Are Cycle Accurate Simulations a Waste of Time?

Vincent M. Weaver and Sally A. McKee
Computer Systems Laboratory
Cornell University
{vince,sam}@csl.cornell.edu

Abstract

Cycle-accurate simulation methods are necessarily slow. This slowness is only acceptable if the simulation results can be shown to have smaller error than other, faster, methods of generating the same results.

We use the SESC simulator as a representative of cycle-accurate simulation. We configure it to match an actual SGI MIPS R12000 system as closely as possible. We run the SPEC CPU 2000 benchmarks and compare simulated results against actual hardware performance counters. We find that for CPI SESC diverges from actual hardware by 22.6% for integer benchmarks and 46.4% for floating point. We then use the Qemu DBI tool as the basis for a faster simulation environment. Qemu generates traces that are consumed by individual simulation processes that run concurrently on a CMP system. We find that for CPI the results diverge from actual hardware by -14.6% for integer benchmarks and 29.3% for floating point. These Qemu results are obtained an order of magnitude faster than those found with SESC (and other popular simulators popular in academia, many of which are much slower).

These results show that a re-evaluation of the tradeoffs of cycle-accurate simulations in the field of computer architecture research could have merit. Furthermore, using multiple methods to validate a tool or to investigate a proposed architecture is simple good science.

1 Background

"Cycle-accurate" simulators are one of the prevailing simulation tools in Computer Architecture research. Unfortunately, the results generated by academic [1] cycle-accurate simulators can be misleading due to unknown amounts of error. More importantly, similar results can be generated faster using dynamic binary instrumentation (DBI) based simulation techniques.

There are a number of problems with cycle-accurate simulators, in general:

- **Speed:** Simulators are slow, often multiple orders of magnitude slower than native execution. Many researchers commonly use "reduced-execution" methods to compensate, yet these methods can compound simulation error if not carefully applied. For instance, Yi et al. [19] find that various reduced execution methods can add large errors—never less than 5%.
- **Obscurity:** The simulation tools are rarely used outside the specialized field of Computer Architecture research. Since the simulators themselves are rarely used for anything except running a limited set of benchmarks, bugs can lurk in the code base for a long time, and many are possibly never noticed at all.
- **Code Forks:** Since few people are using the simulators at any given time, the code base quickly becomes unmaintained and fragmented among the groups using it. Bugs may be fixed at different times and at different institutions. The source codes diverge so much that when one paper claims it uses a particular simulator, that statement may have little meaning, since the code used differs so much from the mainline (so much so as to be unrecognizable).
- **Generalization:** Simulators are often highly configurable, since the authors often want to create a tool that can be used to model a multitude of different situations. The end result is that a single simulator can model all architectures, but it may model them all in an equally poor manner. Another problem is that the more configurable a simulator, the easier it is to configure it improperly, often in non-obvious ways. *This has been one of the biggest problems for these authors.*
- **Validation:** Most simulators are not validated against real hardware, and when they are, the results are rarely within 10% error, even after extensive effort has been taken to attempt to model a known architecture as

[1] Heretofore, when we mention cycle-accurate simulations, we refer to tools and results generated in academia. Industry researchers and developers have created much more accurate simulators, but since their source code is not generally available to academics, we will not discuss them here.

closely as possible [3, 10, 7]. There are exceptions, of course, but the most commonly used academic tools have diverged widely from any versions for which validation was attempted.

- **Documentation:** Simulators are often poorly documented, both at a high level and at the source-code level. This alone probably accounts for more errors in simulation than any overt programming bugs. Researchers simply do not have the information needed to use them correctly.
- **Obsolescence:** Most simulators are already outdated by the time they become mature enough to run useful workloads. It is difficult to gain sufficient documentation on modern processors to accurately implement internals, so, instead, well understood but obsolete processors are modeled.
- **Tools:** Simulators often require a special tool-chain to build suitable executables. The difficulty of using out-of-date toolchains (many need old versions of libraries that are no longer available, for instance) necessitates that researchers often use pre-compiled benchmarks that are rarely updated. New advancements in compiler technology are thus lost, since the toolchain is rarely complete enough to compile whole benchmark suites. Often some of the more interesting benchmarks are simply left out due to toolchain difficulties, This is yet another source of error in simulations [5].
- **Operating System:** Many simulators cannot model full operating systems, which can be problematic: the OS can have a significant impact on full system simulation. Cain et al. [4] find that removing the OS from the simulation equation can have a greater impact on results than ignoring effects of speculation.

These problems results in part from the lack of funding for building and maintaining solid academic architectural tools. One or two students cannot build and maintain a tool *and* use it for their doctoral research in a reasonable amount of time, given today's complicated architectures.

In short, to be blunt, using an invalidated or poorly documented simulator modeling a 10-year old processor by running only small portions of an 8-year old benchmark suite that was compiled with a 10-year old compiler may not represent the best way to do cutting-edge research. Taking this setup and scaling the configuration to match a hypothetical processor only tangentially related to the original design can compound the accuracy problem. Eventually it becomes critical to know how big the potential error is; a small average speedup of 5-10% (which is often sufficient for publication reviewers) might, in reality, be dwarfed by cumulative errors of the infrastructure[2].

[2]We do not discuss issues involved with averages chosen to represent simulation statistics, but see John Mashey's writing online or the MoBS 2007 workshop proceedings for more information on that subject.

2 Methodology

Our original goal was to evaluate the suitability of the SESC simulator for use by our group. We chose it because other groups are also using it, it provides much flexibility, it is an open-source tool (which we whole-heartedly support), it runs more quickly than many other simulators, and it seemed the best available candidate to meet our needs. As with any tool we use in our research, we attempt to validate that tool ourselves as best we can before using it for experimental studies. In this case, we run the same benchmarks on real hardware, the SESC simulator, and a custom Qemu-based DBI/trace-driven simulator, and then investigate the relative results and merits of each.

2.1 Cycle-accurate Simulator

SESC [14] is a widely used cycle-accurate simulator. SESC can simulate CMP systems, but for comparison purposes, we only model a single-core system. The simulator was originally built to model out-of-order MIPS processors, and it runs MIPS binaries. It uses an elaborate configuration file that can specify architectures very different than the originating design.

As far as we know, SESC has never been validated in a peer-reviewed manner. The documentation distributed with the simulator includes a file `README.validation` showing that a few microbenchmarks matched hardware execution time within about 20% for R10000 and R4400 MIPS-based machines.

We configure SESC to match our reference platform as closely as possible (with the help of the tool's original author), which turns out to be difficult, despite our machine's being almost exactly the same as the simulator's original design point. Major differences are that the R12000 has a unified 2-page, 64-entry software-controlled TLB (SESC apparently only handles separate data and instruction TLBs), and the R12000's off-chip L2 cache with a way-predictor (which can affect L2 cache latencies in a way not easily modeled with SESC). The branch predictor in the R12000 is deceptively non-trivial, and again it is not possible to model exactly. (Many of the arcane architectural details are not sufficiently documented for any simulator author to model exactly without "inside" industrial information.)

We make a best attempt to configure SESC properly; unfortunately the configuration format is poorly documented. Many necessary options are not described, the provided sample configurations are not helpful, and even the related source code is not well commented. In the end, after spending much time researching and crafting a configuration file, the author of SESC found 40 errors in our configuration. This does not bode well for configuration attempts by other new users of the tool. A copy of our configuration file is

Processor	300MHz R12000 out-of-order, 4-issue 33 arch registers 64 physical registers
Memory Subsystem	L1i: 32kB, 2-way, 64B L1d: 32kB, 2-way, 32B L2 : 2MB, 2-way, 128B 2GB SDRAM, 1.0GB/s
Branch Predictor	2048 entry 2-bit
TLB	Unified 64-entry

Table 1. Configuration of SGI Octane2 machine used for comparison

available for download.

We use a default version of SESC, checked out from the CVS server on 7 April 2008 and compiled with gcc version 4.2.4. We use the `-k0x800000 -h0x23400000 -p2` command line options when running benchmarks.

2.2 Reference Hardware

The reference platform for our experiments is an SGI Octane2 [17] with an R12000 MIPS processor [18, 12]. A summary of key features is listed in Table 1. The machine runs Linux 2.6.22 patched to provide Octane support. The kernel has been modified to include the perfmon2 [9] performance counter infrastructure.

The R12000 has a relatively unique feature in that the processor's branch prediction methods are configurable at runtime. We wrote a custom kernel module that sets the proper Branch Diagnostic Register bits (cp0 register 22) to change the branch prediction method on the fly. The processor defaults to a 2048-entry 2-bit saturating counter dynamic prediction scheme. This can be changed to various static schemes: always taken, always not-taken, and forward/taken-backward/not-taken. A global pattern history table can be enabled with a configurable number of bits, and the Branch Target Address Cache (BTAC) and Branch Return Cache (BRC) can be individually disabled.

We run various microbenchmarks to verify the performance counters work properly, using the `pfmon` tool to collect performance statistics. This tool enables performance counting from a separate process, so the actual counting is handled entirely by the OS kernel, inducing very little user-space overheard. Counts are collected in aggregate for the full program, with no sampling.

There has been concern about the accuracy of MIPS performance counters: Korn, Teller, and Castillo [11] find up to 25% error with some counters on the R12000 and R10000 under SGI IRIX. We do not notice similar error; potentially, the differences they see are due to their use of sim-outorder as a reference, which has been found to have similar levels of error by Desikan, Burger, Keckler and Austin [6, 7].

2.3 DBI-based Simulator

We use Qemu [1] to generate traces consumed by a set of small independent simulators. Qemu uses dynamic retranslation at the basic-block level to convert from one architecture (in this case MIPS) to another (in this case x86). We add code hooks to output needed trace data.

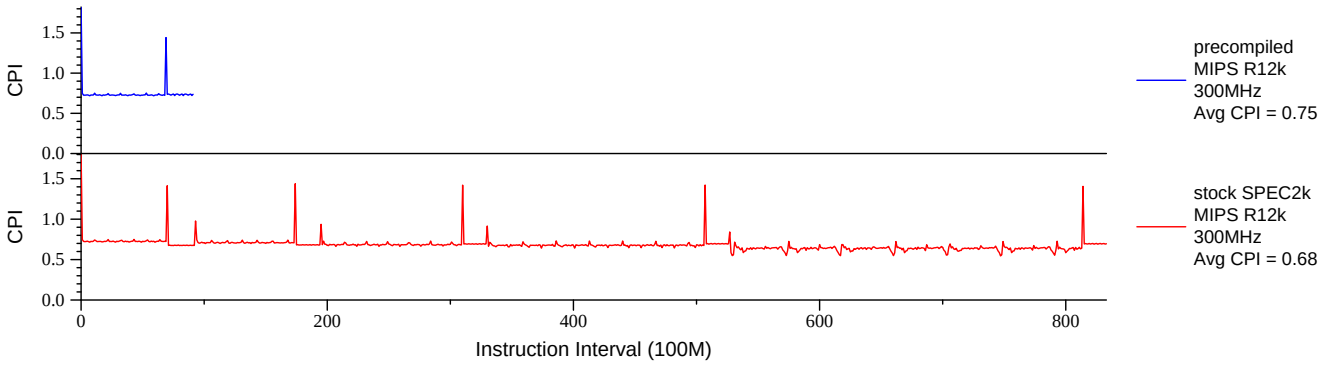
For cache simulation we use the Dinero IV [8] Cache Simulator. Qemu passes trace info in the Dinero file format over a named-pipe to Dinero (which runs in a separate process). To determine branch prediction information we wrote a custom branch predictor (source available on our website). This branch predictor runs in a separate process and obtains the full instruction stream (both address and instruction value) from Qemu over a named-pipe. The predictor decodes the MIPS instructions and determines which are branches (taking special care to handle the "predict taken" `beql` instructions properly). A branch is determined to be taken or not by buffering an additional two instructions to see if the address after the delay slot is PC+8.

Because each of our tools is a separate process, we can take advantage of CMP and SMP systems in a way that most cycle-accurate simulations cannot. Each process can live on its own core, and running the branch predictor thread at the same time as a cache thread adds negligible overhead on a 4-processor machine. The limiting factor while doing this simulation is the cache simulator, not the dynamic translation and execution of the binary.

2.4 Benchmarks

To evaluate the various simulation methods, we use the SPEC CPU 2000 [15] benchmarks. To enable comparison with past uses of the SESC simulator, we use the precompiled versions of the benchmarks provided on the SESC website. All three of our test platforms can run these benchmarks unmodified.

Unfortunately the pre-compiled benchmarks have some limitations. They are potentially not plain SPEC 2000 binaries, even though the included documentation does not mention this. Extra `printf()` commands have been scattered throughout the code (presumably for debugging purposes or for controlling partial simulation experiments), and some benchmarks have been modified for faster run times. For an example, see Figure 1, which shows that `gzip`—as provided—only executes a small fraction of the full benchmark. In addition to the above problems, not all of the CPU 2000 benchmarks are included with the precompiled binaries. For all experiments we run full reference input sets.



CPI at each 100M interval, 164.gzip, source input

Figure 1. The precompiled SPEC 2000 benchmarks available from the SESC website potentially have had modifications to reduce runtime. Above is a phase chart gathered with hardware performance counters showing the provided precompiled binary on the top and a binary compiled from original SPEC sources by us with gcc on the bottom.

Method	Fastest		Slowest		Average Slowdown
R12000	15s	(gzip.log)	57m23s	(swim)	–
QEMU	13m52s	(gzip.log)	1d20h20m47s	(sixtrack)	38x
SESC	2h17m38s	(gzip.log)	16d02h53m15s	(mgrid)	393x

Table 2. Comparison of simulation times

3 Evaluation

We run as many SPEC 2000 benchmarks as possible on the various platforms. Relative run times are shown in Table 2. For the simulated results, we run on a large cluster of 4-processor 3.46GHz Pentium Ds, each with 4GB of RAM.

3.1 Absolute Results

Figure 2 shows actual and predicted L1 instruction cache miss rates. The various tools calculate instruction cache misses in different ways. For the performance counter results these graphs show decoded instructions versus instruction cache misses; for SESC and Qemu the graphs show graduated instructions versus instruction cache misses. The number of instruction cache misses in the floating point case are so small that a small absolute error can cause a large percentage error. Qemu has problems with the `art` benchmarks, which we are investigating.

The reference system has write-back caches, which can introduce accuracy issues with the performance counters. Any memory accesses that happen while the benchmark process is not running can change the values in the cache. While all attempts are made to run the benchmarks on an otherwise quiet system, other processes and even the operating system can evict cache lines on the real system in ways that would not happen in the simulator. Similarly, values stored into cache may not be accounted for in the performance counters if the actual write-back to memory happens when in a different processor's context.

Qemu does not follow wrong-path execution, which can account for some of the differences from actual hardware. Likewise, SESC does not follow wrong-path execution, since that code path is out of date, and is thus disabled in the default configuration. Despite not executing these wrong-path instructions, the results are not far off; this shows that full cycle-accuracy is not always needed for generating good cache simulation results (and further supports the conclusions of Cain et al. [4] regarding OS impact versus speculation, at least in the case of Qemu).

Figure 3 shows L1 data cache miss rates, and Figure 4 shows L2 miss rates. The latter is important, since L2 cache misses all traverse the processor bus of a multiprocessor system. If the tool used records vastly incorrect numbers of misses, multiprocessor simulations will generate erroneous data that could influence a final design. SESC overall does poorly predicting L2 miss rates for floating point benchmarks, this could indicate that the floating point pipeline sections of the configuration file need further adjustment.

The R12000 has a complicated off-chip cache. In order to save pins, the machine incurs significant overhead in changing the cache way. To mitigate this, it uses a cache way-predictor, with a penalty on miss. None of the simulators model this aspect of the system, which can potentially become another source of modeling error.

Figure 5 shows branch predictor results. The R12000 can predict and fetch past up to four branches, so a large amount of speculation can be happening, and Qemu and SESC are not capable of detecting this. There are also many hardware subtleties to the R12000 branch predictor that neither Qemu or SESC are modeling.

CPI results are shown in Figure 6. Qemu has no real concept of time, so we approximate cycles with the formula: $cycles = \frac{I_g * L1_{ht}}{ifs} + DL1_a L1_{ht} + L1_m L1_{mt} + L2_m L2_{mt} + Br_m Br_{mt}$

where I_g is graduated instructions, $L1_{ht}$ is L1 hit time (2 cycles), ifs is the instruction fetch size (4), $DL1_a$ is L1 data accesses, $L1_m$ is L1 misses, $L1_{mt}$ is L1 miss time (14 cycles), $L2_m$ is L2 misses, $L2_{mt}$ is L2 miss time (120 cycles), Br_m is number of branch misses, and Br_{mt} is branch miss delay (2 cycles)

CPI is the metric most often used in validation, so it is important to have these values match extra hardware. There are a lot of architectural and software causes of cycle variation not modeled by either simulator. Most notably, no Operating System effects are modeled at all.

3.2 Relative Results

Many researchers suggest that absolute results are not as important with cycle-accurate simulation, but that relative results are what matter most. As long as the trends are consistent, then a simulator is still useful even if the simulator is unvalidated and the error is large. To investigate this claim, we configure our R12000 to operate in various branch predictor configurations (see Section 2.2 for more details). We then plot relative differences in the metrics to see if consistent trends are visible.

In Figure 7 we show the relative reduction in branch predictor miss rate by going from a dynamic 2-bit predictor to an always-taken predictor. The figure shows that trends are similar across all benchmarks, although the Qemu results are optimistic and the SESC results are pessimistic. Figure 8 shows the same results, compared against a static backward/taken forward/not-taken predictor.

Figure 9 shows how the always-taken predictor compares against the 2-bit one with regards to L2 cache misses. Qemu and SESC both do not model wrong-path execution, so they have identical memory accesses even with different branch predictors. Neither simulation method can predict the significant predictor-based changes in L2 behavior observed on actual hardware. The results with the forward/backward static predictor in Figure 10 are similar.

TLB behavior is shown in Figures 11 and 12. Results are not shown for Qemu because a trace-based TLB simulator was not available (and we didn't have +time to write our

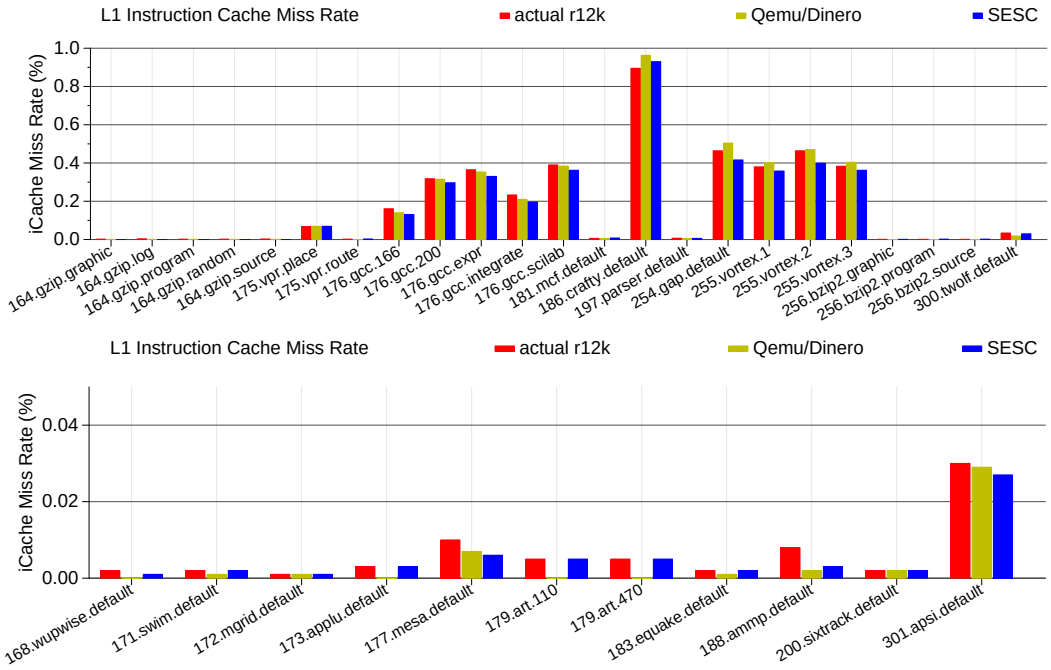


Figure 2. Instruction cache miss rate with integer benchmarks above and floating point below.

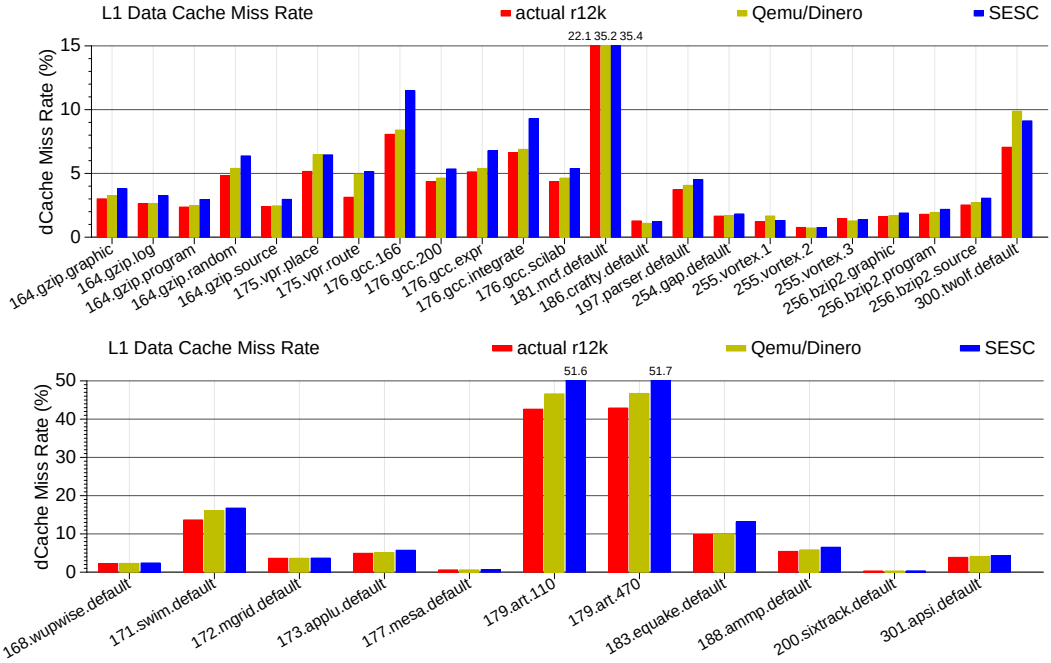


Figure 3. L1 data cache miss rate with integer benchmarks above and floating point below.

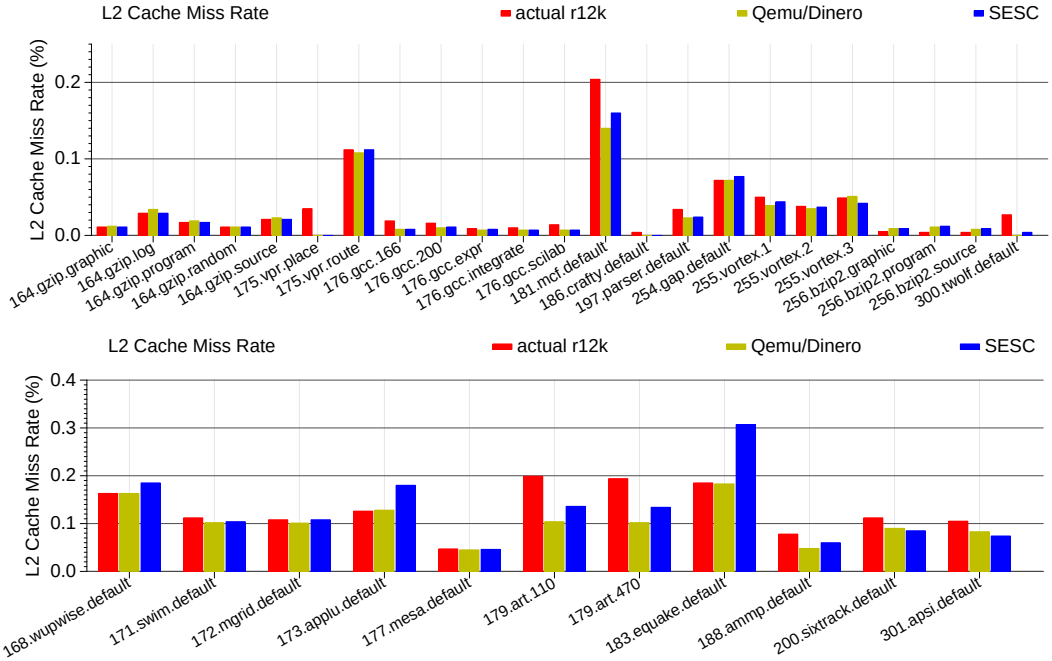


Figure 4. L2 cache miss rate with integer above and floating point below. None of the simulations capture `mcf`'s behavior well. None of the simulation methods predicts the `art` benchmarks well.

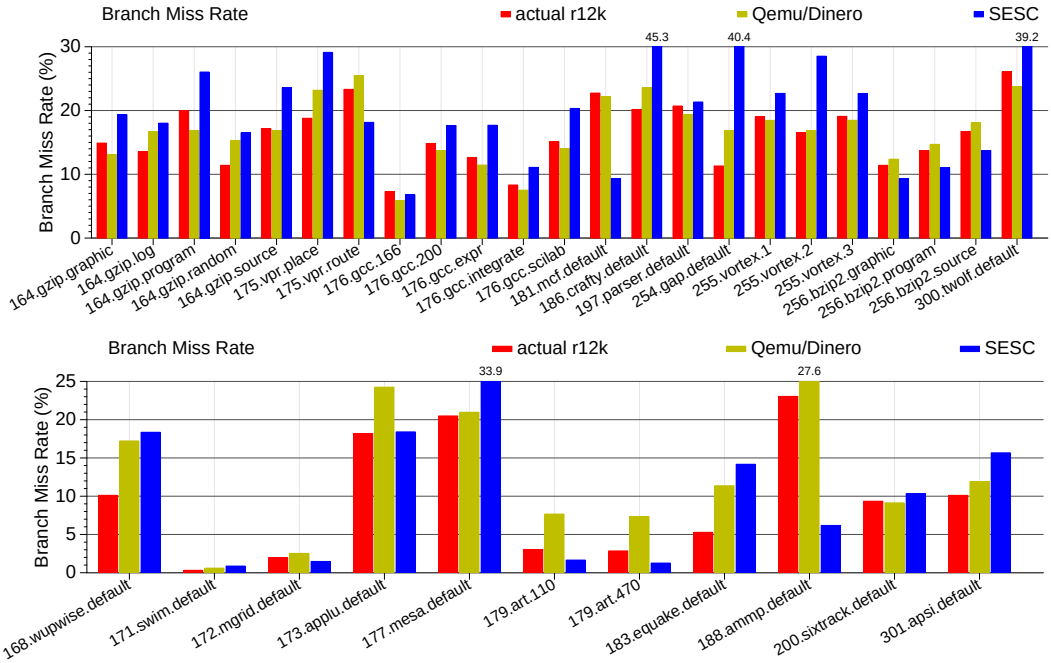


Figure 5. Branch miss rate with integer above and floating point below. The hardware can have up to four outstanding branches; Qemu and SESC do not model wrong-path execution.

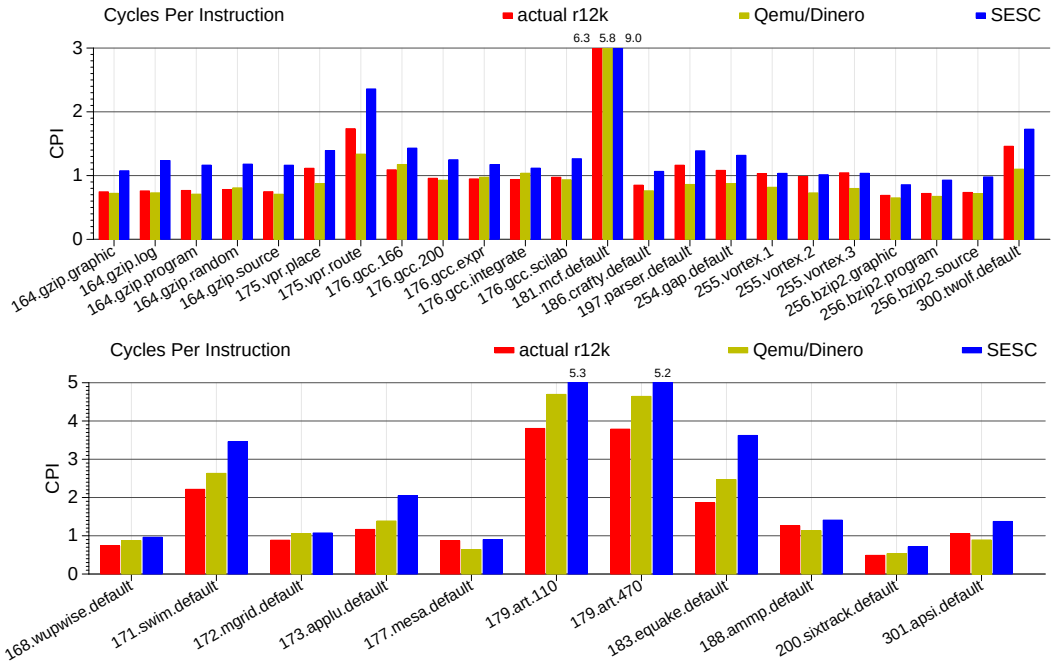


Figure 6. CPI results with integer above and floating point below.

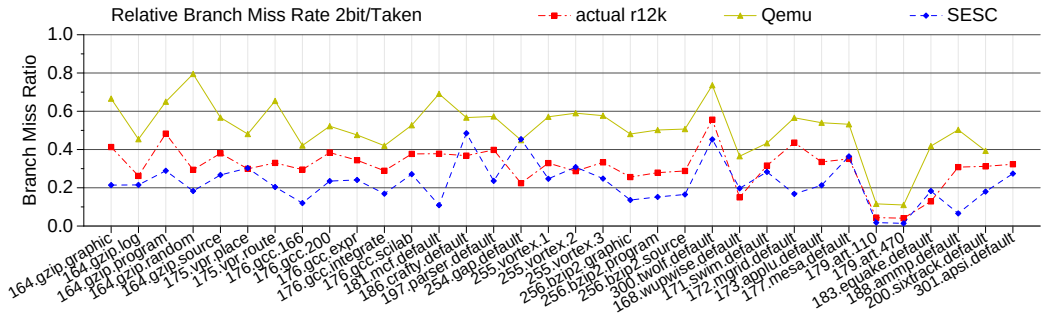


Figure 7. Always taken branch predictor normalized against dynamic 2-bit. The metric shown is branch predictor miss rate.

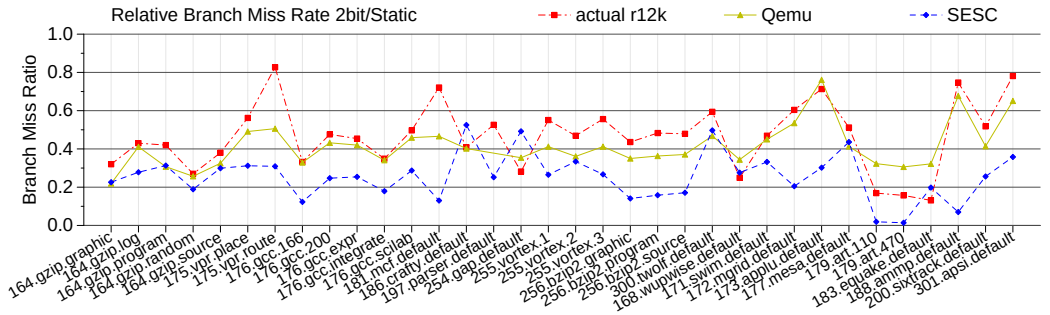


Figure 8. Static branch predictor normalized against dynamic 2-bit. The metric show is branch predictor miss rate.

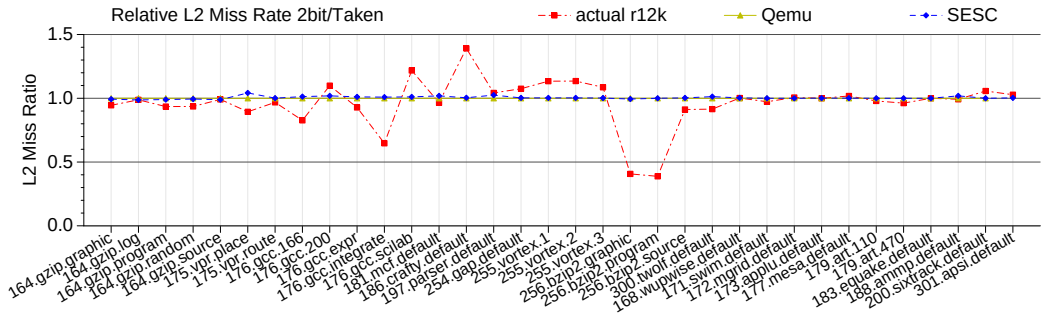


Figure 9. L2 cache miss rates with always-taken predictor, normalized against 2-bit results.

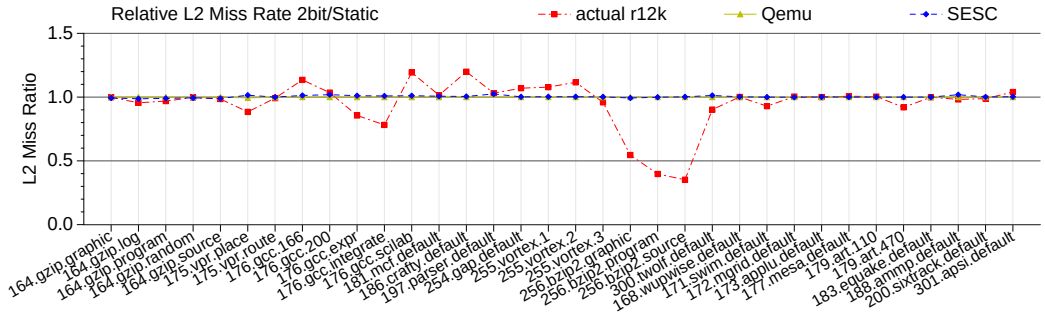


Figure 10. L2 cache miss rates with static predictor, normalized against 2-bit results.

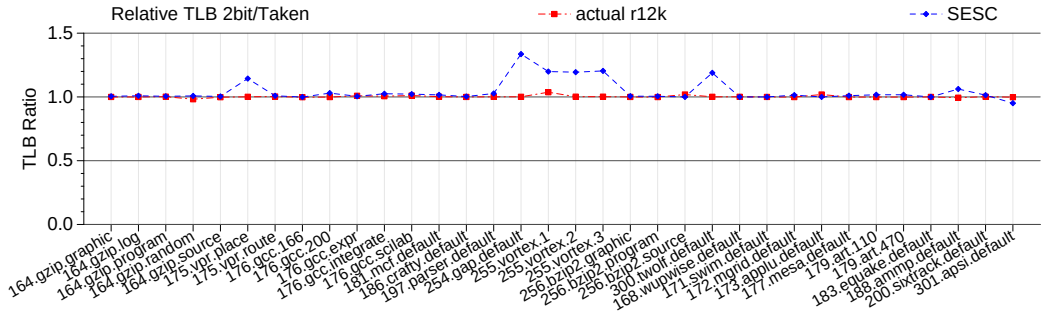


Figure 11. TLB misses with always taken normalized against 2-bit.

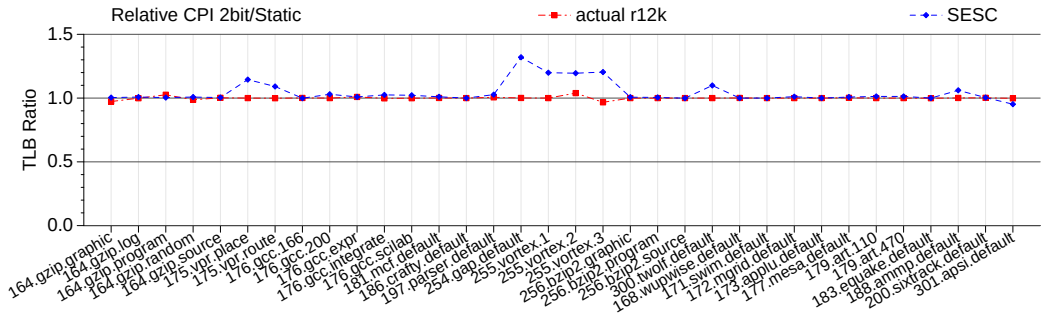


Figure 12. TLB misses with static predictor normalized against 2-bit.

own for the purposes of this article, alas). On actual hardware, the branch predictor seems to have minimal impact on TLB behavior. The MIPS TLB is managed in software, usually with random replacement. This means it is easy for results to diverge. Also, MIPS has a unified instruction/data TLB, which SESC cannot model.

Figure 13 and Figure 14 show the relative results for CPI. The Qemu results are close, despite the cycles count being based solely on cache and branch predictor miss rates.

4 Results

A summary of the absolute results are shown in Table 3. The weighted average is taken of the various metrics across all of the benchmarks that have finished running to completion on all three platforms. This is a total of 22 benchmarks (19 integer, 3 floating point) which unfortunately is only a portion of the 48 available SPEC CPU 2000 benchmark/input pairs.

We find that for the absolute results, SESC does not perform noticeably better than Qemu, despite taking an order of magnitude longer to run.

Table 4 shows the percent error of the average relative performance differences. The CPI results show that these methods can be used to predict performance with an average error of 15% on CPI. The L2 Cache results show that sometimes results can be deceptive; even though neither QEMU or SESC is capable of executing wrong-path execution, their results that do absolutely nothing still fall within 10% error for the relative L2 cache miss rate.

5 Related Work

The most similar work to ours is that of Gibson et al. [10]. They validate various MIPS simulators against their R10000-based FLASH system. They find that even their most carefully designed simulators have surprisingly large errors. They, like we, call into question the value of highly detailed simulators that are not validated against real hardware.

Black et al. [2, 3] create a model of the PowerPC 604 processor and validate it using hardware performance counters. They use a small set of benchmarks for validation, and try to reduce error. Interestingly, they find that fixing bugs in the simulator can actually increase the error in simulation because previous errors masked other bugs.

Desikan, Burger, Keckler and Austin [6, 7] validate the sim-alpha cycle-accurate simulator. They find that the generic sim-outorder simulator has upwards of 40% error, and even a fine-tuned attempting to match an actual Alpha machine still yields errors of around 15%. They run 22 of the SPEC CPU 2000 benchmarks. (This work also sparked much controversy in the architecture community.)

Vlaovic and Davidson develop TAXI [16], which uses a Bochs-based front end to generate traces that are fed to a cycle-accurate simulator modeling an earlier x86 machine. They attempt to validate this method using performance counters, and find their major limiting factor to be lack of documentation for the architecture they are trying to model.

Patil et al.'s work [13] on validating Itanium SimPoints uses a DBI tool (in this case Pin) and performance counters to measure CPI, and then measures performance on machines with differing configurations. Their purpose is not to validate any particular simulator, but rather to explore in depth the SimPoint methodology.

6 Conclusions and Future Work

We have shown that in certain situations cycle-accurate simulation is not inherently more accurate than using much faster DBI-based methods. When engaging in computer architecture research, if the metrics of interest are those that can be produced using DBI-methods then it would be a waste of time (possibly by orders of magnitude) to engage in cycle-accurate simulation.

SESC can model multi-processor and multi-core systems, something that is not currently possible with DBI based simulation. An important area of future work is to see if it is feasible to extend the Qemu simulation methodology to enable multi-core simulations, and see if the speed benefits still exist. A related issue with cycle-accurate simulations if the lack of scalability when running the simulators on multi-processor systems. Our Qemu-based infrastructure, by virtue of its modular nature, can take advantage of multiple processors in a system in a way that a monolithic simulator cannot.

More investigation needs to be done into Operating System effects in the simulated results. This requires a simulation environment that can run a full operating system, which Qemu can do but SESC cannot.

Our results are for commonly used metrics; there will always be a number of low-level metrics that only detailed cycle-accurate simulation can provide. Often investigating these metrics properly require intrusive changes to the cycle-accurate simulator. In the end it might be better to instead write a standalone micro-simulator of just the piece of micro-architecture being investigated. Future work would be to explore the feasibility of using DBI to run these micro-simulators, and to determine if a full cycle-accurate interaction is needed in these situations.

Our relative results show predicted machine performance, but only for changes in branch predictor. It is possible these results do not hold for other kinds of architectural changes. It would be informative to vary the architecture in

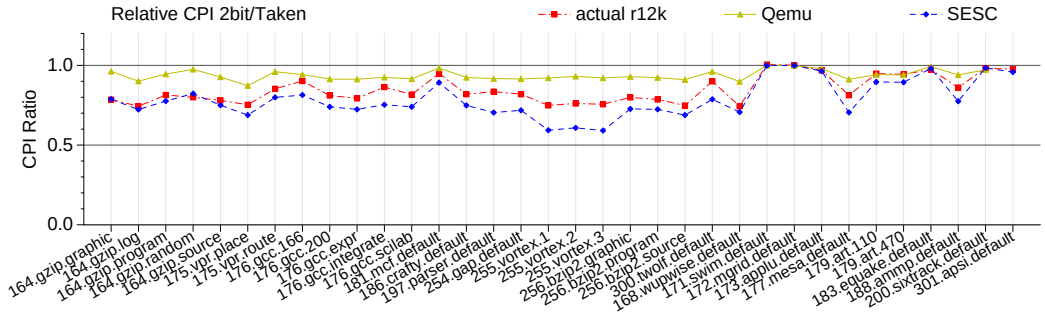


Figure 13. CPI with always taken normalized against 2-bit.

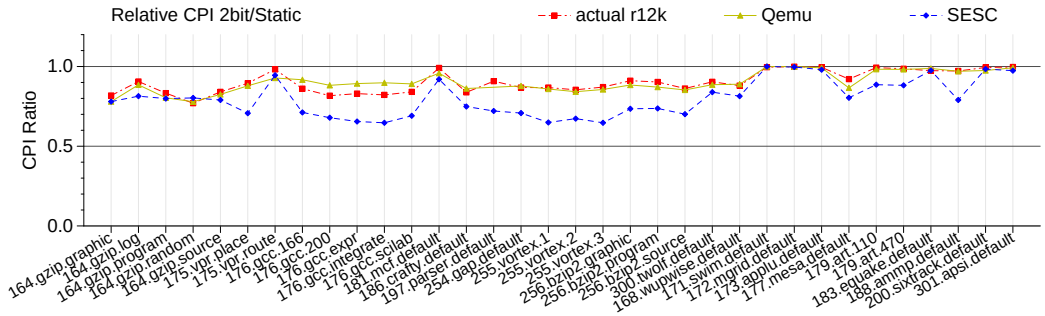


Figure 14. CPI with static predictor normalized against 2-bit.

Metric	Bench Type	R12000 Weighted Average	Qemu Weighted Average	Qemu % Error	SESC Weighted Average	SESC % Error
L1I$ Miss Rate	Int	0.233%	0.334%	43.5%	0.248%	6.4%
	FP	0.008%	0.001%	-83.9%	0.006%	-23.9%
L1D$ Miss Rate	Int	3.928%	4.260%	8.5%	4.726%	20.3%
	FP	5.230%	6.406%	22.5%	6.485%	24.0%
L2$ Miss Rate	Int	0.058%	0.051%	-11.9%	0.042%	-27.6%
	FP	0.127%	0.107%	-16.2%	0.128%	0.4%
BrPred Miss Rate	Int	18.9%	18.4%	-2.7%	27.0%	42.9%
	FP	12.7%	18.2%	43.2%	15.0%	18.4%
CPI	Int	1.20	1.03	-14.6%	1.47	22.6%
	FP	1.09	1.41	29.3%	1.60	46.4%

Table 3. Summary of results. The weighted average is across all of the SPEC 2000 benchmarks which ran to completion on all three platforms: 23 integer and 11 floating point (this is unfortunately only a portion of the 48 available benchmark/input combinations).

Metric	Brpred Type	Qemu % Error	SESC % Error
BrPred Miss Rate	Taken	64.1%	-28.0%
	Static	-11.0%	-44.9%
L2$ Miss Rate	Taken	5.6%	6.1%
	Static	7.1%	7.4%
CPI	Taken	11.5%	-7.1%
	Static	0.1%	-10.9%

Table 4. Summary of relative results. The relative results compare the relative results when moving from 2-bit branch predictor to either taken or static. The error shown is the relative error between the relative average means of all benchmarks on actual hardware versus the predicted relative average means of the simulated results. The results represent the 33 of the SPEC CPU 2000 benchmarks which ran to completion on all three platforms.

other ways and see how the results change, though validating this is difficult because it requires actual hardware with the desired architectural parameters.

Overall, we find that simulators are complicated tools that must be used with caution. Despite the availability of faster methods of obtaining results, cycle-accurate simulators remain indispensable as tools in computer architecture research. We hope we have raised awareness of the inherent benefits and limitations in the use of these tools.

References

[1] F. Bellard. QEMU, a fast and portable dynamic translator. In *Proc. 2005 USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, Apr. 2005.

[2] B. Black, A. Huang, M. Lipasti, and J. Shen. Can trace-driven simulators accurately predict superscalar performance? In *Proc. IEEE International Conference on Computer Design*, pages 478–485, Oct. 1996.

[3] B. Black and J. P. Shen. Calibration of microprocessor performance models. *IEEE Computer*, 31(5):59–65, May 1998.

[4] H. Cain, K. Lepak, B. Schwartz, and M. Lipasti. Precise and accurate processor simulation. In *Workshop on Computer Architecture Evaluation Using Commercial Workloads*, pages 13–22, Feb. 2002.

[5] D. Citron. MisSPECulation: Partial and misleading use of SPEC CPU2000 in computer architecture conferences. In *Proc. 30th IEEE/ACM International Symposium on Computer Architecture*, pages 52–62, June 2003.

[6] R. Desikan, D. Burger, and S. Keckler. Measuring experimental error in multiprocessor simulation. In *Proc. 28th IEEE/ACM International Symposium on Computer Architecture*, pages 266–277, June 2001.

[7] R. Desikan, D. Burger, S. Keckler, and T. Austin. Sim-alpha: a validated, execution-driven Alpha 21264 simulator. Technical Report TR-01-23, Department of Computer Sciences, The University of Texas at Austin, 2001.

[8] J. Edler and M. D. Hill. Dinero IV trace-driven uniprocessor cache simulator. `http://www.cs.wisc.edu/ markhill/DineroIV`, 2003.

[9] S. Eranian. Perfmon2: a flexible performance monitoring interface for Linux. In *Proc. of the 2006 Ottawa Linux Symposium*, pages 269–288, July 2006.

[10] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich. FLASH vs. (simulated) FLASH: Closing the simulation loop. In *Proc. 9th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 49–58, Nov. 2000.

[11] W. Korn, P. J. Teller, and G. Castillo. Just how accurate are performance counters? In *20th IEEE International Performance, Computing, and Communication Conference*, pages 303–310, Apr. 2001.

[12] NEC. *VR10000 Series 64-/32-bit Microprocessor User's Manual*, 2001.

[13] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. In *Proc. IEEE/ACM 37th Annual International Symposium on Microarchitecture*, pages 81–93, Dec. 2004.

[14] J. Renau. SESC. `http://sesc.sourceforge.net/index.html`, 2002.

[15] Standard Performance Evaluation Corporation. SPEC CPU benchmark suite. `http://www.specbench.org/osg/cpu2000/`, 2000.

[16] S. Vlaovic and E. Davidson. TAXI: Trace analysis for X86 interpretation. In *Proc. IEEE International Conference on Computer Design*, pages 508–514, Sept. 2002.

[17] I. Williams. An illustration of the benefits of the MIPS R12000 microprocessor and OCTANE system architecture. White Paper, SGI, 1999.

[18] K. C. Yeager. The Mips R12000 superscalar microprocessor. White Paper, SGI, 2000.

[19] J. Yi, S. Kodakara, R. Sendag, D. Lilja, and D. Hawkins. Characterizing and comparing prevailing simulation techniques. In *Proc. 11th IEEE Symposium on High Performance Computer Architecture*, pages 266–277, Feb. 2005.