimental Error in Microprocessor Simulation

Rajagopalan Desikan*, Doug Burger, and Stephen W. Keckler

Computer Architecture and Technology Laboratory
Department of Computer Sciences
*Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712-1188

Abstract

We measure the experimental error that arises from the use of non-validated simulators in computer architecture research, with the goal of increasing the rigor of simulation-based studies. We describe the methodology that we used to validate a microprocessor simulator against a Compaq DS-10L workstation, which contains an Alpha 21264 processor. Our evaluation suite consists of a set of 21 microbenchmarks that stress different aspects of the 21264 microarchitecture. Using the microbenchmark suite as the set of workloads, we describe how we reduced our simulator error to an arithmetic mean of 2%, and include details about the specific aspects of the pipeline that required extra care to reduce the error. We show how these low-level optimizations reduce average error from 40% to less than 20% on macrobenchmarks drawn from the SPEC2000 suite. Finally, we examine the degree to which performance optimizations are stable across different simulators, showing that researchers would draw different conclusions, in some cases, if using validated simulators.

1 The need for validated simulators

Because of the time, expense, and complexity of constructing hardware prototypes, the computer architecture research community relies heavily on simulation to evaluate new ideas. As a result of the increasing complexity in modern computer systems, and with the corresponding difficulties of building and maintaining software tools that reflect that complexity, simulation infrastructure is now widely shared among both academic and industry researchers. A majority of the papers published in recent conferences use shared tools such as the SimpleScalar tools [4], RSim [17], SimOS [19], and CACTI [10]. While some tools, such as CACTI, have been validated against real hardware, none of the microarchitecture simulators have been subject to such scrutiny. This lack of performance validation may be consistently introducing error into experimental studies, which, if sufficiently large, may cause researchers to draw incorrect conclusions.

Black and Shen introduce a taxonomy for describing the source of inaccuracy in simulations [2]. Modeling errors result from unintentional performance bugs lurking in the simulator code. Specification errors occur when the developer models behavior that does not match that of the specified system, possibly through misunderstanding of the specification or invalid assumptions about the underlying technology. Abstraction errors arise by modeling the system at an insufficient level of detail. Common sources of abstraction errors include neglecting to simulate minor structures in the microarchitecture, overlapping of operations, finite pipelining of storage structures, non-modeled signal delays, or overly simplistic simulation of state machines and corner cases.

While modeling errors are gradually eliminated over a tool's lifetime, significant bugs may be introduced into the code when functionality not anticipated by the original tool designer is added. Specification errors often arise when the designer neglects physical constraints, such as clock speeds, and tend to occur more frequently with increased system complexity. Abstraction errors are not typically reduced as a tool matures, and will become more frequent and serious as global wire delays introduce layout-determined communication delays [1]. Consequently, the relative error in microprocessor simulators is likely growing, thus increasing the motivation for validation of experimental simulators. Worse, the magnitude of the errors, as shown by Black and Shen [2], may be sufficiently large that researchers draw the wrong conclusions when evaluating a new technique. Our results show that the error in common simulators is often larger than the performance gains yielded by new architecture ideas reported in the literature.

In this paper, we describe our validation of an Alpha 21264 simulator, called `sim-alpha`, against a Compaq DS-10L workstation. In Sections 2, 3, and 4, we describe the 21264 architecture and DS-10L platform, the validation of `sim-alpha` using a suite of microbenchmarks, and our methodology for approximating native memory

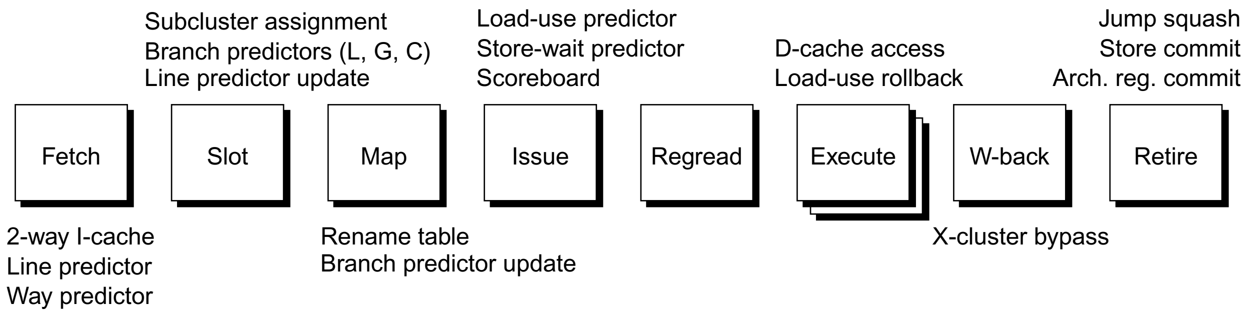

Figure 1. Alpha 21264 pipeline

system parameters. In Section 5, we present the validation of `sim-alpha` using ten benchmarks drawn from the SPEC2000 suite. We also present a case study which demonstrates that different conclusions may be reached when a researcher uses to a high-fidelity, validated simulator. In Section 6, we describe related microprocessor simulation validation efforts. In Section 7, we conclude that simulators which do not model real targets tend to overestimate performance, since they are not subject to implementation constraints. We also provide a list of recommendations for improving the rigor of simulation-based, empirical architecture research.

2 Alpha 21264 review

The Compaq Alpha 21264 processor [6,11] is an out-of-order issue microarchitecture that is designed to operate at frequencies higher than other processors of its generation. We chose the 21264 because the specification of its microarchitecture has been published in much more detail than chips from other vendors. We relied primarily on refereed publications by Kessler [11, 12], the Alpha 21264 Compiler Writer's Guide [6], and the hardware reference manual [5].

2.1 Pipeline stages

The pipeline of the 21264, shown in Figure 1, includes seven basic stages plus a variable number of execute stages, depending on the instruction. The fetch stage of the pipeline accesses the 64KB, 2-way set-associative, 64-byte block instruction cache in a single cycle. A 128-bit packet of four instructions, called an octaword in the Compaq Alpha literature, is fetched on every cycle. All fetches must be aligned on an octaword boundary. To improve fetch bandwidth, the I-cache contains a line predictor and a way predictor. The line predictor contains a prediction used to perform the next i-fetch, which requires a pointer to an I-cache set and the offset to an octaword within a line. The way predictor assists the I-cache by predicting which line in the set to access, incurring a two-cycle bubble upon a way misprediction. The fetch stage will prefetch up to four 64-byte instruction cache lines when an I-cache miss occurs.

The slot stage contains the tournament branch predictor, which is a combination of three predictors: a two-level local predictor, a path-based global predictor, and a choice predictor that chooses between the local and global predictions for individual branches. The local predictor (L) contains 1024 10-bit local histories, which are used to access a 1024-entry table of 3-bit saturating counters. The global predictor (G) uses a 12-bit history shift register to index into a 4K-entry table of 2-bit saturating counters. The choice predictor (C) uses the PC to index into a 4K-entry table of 2-bit saturating counters, which select the local or global prediction for each branch. In addition to the branch prediction, the slot stage also performs a table lookup to assign each instruction in the fetched packet to one of the two subclusters in the execution core, according to predetermined rules stored in a table.

The map stage performs register renaming, assigning one of the 80 physical registers (40 integer and 40 floating point) to each instruction's result. In addition, the map stage assigns each instruction a unique 8-bit number, called the *inum*, and loads the instructions into the reorder buffer and collapsible issue queues. The rename table is designed not to stall as long as at least eight free physical register names are available.

The issue stage selects instructions from the 20-entry integer queue and the 15-entry floating-point queue. Instructions are issued to one of the two register file/execution unit clusters, each of which has an upper and a lower subcluster. The *subcluster* assignment is determined in the slot stage. The scheduler and scoreboard of the issue stage determine the *cluster* to which an instruction should be sent, since there is a one-cycle delay required to bypass instructions between clusters (but not subclusters). The collapsible queues issue instructions strictly based on the *inum*, with oldest instructions issued first. The issue stage uses two predictors: a *load-use* predictor, which is a four-bit counter that speculates whether a load instruction will hit in the level-one data cache, and a *store-wait* predictor, which is a 1024x1 bit table that speculates whether a load

instruction	latency
integer ALU	1
integer multiply	7
integer load (cache hit)	3
FP add, multiply	4
FP divide/sqrt (single precision)	12/18
FP divide/sqrt (double precision)	15/33
FP load (cache hit)	4
unconditional jump	3

Table 1: 21264 instruction latencies

should be issued if there are earlier, unresolved stores that may share the same address as the load. The 21264 can issue a maximum of six instructions per cycle: two floating-point instructions and four integer operations. In Table 1, we list the number of cycles required to execute the various instruction classes of the 21264.

In the write-back stage, results are bypassed to issued consumers, and are written back into the register file. Stores are sent from the store buffer to the memory system, results are written back to the architectural register file, and instructions are retired from the reorder buffer. Support exists in the reorder buffer for bursty retires, up to eleven per cycle, for those cases in which a single instruction that has stalled retirement of a large number of instructions completes.

In addition to the instruction cache, the 21264 on-chip memory system includes a 64KB, two-way set associative data cache with 64-byte blocks, an eight-entry victim buffer for data cache misses, an eight-entry, combining miss address file (or MAF)—commonly known as a miss status holding register (or MSHR) [13]—to process off-chip cache misses, and two dedicated off-chip connections: a 128-bit channel to the off-chip backside (level-two) cache, and a connection to the 64-bit memory bus.

2.2 DS-10L platform

The Compaq DS10-L workstation that we used to validate the simulator contains a single 21264 processor clocked at 466 MHz, a 2 MB L2 cache (direct mapped, with 64 byte blocks), and 256 MB of physical memory. The workstation runs version 5.1 of Compaq Tru64 UNIX, and all of our microbenchmarks were compiled with version 6.3-025 of the Compaq C compiler. The macrobenchmarks were compiled using the native DEC C V5.9-008 compiler on Digital UNIX V4.0, and the DIGITAL Fortran 90 V5.2-705 compiler.

2.3 Measuring native system performance

To validate the simulations, an accurate time count must be obtained for each benchmark running on the native system. We used the Compaq DCPI (DIGITAL Continuous Profiling Infrastructure) tool to measure time. DCPI employs hardware counters to measure execution time (in cycles), number of instructions committed, and a few other hardware events, such as load replay traps and TLB misses. The events may be sampled at several intervals, from 1,000 cycles to 64K cycles. Larger sampling intervals dilate the execution time less, but introduce additional error when counting events. We chose a sampling interval of 40,000 cycles, which showed the best trade-off between sampling error and instrumentation dilation.

3 Microbenchmark suite

To isolate and validate specific aspects of the 21264 implementation, we constructed a suite of microbenchmarks. All of the benchmarks, except those specifically targeted at the memory system, are instruction cache, data cache, and TLB resident, eliminating system-level effects. We break the microbenchmarks into three categories: control (“C” microbenchmarks), execution (“E” microbenchmarks), and memory (“M” microbenchmarks), each of which stresses different stages of the pipeline, and are discussed in the subsections below.

3.1 Front-end benchmarks

The 21264 relies heavily on control and dependence speculation, using five distinct predictors to keep the instruction pipe as full as possible. We designed four microbenchmarks to stress different parts of the front end: control-conditional (C-C), control-recursive (C-R), control-switch (C-S), and complex-control (C-O).

C-C implements a simple if-then-else construct in a loop that is repeatedly executed, and alternates between taking and not taking the conditional branch. We discovered that two different implementations of the Alpha compiler, Tru64 Unix 5.1 Compaq C V6.3-025 and Digital UNIX 4.0 DEC C V5.9-008, generated slightly different code sequences, called C-Ca and C-Cb, respectively. Different padding with no-ops (`unop` in the Alpha instruction set) results in the line predictor being trained by different branches in C-Ca and C-Cb. Both cases were useful for tuning predictors in the front end of `sim-alpha` correctly.

C-R tests subroutine calls and the `bsr` instructions by incorporating a 1,000-level deep recursive function call within an outer loop. C-Sn tests indirect jumps (`jmp`) with a 10-way case statement within a loop. We call the benchmark *control-switch-n* because each case statement is taken *n* times on consecutive loop iterations before moving on to the next case statement.

The C-O benchmark is a hybrid between C-C and C-S, which loops over an *if-then-else* statement, executing C-S2 in the *if* clause and C-S3 in the *else* clause.

3.2 Execution core benchmarks

The first of the execution core benchmarks, *execute-independent* (E-I), simply adds the index variable to eight independent, register-allocated integers twenty times each within a loop. The absence of memory operations, control hazards, or data dependences should allow close to ideal throughput on this microbenchmark. The second execute microbenchmark, *execute-float-independent* (E-F), performs the same computation as E-I, except on floating-point variables. The third execute microbenchmark, *execute-dependent-n* (E-Dn), implements n dependent chains of register-allocated integer additions within a loop. Each arithmetic instruction in the loop is dependent on the instruction n positions earlier. E-DM1 is simply E-D1 using multiply instructions instead of adds.

3.3 Memory system benchmarks

The first memory system benchmark, *memory-independent* (M-I), repeatedly executes independent loads, all resident in the level-one data cache, and adds their results and the loop index to a register-allocated scalar variable. This benchmark tests the level-1 D-cache bandwidth. The *memory-dependent* microbenchmark (M-D) tests level-1 D-cache latency by executing a loop that walks a linked list, waiting for each load to complete before starting the next one. By executing loads one at a time, we measure the access latency of the L1 D-cache. The *memory-L2* (M-L2) and *memory-memory* (M-M) microbenchmarks are similar, except that the array access patterns in M-L2 are coded to miss in the L1 D-cache on every reference, and in M-M to miss in both the L1 and L2 caches. Finally, the *memory-instruction-prefetch* (M-IP) benchmark tests the efficacy of instruction prefetching by iterating over an enormous loop that flushes the L1 instruction cache with each iteration.

3.4 Validation process

Our 21264 simulator, called `sim-alpha`, was written using the SimpleScalar environment, the Alpha ISA definition file, the loader, checkpoint functionality, and a number of data structures from `sim-outorder`, the superscalar simulator that comes with the SimpleScalar tools [4]. Nearly all of the timing simulation code in `sim-alpha` was written from scratch.

In Table 2, we depict the results of our validation for each microbenchmark. The second column contains the IPC of the native Alpha, measured with DCPI. The IPC values range from 0.56 (C-S1, which incurs a line misprediction on every switch statement) to a perfect pipeline utilization of 4.00 (E-I, which incurs no structural, data, or control hazards in the main loop). The third and fourth columns contain the IPC values and errors (compared to the native DS-10L) for the initial version of `sim-alpha` that had been run on simple tests but not validated. We call that early version `sim-initial`. The fifth and sixth columns contain the IPC values and errors for our most current, validated version of `sim-alpha`. The right-most two columns contain the IPC values generated by the SimpleScalar simulator, which we discuss later. All errors are computed as a percentage difference in CPI.

The microbenchmarks running on `sim-initial` show a mean error of 74.7% compared to the reference machine. The mean errors are computed as the arithmetic mean of the absolute errors. While some of the errors are negligible, in particular simple timing cases like E-F and E-D1, most of the microbenchmarks show errors of 20% or greater. The C-Ca, C-Cb, and C-R microbenchmarks all underestimate performance by over 100%. While most of the microbenchmarks underestimate performance, E-DM1 and C-S1 overestimate performance by 85.7% and 31.2%, respectively.

We used a variety of strategies to discover and eliminate the sources of error in `sim-initial`. We first made all resources in the pipeline perfect, and then searched for performance bottlenecks. In addition to measuring total execution time, we also monitored event counts, such us mispredictions requiring rollback in various predictors. Below, we discuss some of the errors we discovered and fixed in `sim-initial` to improve its accuracy with respect to the reference machine.

Instruction Fetch: Since the front end of the 21264 is the most complicated component of the processing core, with many interacting state machines, it is unsurprising that most of the errors occurred there. We addressed the C-C and C-R errors first, since they were the largest in magnitude. The most significant contributor to the C-C and C-R error was an excessive branch misprediction penalty. `sim-initial` waited until after the execute stage to discover a line misprediction and initiate a full rollback. We determined that there is an undocumented adder that resides between the fetch and slot stages. That adder computes the target for PC-relative branches early, and is used by the branch predictor to override the line predictor in some cases.

We experimentally determined the rules for the interaction between the line predictor and the branch predictors. The branch predictor will overrule the line predictor on conditional or unconditional branches (not jumps) if it predicts taken, can compute the target early, and the target computation disagrees with the line prediction. Furthermore, we chose the initialization bits for the line predictor (01) that minimized error, and adjusted the line predictor state machine to minimize error as well. When testing the macrobenchmark *eon*, we noticed that performance was extremely poor compared to our reference. That benchmark exhibits an unusually high number of way mispre-

	Alpha 21264	Initial simulator (sim-initial)		Validated simulator (sim-alpha)		SimpleScalar 3.0b (sim-outorder)	
benchmark	IPC	IPC	% error	IPC	% error	IPC	% difference
C-Ca	1.80	0.38	-498.1%	1.87	4.3%	3.17	28.2%
C-Cb	1.87	0.52	-260.4%	1.87	0.6%	3.00	37.8%
C-R	2.65	0.89	-198.4%	2.66	0.3%	3.54	25.2%
C-S1	0.56	0.81	31.2%	0.60	6.4%	0.88	36.1%
C-S2	0.85	0.82	-3.6%	0.86	2.1%	1.33	36.5%
C-S3	0.95	0.87	-8.5%	0.95	0.5%	1.64	42.2%
C-CO	1.75	0.53	-273.6%	1.74	-0.6%	2.05	3.0%
E-I	4.00	3.31	-20.9%	3.99	-0.4%	3.99	-0.4%
E-F	1.01	1.01	-0.1%	1.01	0.2%	1.01	0.2%
E-D1	1.03	1.04	0.3%	1.04	0.4%	1.04	0.4%
E-D2	2.16	2.15	-0.0%	2.15	0.0%	2.21	2.6%
E-D3	2.72	2.99	9.3%	3.07	11.5%	3.19	14.8%
E-D4	2.79	2.89	3.6%	2.80	0.3%	4.00	30.2%
E-D5	3.30	3.23	-2.1%	3.50	5.8%	4.00	17.6%
E-D6	3.11	3.31	6.1%	3.15	1.3%	4.00	22.2%
E-DM1	0.15	1.04	85.7%	0.15	-0.3%	0.15	-0.3%
M-I	2.98	2.39	-24.2%	2.99	0.6%	3.00	0.7%
M-D	1.66	1.25	-32.9%	1.66	0.4%	1.26	-31.1%
M-L2	0.36	0.34	-4.0%	0.35	-0.9%	0.55	35.6%
M-M	0.07	0.07	-8.2%	0.08	4.2%	0.07	-0.3%
M-IP	1.75	0.89	-97.9%	1.76	0.5%	1.22	-43.1%
Mean			74.7%		2.0%		19.5%

Table 2: Microbenchmark validation

dictions in the I-cache. We found that we had been charging an extra cycle to access the way predictor. Elimination of that extra cycle reduced the error dramatically.

In addition to the line predictor/branch predictor interaction, another major source of error for C-C and C-R was in the speculative predictor updates. We did not initially update any of our predictors speculatively. We determined that the branch history shift register, which is used to index into the global and choice branch predictors, and the return address stack are both updated speculatively and rolled back upon recovery from a mis-speculation. The line predictor is also updated speculatively, but is fixed if a speculation was determined to be incorrect. Finally, we determined—using the microbenchmarks—that no penalty is applied for squashing instructions in a fetched octaword that follow a taken branch within the same octaword. We had been modeling a one-cycle penalty for clearing those instructions from the fetch queue.

We determined that the C-S benchmarks were performing too well because we were undercharging for indirect jumps. The `jmp` instruction must proceed through the pipeline because its target cannot be computed with the slot stage adder. We determined with C-S1 that each mispredicted `jmp` incurs a 10 cycle penalty to flush and restart the pipeline.

Execute: When running E-I, assuming perfect fetch and memory, we noticed that the add throughput was only 2 instead of the expected 4 instructions per cycle. We had inadvertently used two multipliers and two adders as the four execution pipes, rather than the one adder/multiplier and three adders resident in the 21264. This is an easy trap to fall into since many simulators [4] specify functional units as generic resources, and neglect the restricted mappings of instruction to units.

We also discovered several abstraction errors that affected performance in the execute stages of the pipeline. First, `sim-initial` did not remove unops, the Alpha "universal no-op", in the map stage, but instead allowed them to proceed until the retire stage and consume real issue slots. Errors in the E-Dn microbenchmarks were caused by incorrect modeling of the issue prioritization. To improve performance, we originally designed `sim-alpha` with an aggressive scheduler that minimized cross cluster delays, sending an older instruction to one cluster if a younger instruction needed to go to the other[1]. That policy increased E-Dn performance beyond that of the 21264, as the actual 21264 scheduling logic is not that sophisticated. If an instruction can be issued to either cluster, the scheduler issues it to cluster 1 if it has been slotted to the upper subcluster, and cluster 0 if it has been slotted to the lower subcluster, regardless of the issue restrictions of younger instructions in the queue.

1. An instruction must be issued to a specific cluster if it has a source instruction that has not yet written its result from one cluster to the other.

The 11.5% error in E-D3 is due to a minor approximation in the way we implement bypassing; when an instruction is bypassed, we subtract the latency that is saved as a result of the bypass from the latency of the execution. That simplification results in different issue orders than might occur in the actual 21264 core. Future versions of `sim-alpha` will contain a scheduler that accurately represents the 21264 instruction issue policies.

Memory: We originally noticed an unusually high number of load traps in `sim-alpha` resulting from multiple loads to the same address executing out of order, and thus violating the coherence requirements built into the 21264 memory system. We hypothesized that the simulator was too conservative because it masked out the lower three bits of the addresses before comparing them in the load-trap identification logic. The error dropped dramatically in M-D when the entire memory address was used to detect these conflicts. We also found that the L2 latency shown in M-L2 was a cycle longer than that specified in the Compiler Writer's Guide. That anomaly was found to be a modeling error in which the simulator charged too many cycles for the register read stage on loads that missed in the cache. We were also charging one cycle too few for recovery upon load-use mis-speculation, which we discovered with the M-D benchmark. Finally, we did not initially implement the store-wait table, expecting it to make only a small difference in performance. However, when we observed the large number of store replay traps in the C-R benchmark, we implemented that table and noticed a precipitous drop in error. The results in Table 2 for `sim-initial` include the store-wait table.

3.5 Validation results

Correcting the long list of errors described in Section 3.4 resulted in substantial improvements across most of the microbenchmarks, reducing the error from a mean of 74.7% to 2%. Some of the microbenchmarks have higher IPCs on `sim-alpha` (positive error); the highest is 11.5%, with E-D3. Others show higher performance on the DS-10L than on `sim-alpha` (negative error), the largest of which is -0.9%, for M-L2. Having both positive and negative errors is consistent with previous validation efforts [2].

3.6 Remaining sources of error

There are several sources of error that may be contributing to the remaining microbenchmark error. Our load-use speculation implementation is an approximation of the way a real 21264 works. The 21264 may speculatively issue an operation that is dependent on a load, anticipating that the load will hit in the cache. If the load causes a cache miss, all instructions that issued within the preceding two cycles are squashed and reloaded into the issue window. In `sim-alpha`, we squash and re-issue only the instructions that were dependent on the violating load.

We do not model the resource contention produced when the I-cache attempts to access a line that is in the midst of its line-predictor training. It is not clear from the 21264 documentation when instructions are removed from the collapsible issue queue. Kessler [11] states that the instructions are removed immediately after issue, whereas the 21264 Compiler Writer's Guide [6] states that issued instructions may be removed "two cycles or more" after issue. The delayed removal would facilitate cheaper roll-back upon load-use mis-speculation recovery, but would reduce performance when the issue window was full. We use the second strategy. We are also conservative when generating store traps, as we ignore the three low-order bits when comparing addresses for load-store violations. Consequently, a load and a store to different bytes in the same word will still produce a store replay trap.

In `sim-alpha`, only one register is consumed when a double word is loaded. In `sim-alpha`, we do not partition conditional move instructions into two separate internal instructions as is done in the 21264, although `sim-alpha` does accurately model the multiple dependences of the conditional move. Finally, our implementation of the memory system beyond the L1 cache is less accurate than that of the processing core. While this inaccuracy does not affect the cache-resident microbenchmarks, it does reduce the accuracy of the simulator on macrobenchmarks. We describe the effect of memory system inaccuracies on our benchmarks in the next section.

4 Memory system validation

A substantial challenge for any microprocessor simulator is to replicate the behavior of the DRAM and virtual memory systems. Access latency in modern DRAMs, such as synchronous and Rambus DRAM, is highly dependent on the stream of physical addresses presented to them, which in turn depends on the virtual to physical page mappings. Like many other microprocessor simulators, such as SimpleScalar [4] and RSim [17], `sim-alpha` does not simulate past the system call boundary, thus replicating the the page mappings of the native system is difficult if not impossible. Complete system simulators, such as SimOS [19] or SimICS [14], suffer from the same problem as the page mappings in the native machine depend on the set of allocated pages prior to starting and measuring the selected program. These mismatches between simulated and native page mappings can cause non-cache-resident benchmarks to experience error due to the variable DRAM access time. Our challenge, then, is to match the specifications and observable behavior of our memory system as closely as possible with that of the native system, and accept that some intrinsic error will not be completely eliminated.

4.1 Shortcomings of the memory system model

The `sim-alpha` memory system is the least accurate aspect of the simulator. We model the cache hierarchy in more detail than many other simulators, simulating contention at all buses, MSHRs with combining targets at each level, full hardware page table walks on TLB misses, and address translations for virtually indexed, physically tagged caches. We model the DRAM latency using the simulator provided by Cuppu, *et al.* [8].

However, the DS-10L external memory system differs significantly from our memory system. While we attempted to discover all of the relevant parameters of the memory system, we were thwarted by a lack of available documented information and observable data that could be used to empirically derive it. Below, we list all of the known and potential differences between our target memory system and the native machine. We do not model the memory controller in the DS-10L, which consists of the C-chips and D-chips; instead we add a constant number of overhead cycles to the DRAM latency. We do not model the split memory buses: the 64-bit bus from the processor pins to the D-chips, which runs at an unknown frequency, and the 128-bit, 75MHz bus from the D-chips to the memory array. We have approximated the bandwidth to the 186MHz L2 cache empirically, but are unable to validate it definitively. The native 21264 disallows caching virtual aliases in the L1 D-cache to enable overlap of the TLB lookup and the data and tag accesses. In `sim-alpha`, we do not enforce that restriction. We model neither the consumption of ports in the cache for returning misses, nor the consumption of a register write port when a load returns from a cache miss. Instead of forcing stores in the store-queue to wait until an idle L1 data cache cycle is available, we assume that writes can complete unimpeded. While the 21264 has an 8-entry miss address file (MAF) shared among the three caches to track outstanding cache misses, in our memory system, each cache has its own eight-entry MAF. Although the 21264 allows decoupling of tag and data accesses to allow a greater utilization of the data bus, we do not simulate the L1 data cache bus at this level of detail. Finally, while the 21264 executes TLB misses in software via PAL code, thus stalling the program, `sim-alpha` simulates a hardware walk of the five levels of page tables and does not stall the pipeline.

4.2 Memory system approximation

To reduce the effect of the unknown latencies, schedules, and bandwidth in our memory system, we tuned our DRAM and memory system parameters to produce minimal error across three memory-specific benchmarks: M-M (described in Section 3.3), *stream* [15], and *lmbench* [16]. The M-M microbenchmark walks a linked list to obtain the load latency for back-to-back memory operations. The *stream* benchmark measures bandwidth from main memory for four different "streaming" kernels (copy, scale, add, and triad), and reports the achieved bandwidth for each. The *lmbench* program performs repeated operations to memory, and reports the mean latency to each level in the hierarchy.

Unfortunately, since different reference streams will have different distributions of DRAM page misses, bank conflicts, and L2 conflicts caused by the physical memory page mappings, there is no set of memory system parameters that will minimize error uniformly across the macrobenchmarks. To minimize the error, we first set the memory system to be as close as possible to that of the DS-10L: a 64KB, 2-way set associative, 64-byte block, write-back level-one instruction cache and data cache; a three-cycle load-to-use penalty for a data cache hit; a 2MB, direct mapped, write-back, 64-byte block L2 cache with a 13-cycle load-to-use penalty; and the 8-entry, fully associative victim/write-back buffer. As in the native machine, the simulated DRAM is set to run at approximately 25% the processor speed.

To determine the configuration of the memory system, we first measured the execution time, in cycles, of M-M, *stream*, and *lmbench*, and then compared the results to those obtained from the simulator. We varied the RAS time, the CAS time, the precharge latency, and controller latency (0 or 1 cycles each way between the processor and the DRAM). We also compared an open-page policy, in which DRAM pages are kept active until a precharge is forced by a page miss, to a closed-page policy in which DRAM pages are closed and a precharge begun immediately after each access. Our experiments showed that the open page policy with a 2-cycle RAS, 4-cycle CAS, 2-cycle precharge, and total of 2 cycles of memory controller latency produced the least overall error between the DS-10L and `sim-alpha`. The difference in execution time between the native and simulated systems in M-M, *stream*, and *lmbench* is 2.8%, -6.5%, and 13%, respectively. These parameters are used for the rest of the macrobenchmark experiments.

5 Macrobenchmark validation

In this section, we attempt to quantify two important properties of any experimental simulation framework: *accuracy* and *stability.* To quantify accuracy, we execute ten of the SPEC2000 benchmarks on the DS-10L workstation, and compare the resultant performance against the performance of a number of simulators configured like the DS-10L. We measure how the addition and removal of ten

	gzip	vpr	gcc	parser	eon	twolf	mesa	art	equake	lucas	mean
Alpha 21264 IPC	1.53	1.02	1.04	1.18	1.21	1.10	1.57	0.48	1.02	1.57	1.05
sim-alpha IPC	1.28	0.99	0.90	0.97	1.21	1.07	1.17	0.82	0.94	1.37	1.05
% error	-22.01	-4.63	-18.07	-23.09	-0.92	-6.07	-38.37	43.04	-10.94	-14.74	18.19
sim-stripped IPC	1.07	0.74	0.84	0.89	0.96	0.84	1.04	0.82	0.83	1.44	0.92
% difference	-51.52	-44.12	-42.33	-42.01	-34.10	-42.09	-62.10	39.75	-32.71	-9.96	40.07
sim-outorder IPC	2.28	1.62	1.89	2.00	2.08	1.76	2.59	2.14	1.69	1.79	1.95
% difference	28.56	34.04	37.20	37.05	38.29	32.25	36.80	76.89	34.60	11.54	36.72

Table 3: Macrobenchmark validation

different microarchitectural features affects performance, to determine what should be simulated to achieve a given level of accuracy. A microarchitectural optimization is stable if it results in similar performance improvements across multiple platforms. To quantify stability, we measure how the deltas in performance caused by optimizations change over a range of simulator configurations, as well as over a range of simulators.

5.1 Accuracy

In Table 3, we compare the performance of the DS-10L against three simulators. We used ten of the SPEC2000 benchmarks, which were compiled with the native Alpha compilers, using "-arch ev6 -non_shared -O4" for each benchmark. We ran all of the benchmarks to completion with the standard "test" input sets from the SPEC distribution. The three simulators that we compare are the following:

- **`sim-alpha`**: The validated DS-10L simulator. It is possible that the accuracy of this tool could be improved with further effort, but without more information about the processing core and board-level components, qualitative improvements in accuracy are unlikely.
- **`sim-stripped`**: A version of `sim-alpha` with many of the low-level features removed. We chose the level of detail to match what is typically seen in simulators in the architecture community: pipeline organization, functional unit latencies, etc., but few low-level limitations. We studied removal of the following seven performance optimizing features: (1) **addr**: an extra adder for quick computation of jump targets in the front end; (2) **eret**: early retirement of no-op instructions in the map stage; (3) **luse**: load-use speculation; (4) **pref**: instruction cache hardware prefetching; (5) **spec**: speculative update of the line and branch predictors; (6) **stwt**: the store-wait predictor; and (7) **vbuf**: the level-one data cache victim buffer. We also examined removing three performance-constraining features which are necessary for high clock rates, but reduce IPC: (1) **maps**: a three-cycle stall if the number of available physical registers drops below eight; (2) **slot**: no slotting restrictions in the pipeline; (3) **trap**: mbox traps, which flush the pipeline on MSHR conflicts and concurrent references to two blocks that map to the same place in the cache.
- **`sim-outorder`**: The out-of-order simulator from version 3.0b of the widely used SimpleScalar tool set [5]. The tools simulate a processor organization that would not be feasible at high frequencies and consequently have never been validated against hardware. Nevertheless, the tools have been used to produce results for hundreds of research publications. The simulator models a five-stage pipeline and is based on the Register Update Unit (RUU) [20], which combines the physical register file, reorder buffer, and issue window into a single structure.

The first row of Table 3 shows the instructions per cycle for the actual DS-10L hardware, measured with the DCPI tool. Note that the actual machine never achieves more than 1.6 IPC. The second and third rows show the IPC and the percentage error for `sim-alpha`, compared to the DS-10L. The next two rows show the IPC for `sim-stripped` and the percentage error compared to the native DS-10L. The last two rows show the same data (IPC and percent difference from the native hardware) for `sim-outorder`. Aggregate IPCs are computed using the harmonic mean, and aggregate error is the arithmetic mean of the absolute value of the percent error across the benchmarks.

In most cases, `sim-alpha` underestimates the performance of the native DS-10L, with a maximum negative error of -38.4% for *mesa*, which has an unusually high L2 cache miss rate (43%). Possible sources of this error include page coloring or memory controller optimizations to increase page hits, which are not modeled in the simulator. On only one benchmark (*art*) does `sim-alpha` perform better than the native hardware, by a large margin of 43.0%. That gap is partially due to a larger number of replay traps occurring in the native machine. The DS-10L running *art* incurs 52 million traps, whereas `sim-alpha` incurs only 43 million replay traps over 1.4 billion instructions of execution.

Although those two benchmarks show high error, the aggregate error is much lower, at 18%. The benchmarks

	ref	addr	eret	luse	pref	spec	stwt	vbuf	maps	slot	trap
sim-alpha IPC	1.05	0.98	1.10	0.99	1.05	0.99	1.00	1.05	1.07	1.05	1.05
% change		-7.78	-0.67	-5.79	-0.29	-5.92	-4.25	-0.37	2.11	0.36	0.31
std. deviation		5.81	1.09	2.52	1.27	5.07	5.60	1.07	2.85	1.64	0.99

Table 4: Effects of low-level features on performance

that tend to be cache-resident have the lowest errors, such as *vpr*, *eon*, and *twolf* (-4.6%, -0.9%, and -6.1%, respectively). The prevalence of benchmarks that show negative errors, for which the simulator performs worse than the hardware, indicates that our memory system simulations are overly conservative, and there are likely optimizations and performance tuning that we are not modeling or capturing. The harmonic mean of the native IPCs and those for `sim-alpha` are identical at 1.05, which is coincidence, given the variance in IPC across individual benchmarks.

The `sim-stripped` simulator shows consistently lower performance than `sim-alpha` for all but one benchmark (*lucas*). The mean error compared to `sim-alpha` is 40%, with individual errors as great as -60%. In fact, except for *lucas*, all benchmarks perform at least 30% worse than does the actual workstation. The degradation in performance is due to the negative effect of the removal of the performance-enhancing features, such as I-cache prefetching and load-use speculation, outweighing the positive effects of the removal of the performance constraining features.

We configured `sim-outorder` to match the 21264 as closely as possible: the memory system parameters include similarly configured caches (the 3-cycle L1 data cache hit latency, and 62-cycle latency for DRAM), the combined LSQ is set to 64 entries (instead of the 21264's split 32-entry load and store queues), and the RUU uses 64 entries. The 2-level adaptive branch predictor along with the BTB contains a similar quantity of state to the Alpha's tournament and line predictors. Interestingly, the performance of `sim-outorder` is entirely in the opposite direction from `sim-stripped`, outperforming the DS-10L by at least 28% for every benchmark except *lucas*, and by 36.7% on average. This optimistic level of performance results from the lack of cycle-time constraints on the `sim-outorder` core, including a shallower pipeline, no partitioning of the execution core, less contention in the simulated memory system, a more accurate target prediction (BTB instead of a line predictor), no replay traps, and fewer pipeline flushes. Consequently, great care must be taken when using these simulators to predict performance. Assuming a high clock rate—while not accounting for the constraints on the microarchitecture—can lead to substantially misleading conclusions.

5.2 Effect of individual features on accuracy

In Table 4, we show the effects that each of the ten individual features from the previous subsection have on overall performance. The **ref** column corresponds to `sim-alpha` with all of the features while the rest of the columns represent `sim-alpha` minus only the feature listed in the column heading. In the first row, we list the harmonic mean of the macrobenchmark IPC values for each configuration. In the second row, we show the mean percent change in performance compared to `sim-alpha`, which results from removing each feature. The third row displays the standard deviation of the changes in performance across the benchmarks for each configuration.

Performance drops significantly when any of four particular features are disabled, as they each independently provide more than 4% in performance to `sim-alpha`. These features are the jump adder (7.8%), load-use speculation (5.8%), speculative predictor update (5.9%), and store-wait bits (4.3%). Of the performance-constraining features, the only one that affects performance more than 1% is map-stage stalling. When those stalls are removed from `sim-alpha`, performance increases by 2.1%. Finally, we note that the variability is high: all of the standard deviations—which represent the degree to which the percentage improvements of the optimizations vary across the benchmarks—are greater than one percent. The standard deviation for the jump adder, speculative update, and store-wait bits is particularly high, more than 5% in each case.

5.3 Stability

When a feature or idea is evaluated on a simulator, that feature is *stable* if it provides similar benefits or improvements across other simulators and environments. Detailed, validated simulators may be unnecessary if new features are stable across more abstract, yet unvalidated, simulators. Conversely, an added feature that is unstable may appear to provide benefits on an inaccurate simulator, while on a validated simulator, the benefits might disappear or even reverse. In this subsection, we measure the stability of four different parameter sets across a range of multiple simulator configurations.

In Table 5, we show the change in performance when three improvements are made: reducing the L1 D-cache access latency from three cycles to one, increasing the L1 D-cache size from 64KB to 128KB, and doubling the number of physical registers. Each column displays the

Optimization	sim-alpha	addr	eret	luse	pref	spec	stwt	vbuf	maps	slot	trap	sim-strip.	sim-out
3 to 1-cycle L1 D$	5.53	5.45	5.98	n/a	6.25	5.45	6.49	6.42	5.90	5.25	5.95	9.85	5.78
64KB to 128KB L1 D$	2.04	1.72	2.03	1.70	2.23	1.96	2.43	2.14	2.02	1.55	1.38	1.70	0.66
40 to 80 physical regs.	0.63	0.91	0.53	0.63	0.98	1.07	1.44	0.55	0.88	1.27	0.95	0.64	0.23

Table 5: Simulator stability (% improvement)

respective improvements for one simulator configuration. The left-most column represents `sim-alpha`. The next ten columns contain the same configurations as discussed in Table 4, namely `sim-alpha` with one feature disabled per column. The right-most two columns hold the results for `sim-stripped` and a modified `sim-outorder`, in which the physical register file is a separate structure [1].

The eleven `sim-alpha` configurations are stable, with about one percentage point of variation among them across each row. When we varied configurations with `sim-stripped` (adding one feature at a time), we noticed only slightly larger swings. However, the stability across the simulator configurations is less consistent; the cache latency reduction benefits `sim-stripped` nearly twice as much as the other simulators. The optimizations improve `sim-outorder` less than for the other configurations, likely because its base performance is higher.

We compare stability across a different pair of simulators in Figure 2. A recent study measured the performance effects of multi-cycle register file delays, with and without complete bypassing [7]. That study used an in-house, 8-way issue simulator to simulate the SPEC95 benchmarks for different register file configurations. We configured `sim-alpha` similarly, making sure to balance each pipeline stage to avoid obvious bottlenecks. The three bars in each cluster represent the three register file configurations. The heights of the bars correspond to the measured IPC in the previous study, while the dark lower portions of each bar correspond to the IPC measurements, using `sim-alpha`, for the same experiments.

The performance loss seen by the incomplete bypass results led the authors to explore multi-banked and hierarchical register file organizations. However, in our simulations, the performance loss that motivated their work does not exist; the Alpha microarchitecture is limited by other overheads, and has sufficiently aggressive scheduling to overcome the effects of one-cycle bubbles between dependent instructions. Furthermore, while the two studies are modeling similar target architectures they produce strikingly different absolute performance results. From two different simulators that have similar configurations, one would draw quite different conclusions.

This example is meant to be illustrative only, as the hierarchical register file is a useful solution for more restricted register files. In `sim-alpha`, we saw the same sort of performance losses that the authors report for register files with greater than two-cycle access times, or no bypassing at two cycles. However, it is clear that a common reference point is needed to ensure that researchers running similar experiments draw the same conclusions.

6 Related work

The effort most similar to our own was the study performed by Black and Shen [2], which validated a performance model of a PowerPC 604 microprocessor. Their validation efforts benefited from the performance counters on the 604, which allowed them to track individual instructions, pairs, and combinations through the pipeline and compare the cycles consumed to those in their performance simulator. Since DCPI can measure only a few events in addition to cycle and instruction commit counts, we were restricted to running assembly tests for numerous iterations to isolate the behavior of instruction combinations. While the 604 validation study achieved low (4% mean) errors, it assumed a perfect L2 cache and could only measure performance of small, cache-resident benchmarks. Our work adds to their efforts by modeling a more complex microarchitecture, which contains seven full predictors, running memory-intensive benchmarks in addition to our kernels, and isolating the performance contributions of distinct microarchitectural features.

Gibson, *et al.* [9] described a validation of the FLASH multiprocessor hardware against two software processor simulators (Mipsy and MXS) coupled with two internal memory system simulators (Flashlite and NUMA), and using SimOS to model OS performance effects. The authors' validation focused more on the memory system than on the microarchitecture, as Mipsy does not model pipelines, while MXS models a generic pipeline, rather than the R10000 that was in their machine. The authors found that TLB behavior had a surprisingly substantial effect on performance and point out that OS page coloring can reduce cache misses. As we describe in Section 4, `sim-alpha` does not account for the TLB overheads correctly, nor does it model any effects of page coloring.

Reilly and Edmonson's work on a performance model for the Alpha 21264 was intended to enable quick exploration of the design space, rather than model the microarchitecture in detail [18]. Finally, Bose and Conte discuss performance evaluation from a design perspective and suggest the use of microbenchmarks, as well as the com-

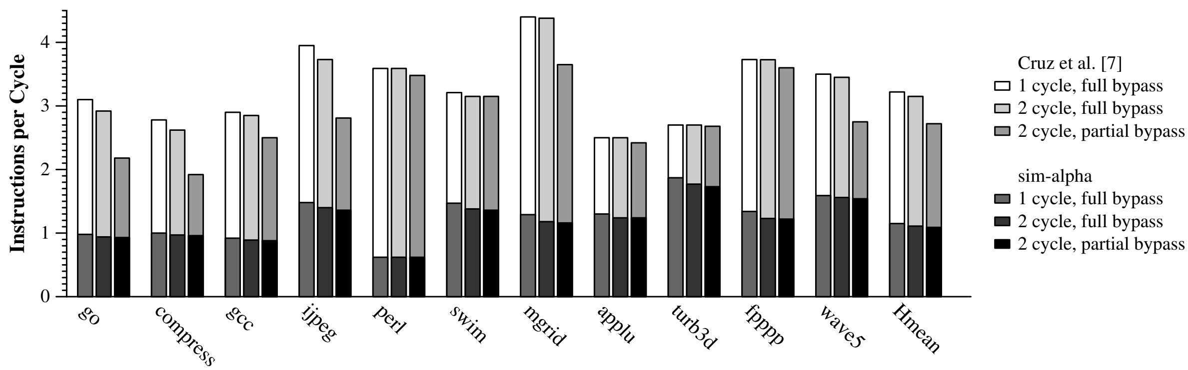

Figure 2. Register file sensitivity

parison of simulation data with hardware sampled event counts to detect performance bugs [3].

7 Conclusions

Because the architecture research community relies so heavily on simulation, it is disconcerting to think that our simulators may be highly inaccurate due to abstraction, specification, or modeling errors. Many of the studies published in our conferences and journals may report results that are unintentionally erroneous. Architecture ideas may show promising results merely because of simulator pipeline artifacts, performance bugs, or unrealistic baselines.

It is also possible that our unvalidated simulators are sufficiently accurate, errors balance out, and we can trust the results we obtain. Injecting new ideas into the literature may be more important than a quantitative evaluation to the second or third decimal place of precision. In that case, the community (including the authors of this study) could certainly benefit from fewer late nights spent producing gigabytes of simulation data.

The answer to the question of simulator validation depends on the conventionality of the research approach. A fundamentally new computer organization or technology can not be evaluated in the context of a conventional superscalar processor. Furthermore, no baseline exists against which the simulation of a radically new idea can be compared. However, innovations that provide incremental modifications to a conventional pipeline certainly can be evaluated within a conventional context. The more conventional the framework—and the smaller the performance gain—the greater the onus on the researchers to verify that the experimental framework, and thus their conclusions, are valid.

In this paper, we describe an attempt to verify a high-level timing simulator against actual hardware, the Compaq DS-10L workstation. Our goal was to measure and understand the error that researchers (including ourselves) incur by assumptions that we make in our simulation infrastructures. We measured the performance impact of many features the Alpha 21264 incorporated to permit fast clocks: instruction slotting, coupled line, way, and branch prediction, load-use speculation, deep pipelines, and early branch resolution. Our microbenchmark errors are small, on average less than 2%, but the large number of unknowns in the external memory system, despite our efforts to obtain documentation, made our efforts to obtain similar accuracy across our macrobenchmark suite unsuccessful, resulting in an 18% average error. Of the features we measured, we determined that early jump address calculation, load-use speculation, speculative updating of predictors, and store-wait prediction had the largest impact on performance and should be considered for incorporation by experimental architecture researchers.

By comparing `sim-alpha` against a stripped-down version of itself (`sim-stripped`), and then against `sim-outorder`, we were able to draw some broad conclusions about simulator implementation. First, simulators that model abstract machines are likely to outperform validated simulators that model real targets. The lack of implementation constraints, particularly in the high-frequency world of modern systems, makes an abstract target too optimistic. The SimpleScalar tools, which incorporate a shorter pipeline than the 21264, no replay traps, a centralized execution core, and a frequency-unconstrained front end, consistently outperformed the native Alpha by about a third—a mean of 36.7%.

Second, non-validated simulators that model real targets are likely to underestimate actual performance. They may implement all of the penalties that modern designs incur (clustering, deep pipelines, long memory latencies, etc.) but may not include all of the careful performance tuning microarchitecture structures that mitigate the penalties. The `sim-stripped` simulator consistently underestimated 21264 performance, with a mean error of -40%.

We conclude with four recommendations that, if followed, will improve the scientific rigor of much of the research performed in our community.

- **Reproducibility**: Research studies should provide sufficient detail for others to reproduce their results, through some combination of making the simulator code available or publishing all of the pertinent details in a paper or a technical report. We predict that the results of this paper will be reproducible because we have a specific reference machine, have provided substantial detail about what is not modeled in the simulator, and have made the code publicly available.
- **Consistent parameters**: Currently, many studies choose parameters, such as DRAM latencies, in an ad-hoc manner. That variance makes comparing results across papers difficult or impossible. Simulation parameters should be chosen against either reference machines, common models, or communal parameter sets, to maximize consistency across research studies. In this study, we used the DS-10L workstation as a source for our parameter choices.
- **Common baselines**: Studies of similar machines result in drastically varied performance. In the ISCA-27 proceedings, five different studies reported IPCs of the SPEC95 *gcc* benchmark that were evenly distributed from 0.9 to 3.5. Research studies should compare their results to a known baseline, such as an actual machine or standardized public simulator, so that the reported results are not wholly self-consistent.
- **Quantified stability**: To ensure that an optimization is widely effective, and is not merely a fortunate by-product of coincidence, it should be measured across a range of processor and system organizations. Achieving better reproducibility would help this goal, as stability could be measured across multiple research groups and simulation environments.

For many of the reasons highlighted above, results currently produced by architecture researchers are rarely used by practitioners. The ideas are frequently re-evaluated with an attempt to reproduce and refine the results in a company's internal environment. Improved accuracy, reproducibility, and consistency would greatly improve the utility of the results we generate for both practitioners and other researchers.

Acknowledgments

Thanks to Joel Emer and Steve Root for answering many questions about the Alpha microarchitecture. We thank Todd Austin, for providing both the Alpha ISA semantics file and all of the SimpleScalar wrapper code that we used, Alain Kägi for his comments and insights, as well as Bruce Jacob and Vinod Cuppu for providing their SDRAM code. This work was supported by the NSF CADRE program, grant no. EIA-9975286, and by equipment grants from IBM and Intel.

References

[1] Vikas Agarwal, M. S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Clock rate versus IPC: The end of the road for conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[2] Bryan Black and John Paul Shen. Calibration of microprocessor performance models. *Computer*, 31(5):59–65, May 1998.

[3] P. Bose and T. Conte. Performance analysis and its impact on design. *Computer*, 31(5):41–49, May 1998.

[4] Doug Burger and Todd M. Austin. The simplescalar tool set version 2.0. Technical Report 1342, Department of Computer Sciences, University of Wisconsin-Madison, June 1997.

[5] Compaq Computer Corporation. *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.

[6] Compaq Computer Corporation. *Compiler Writer's Guide for the Alpha 21264*, 1999.

[7] José-Lorenzo Cruz, Antonio González, Mateo Valero, and Nigel P. Topham. Multiple-banked register file architectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 316–325, June 2000.

[8] Vinodh Cuppu, Bruce Jacob, Brian Davis, and Trevor Mudge. A performance comparison of contemporary DRAM architectures. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 222–233, May 1999.

[9] Jeff Gibson, Robert Kunz, David Ofelt, Mark Horowitz, John Hennessy, and Mark Heinrich. Flash vs. (simulated) Flash: Closing the simulation loop. In *Proceedings of the 9th International Symposium on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[10] N. P. Jouppi and S. J. E. Wilton. Tradeoffs in two-level on-chip caching. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, April 1994.

[11] R. Kessler. The Alpha 21264 microprocessor. *IEEE micro*, 19(2):24–36, March 1999.

[12] R. Kessler, E. McLellan, and D. Webb. The Alpha 21264 microprocessor architecture. In *Proceedings of International Conference on Computer Design*, pages 90–105, October 1998.

[13] David Kroft. Lockup-free instruction fetch/prefetch cache organization. In *Proceedings of the Eighth International Symposium on Computer Architecture*, pages 81–87, May 1981.

[14] P. S. Magnusson, F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenstrom, and B. Werner. Simics/sun4m: A virtual workstation. In *Proceedings of the Usenix Annual Technical Conference*, pages 119–130, June 1998.

[15] J. McCalpin. The stream benchmark site. http://www.cs.virginia.edu/stream/.

[16] L. McVoy and C. Staelin. Lmbench: Portable tools for performance analysis. In *Proceedings of the USENIX 1996 Annual Technical Conference*, pages 279–294, January 1996.

[17] V. Pai, P. Ranganathan, and S. Adve. RSim: A simulator for shared-memory multiprocessor and uniprocessor systems that exploit ILP. In *Proceedings of the 3rd Workshop on Computer Architecture Education*, 1997.

[18] Matt Reilly and John Edmondson. Performance simulation of an Alpha microprocessor. *Computer*, 31(5):50–58, May 1998.

[19] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The SimOS approach. In *IEEE Parallel and Distributed Technology*, 1995.

[20] Gurindar S. Sohi. Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. *IEEE Transactions on Computers*, 39(3):349–359, March 1990.