Variability in Architectural Simulations of Multi-threaded Workloads

Alaa R. Alameldeen and David A. Wood
Computer Sciences Department, University of Wisconsin-Madison
{alaa, david}@cs.wisc.edu

Abstract

Multi-threaded commercial workloads implement many important internet services. Consequently, these workloads are increasingly used to evaluate the performance of uniprocessor and multiprocessor system designs. This paper identifies performance variability as a potentially major challenge for architectural simulation studies using these workloads. Variability refers to the differences between multiple estimates of a workload's performance. Time variability occurs when a workload exhibits different characteristics during different phases of a single run. Space variability occurs when small variations in timing cause runs starting from the same initial condition to follow widely different execution paths.

Variability is a well-known phenomenon in real systems, but is nearly universally ignored in simulation experiments. In a central result of this paper, we show that variability in multi-threaded commercial workloads can lead to incorrect architectural conclusions (e.g., 31% of the time in one experiment). We propose a methodology, based on multiple simulations and standard statistical techniques, to compensate for variability. Our methodology greatly reduces the probability of reaching incorrect conclusions, while enabling simulations to finish within reasonable time limits.

1. Introduction

Multi-threaded, throughput-oriented applications—such as databases and web servers—represent a dominant class of internet service workloads. Current and future computer architectures (e.g., multi-threaded processors [4] and chip multiprocessors [3, 4]) are increasingly designed with these applications in mind. Standardized multi-threaded benchmarks [34, 35] are commonly used to evaluate uniprocessor and multiprocessor systems, using both measurement of current systems and simulation of future ones. Execution-driven evaluation of these workloads requires full-system simulation, since they spend a significant portion of their time in the operating system [1, 2, 26].

This paper identifies performance variability as a potentially major challenge for architectural simulation studies using multi-threaded workloads. Variability refers to the differences between performance estimates obtained from multiple runs of the same workload. Time variability occurs when a workload exhibits different performance characteristics during different phases of a single run. Space variability occurs when small timing variations cause different runs starting from the same initial conditions to follow widely different execution paths from the space of all possible paths.

If unaddressed, both types of variability can lead to incorrect conclusions being drawn from simulation experiments. Time variability can cause errors when the measured program phase does not represent the workload's average behavior. Space variability can cause errors when minor timing differences between two configurations result in widely divergent execution paths.

We show that time and space variability are real phenomena that occur in real systems running multi-threaded workloads (Section 2). Variability is a well-known phenomenon in measurement studies, and standard statistical techniques call for taking the mean of multiple observations. Another common (and roughly equivalent) alternative is to measure long enough to minimize the impact of variability. For example, the TPC-C V5.0 benchmark requires a minimum measurement length of two hours [36].

Conversely, variability has been rarely considered in architectural simulation studies. This stems, in part, from the observation that most simulators—including ours—are deterministic: they produce the same timing result every time for the same workload and system configuration. However, we find that small changes in system timing expose the inherent workload variability, leading to large fluctuations in simulated runtime.

In a central result of this paper, we show that ignoring workload variability may result in incorrect conclusions being drawn from simple simulation experiments. In one experiment (Section 4.1), we examined the effect of increasing the cache associativity from 2-way to 4-way on a 16-processor system running our OLTP workload. As expected, this has a small but positive impact on performance when averaged over twenty runs (Section 3.3 describes our technique for injecting random perturbations to create a space of possible runs). However, if we had randomly picked one run from each configuration, we would have had a 31% chance of drawing the wrong conclusion (i.e., that the lower-associativity cache performed better).

We further show (Section 4) that time and space variability pervade full-system simulation. We show that space variability occurs, in varying degrees, across five

This work is supported in part by the National Science Foundation with grants EIA-0205286, EIA-9971256 and CDA-9623632, a Wisconsin Romnes Fellowship (Wood) and donations from IBM, Intel and Sun Microsystems.

commercial workloads and two scientific benchmarks. Space variability also decreases for longer simulations.

These results demonstrate that variability has serious implications for architectural simulation studies using multi-threaded workloads. Worse, the standard measurement solution—running long enough—is not easily applied to simulation because of the orders-of-magnitude slowdown. Modeling a 16-processor system with out-of-order processors on a uniprocessor host, our simulation slowdown is approximately 24,000x compared to the simulated system. This means that simulating the official two-hour TPC-C benchmark would require more than five years, and a one-minute run would require more than 16 days!

To address this problem, we propose a simulation methodology (Section 5) that combines multiple simulations with standard statistical techniques. This methodology greatly reduces the probability of drawing wrong conclusions. It also permits reasonable simulation times using coarse-grain parallelism, provided that multiple simulation hosts are available.

This paper makes two important contributions. First, we define, analyze and discuss the variability phenomenon in multi-threaded workload simulation. Second, we propose a statistical methodology that can be used to obtain more accurate simulation results, while maintaining short simulation runtimes. Our objective is to draw the attention of architects to the variability phenomenon when designing simulation experiments to evaluate systems running multi-threaded workloads.

2. The Variability Phenomenon

In this section, we define variability and explore what causes it in time and space. We demonstrate that this phenomenon occurs in both real and simulated systems and motivate why researchers must address it when performing architectural performance studies.

2.1. What Is Variability?

Variability refers to the differences in performance estimates obtained from multiple runs of a workload. In other words, it refers to the sensitivity of a workload's performance to the particular execution path it takes. Time variability describes the phenomenon where a workload exhibits different performance characteristics over time. Space variability describes the phenomenon where two runs exhibit different performance characteristics, despite starting from the same initial conditions.

Time variability is a well-known phenomenon. Earlier work has frequently described this phenomenon as phase behavior, because many workloads exhibit distinct execution phases with widely different performance characteristics [20, 30, 31]. We prefer the more general name "time variability" because the throughput-oriented commercial workloads we study have a more homogeneous behavior[1], although the exact mix of transactions may vary over time.

Space variability is also well-known in the measurement community. Space variability can arise in any parallel or multi-threaded workload where small timing variations can result in different execution paths, perhaps yielding different performance characteristics.

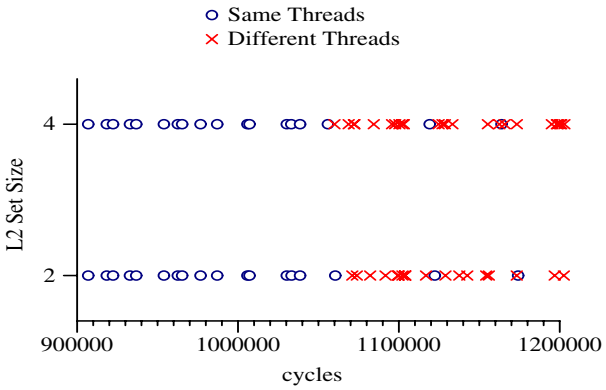


Figure 1. Differences in OS-scheduled threads between two short simulation runs

Small-scale variations arise in real systems due to interrupt timing, bus contention with DMA access, etc. Variations arise in simulation due to small changes in system parameters (e.g., cache size, associativity or miss latency). This small-scale variability can be magnified for a number of reasons, including:

1) the operating system may make different scheduling decisions (e.g., a scheduling quantum may end before an I/O event in one run, but not another);
2) locks may be acquired in different orders, resulting in significant contention in one run, but not another;
3) a transaction may complete during the measurement interval in one run, but not another (see Section 3.1).

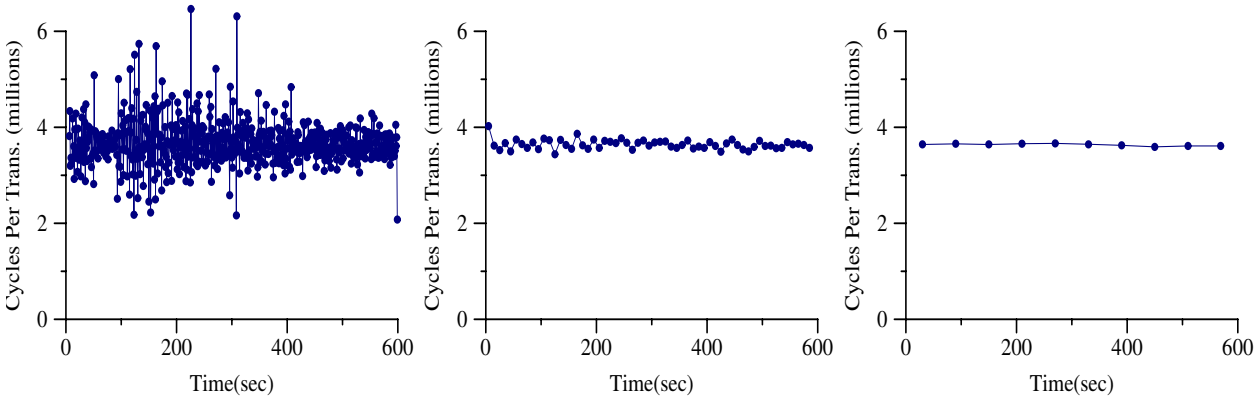
Figure 1 illustrates a snapshot of different OS scheduling decisions in two simulation runs of our OLTP workload. Each data point represents a scheduling event. Both runs start from the same initial conditions. Run 1 (bottom) simulates a system with 2-way set associative caches, while Run 2 simulates a system with 4-way set associative caches. The OS in both runs scheduled the same threads until about 1,060,000 cycles (approximately 1 ms of target execution time). After that point, the behavior in the two runs differs dramatically. The OS scheduled different threads or the same threads in different orders, leading to two completely different execution paths.

2.2. Does Variability Matter In Real Systems?

Time and space variability of multi-threaded workloads are well-known phenomena in real systems. Measurement experiments generally average multiple observations or run long enough to minimize variability.

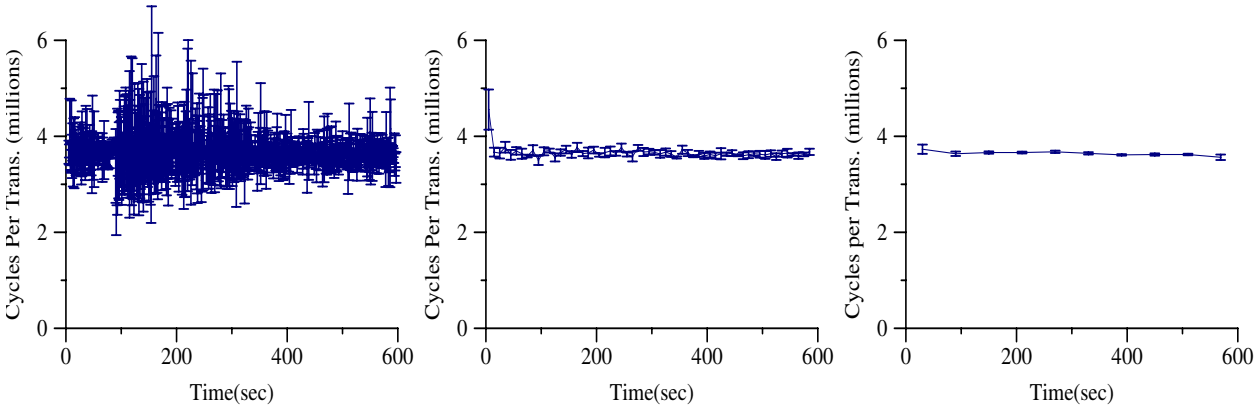
We performed experiments to determine what "long enough" means for our workloads. These experiments were conducted on a Sun E5000 multiprocessor machine with twelve 167 MHz UltraSparcII processors,

1. Many commercial workloads also exhibit distinct phases, e.g., garbage collection in an application server's JVM or flushing the log in a DBMS. However, we focus on the more homogenous parts of these workloads.



(a) 1 second b) 10 seconds c) 60 seconds

Figure 2. OLTP time variability in a real system for different observation intervals (one run)



(a) 1 second b) 10 seconds c) 60 seconds

Figure 3. OLTP space variability in a real system for different observation intervals (five runs)

each with a 512 KB unified L2 cache. UltraSparc processors have hardware performance counters that can be used to measure architectural events on a per-processor basis. We measured the number of cycles per transaction for our OLTP benchmark (described in Section 3.1) emulating 96 users for ten minutes.

Figure 2 demonstrates time variability for one OLTP run. We measured the number of cycles per transaction averaged over intervals of one, ten and sixty seconds. Figures 2a and 2b clearly show that the benchmark exhibits widely different performance characteristics over time (nearly a factor of three difference in some cases). The variability is greatly reduced when the observation interval reaches 60 seconds (Figure 2c is almost a straight line). Some variability is expected, because the OLTP workload consists of five different transaction types. However, the magnitude of variability is surprising, since this system completes over 350 transactions per second on average.

Figure 3 demonstrates the phenomenon of space variability for OLTP. This figure shows the mean and error bars (representing +/- one standard deviation from the mean) of cycles per transaction for five runs on the same Sun E5000 system. Each of the five runs was started from a newly-built database and conducted with no other user processes running to minimize variability due to external factors. This figure illustrates that OLTP exhibits significant space variability even between intervals including over 3,000 transactions (10 seconds). As with time variability, the magnitude of space variability is greatly reduced for 60-second intervals (Figure 3c).

2.3. Does Variability Matter for Simulation?

Researchers often use simulation to evaluate the performance of a design enhancement relative to a base design. In this case, they care less about absolute performance than about the relative performance of a workload on two (or more) different system configurations. Hence space variability is arguably more important for simulation than time variability.[2]

We conducted a simple experiment to illustrate that space variability can affect simulation results enough to cause incorrect conclusions. Figure 4 shows the effect on cycles per transaction of changing the DRAM

2. A common argument is that while a short workload run may not accurately represent the whole workload, at least it represents part of a real workload.

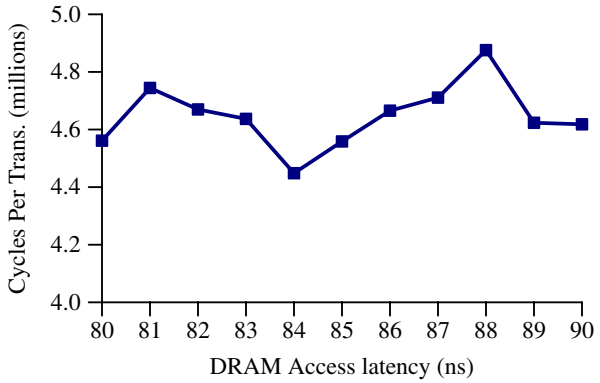


Figure 4. Performance of 500-transaction OLTP runs with different DRAM latencies

latency from 80 to 90 ns, with all other system parameters fixed. All runs start from the same simulation checkpoint (Section 3.2) and complete 500 OLTP transactions. The obvious expected result is that cycles per transaction should increase slightly with the increase in DRAM latency. Clearly, however, the small differences in memory system timing can have an impact at a higher level (e.g., OS scheduling decisions). For example, the 84-ns configuration was 7% faster than the 81-ns configuration. In these two simulations, the OS made completely different scheduling decisions after the first 560,000 cycles (approximately) were completed. While no one is likely to conclude that slower memory is better, it is clear that space variability could lead to the wrong conclusion in other cases.

3. Workloads and Simulation Framework

In this section, we briefly describe the workloads and the simulation infrastructure we use for evaluation. We then motivate and discuss the method we use to introduce variability in simulation runs.

3.1. Workloads

We examine the variability phenomenon using a diverse set of commercial and scientific benchmarks. In this paper, we primarily report results for the OLTP benchmark to facilitate comparisons across experiments. Other commercial benchmarks we study are: Apache (static web content serving), SPECjbb (a Java server benchmark, SPECjbb2000 [34]), Slashcode (dynamic web content serving, used by slashdot.org), and ECPerf (a 3-tier Java workload). Alameldeen et al. [1] and Karlsson et al. [15] describe these workloads in more detail. We also study Barnes-Hut with 16K bodies and Ocean with a 514x514 grid from the SPLASH-2 benchmark suite [38] as examples of scientific applications.

OLTP: DB2 with a TPC-C-like workload. Our OLTP workload is based on the TPC-C v. 3.0 benchmark [35], using the IBM DB2 v. 7.2 EEE database management system. The TPC-C benchmark models the activities of a wholesale supplier, with many concurrent users performing read and write transactions against the database. We used an IBM benchmark kit to build the database and model users who execute transactions without keying or think time. Our experiments use a 4000-warehouse database, with districts, items per warehouse, and customers per district scaled down to limit overall dataset size. The initial size of the database is approximately 800 MB, spread across five raw disks with an additional dedicated database log disk. Simulation experiments emulate 8 users (implying 8 database threads) per processor, and the database is warmed up by running for 10,000 transactions before taking the measurements (unless otherwise specified).

Transaction Quantization Errors. Most commercial multi-threaded workloads are throughput-oriented. These workloads are evaluated using metrics such as transactions per second, calculated by dividing the total number of completed transactions by the measurement interval. To facilitate comparison with traditional benchmarks, we adopt a methodology that measures the (simulated) time to finish a fixed number of transactions [1]. We use the number of cycles per transaction as our performance metric throughout the paper, even for workloads with different types of transactions.

Using either method, cold-start and end-effects may bias the results if too few transactions are measured. The first transaction to complete within the interval will have started before the interval began. Similarly, when the N^{th} (last) transaction completes, the $N+1^{st}$, $N+2^{nd}$, etc. transactions have already started. Simulation runs should be long enough to mitigate these effects.

3.2. Simulation Infrastructure

This section describes our target system simulation model, and details of our simulation infrastructure.

3.2.1. Target System Model

We model a 16-node system similar to the Sun E10000 [6]. Each node contains a processor core, split L1 instruction and data caches (each is 128 KB 4-way set associative, using 64-byte blocks), a unified L2 cache (4 MB, 4-way set associative, 64-byte blocks), a cache controller, and an integrated memory controller for a portion of the 2 GB shared main memory. System caches are kept coherent using an MOSI invalidation-based snooping cache coherence protocol. We assume a two-level hierarchy of crossbar switches for the interconnection network to connect the nodes, with a delay of 50 ns for each network traversal (which includes wire propagation, synchronization, and routing). We assume 80 ns for the DRAM access time. When a protocol request arrives at a processor or at memory, it takes 25 ns or 80 ns, respectively, to provide data to the interconnect. These assumed latencies result in a 180 ns latency to obtain a block from memory and a 125 ns latency for a cache-to-cache transfer. We assume a 1 GHz system clock (i.e., a 1 ns system cycle).

3.2.2. Full-System Simulation

We use Simics [22, 37], a full-system multiprocessor simulator. Simics is a system-level architectural simulator developed by Virtutech AB that is capable of running unmodified commercial operating systems and applications. We configured Simics to model a SPARC V9 target system running unmodified Solaris 8. Simics is only a functional simulator by default, but it can be extended with detailed processor and memory system timing models, described in the next two subsections.

Simics has a checkpointing facility that enables us to capture the full state of the system (including register state, memory, disks and outstanding interrupts). We use this checkpointing facility to start simulation runs from the same initial conditions. We also record multiple checkpoints over a workload's execution to allow evaluation of time variability.

3.2.3. Memory System Model

We extend Simics with a memory system simulator that accurately models the timing of memory requests. The simulator—described in more detail by Martin et al. [23, 24] and Alameldeen et al. [1]—supports a broad range of coherence protocols, specified using a table-driven specification methodology. It accurately models state transitions (including transient states) of the coherence protocols in cache and memory controllers. Our simulations capture timing-dependent race conditions and lock contention events that cannot be captured using a trace-driven methodology.

3.2.4. Processor Models

We present results using two different processor models. For most results, we use a fast but simple blocking processor model that would complete one billion instructions per second at 1 GHz (i.e. an IPC of 1) if the L1 caches were perfect. We extended this simple timing model with the Ruby memory hierarchy simulator, to accurately model the L1 and L2 caches and the remainder of the memory system.

For more detailed—but 6–8 times slower—simulations, we use TFsim [25] to model out-of-order processor cores and L1 caches. TFsim models a four-wide superscalar processor, with an eleven stage pipeline (fetch (3), decode (3), schedule (1), execute (1 or more), and retire (3)). TFsim models a 1-KB YAGS direct branch predictor [11], a 64-entry cascaded indirect branch predictor [9], a 64-entry return address stack predictor [14], and a 64-entry reorder buffer (unless otherwise specified).

3.3. Introducing Variability

As described in Section 2.1, workloads exhibit space variability on real systems due to small timing variations. Our simulator, however, is deterministic: it produces the same execution path for each workload/system configuration every time (starting from the same checkpoint). To evaluate space variability, we must inject small timing variations to create a space of possible executions starting from the same initial conditions.

To do this, we artificially introduce small perturbations in memory system timing. On each L2-cache miss, we added a uniformly distributed pseudo-random integer between 0 and 4 ns. This increases the average L2 miss latency by 2 ns. For multiple simulations, each run uses a unique random seed, leading to a different sequence of miss latencies. Note that the average miss latency remains the same for each run. However, the small perturbations lead to different execution paths and different runtime results.

In a sensitivity experiment on OLTP, we showed that the magnitude of the random perturbation did not have a significant effect on variability. When the uniformly-distributed discrete increment was chosen between 0 and 1 ns (instead of 0-4 ns), the coefficient of variation of the runtimes (defined as the 100 times the ratio of the standard deviation to the mean [12]) was not significantly affected.

This artificial method of introducing timing perturbations is used throughout the next section to produce a space of runs for our simulation experiments. We use the mean of these runs as our performance metric.

4. Variability in Simulation Results

This section examines in detail the impact of variability on simulation results. We first define a metric that indicates the likelihood of drawing the wrong conclusion if variability is ignored. We then examine the sensitivity of space variability to different benchmarks and simulation run lengths. We also confirm that time variability is significant in workload simulation.

4.1. Wrong Conclusion Ratio

Section 2 demonstrated the possibility of drawing an incorrect conclusion when using single simulation experiments to compare different system configurations. To quantify this risk, we define a new metric, the wrong conclusion ratio (WCR), as the percentage of comparison experiment pairs that reach an incorrect conclusion. For example, when conducting an experiment to compare the performance of systems A and B, we use N runs of workload W for each system. The correct conclusion is the relationship between the averages of the N runs on A and the N runs on B (for example, A outperforms B). The wrong conclusion ratio is the percentage of the N^2 pairs of runs that lead to the opposite conclusion (in this case, that B outperforms A).

WCR can be used to estimate the *wrong conclusion probability* if a researcher ignores variability in multi-threaded workloads and uses single simulations. We next present WCR results for two experiments involving cache and microarchitectural design decisions.

4.1.1. Experiment 1: Cache Design

In this experiment, we simulated twenty 200-transaction runs of our OLTP workload using the simple processor model. We varied the L2 cache associativities, while holding the cache hit and miss latencies constant. We considered direct-mapped, 2-way and 4-way set associative caches, with cache sizes fixed at 4 MB. The

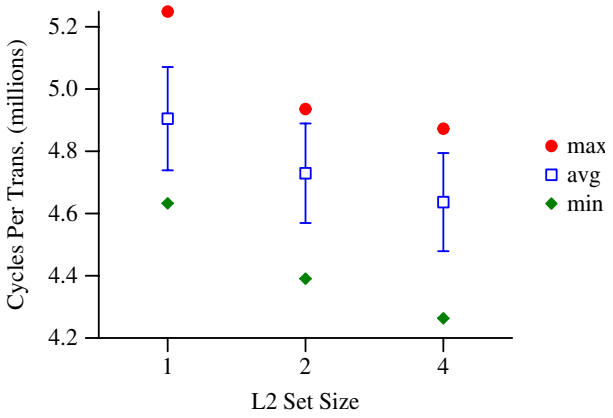


Figure 5. OLTP performance for different L2 cache associativities

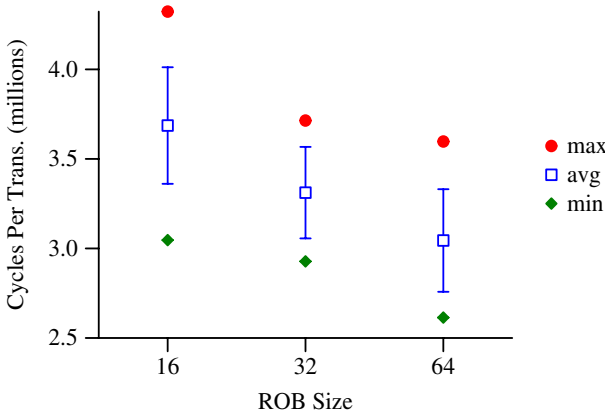


Figure 6. OLTP performance for different reorder buffer sizes

expected conclusion from this experiment is that runtime decreases when the associativity increases.

Figure 5 shows the average (with error bars representing +/- one standard deviation), maximum, and minimum number of cycles per transaction for the three L2 cache configurations. The average performance for these configurations confirms our expected conclusion. However, the range of results for the three configurations overlap. Clearly, if we performed only one simulation run for each configuration, there is a risk that we might erroneously conclude that a direct-mapped cache outperforms a 4-way set associative one.

To estimate this risk, Table 1 shows the WCR values for this experiment, obtained by enumeration of all possible pairs. The results show, for example, that 31% of the pairwise comparisons would lead to the wrong conclusion that a 2-way set-associative cache configuration outperforms a 4-way configuration.

An interesting observation is that single simulations can result in misleading conclusions either way. For example, while the average of the 4-way system outperforms the average of the direct-mapped system by 6%, the two opposite extremes lead to completely different and misleading conclusions. The minimum of the direct-mapped system outperforms the maximum of the 4-way system by 5%, whereas the minimum of the 4-way system outperforms the maximum of the direct-mapped system by 23%. However, there is a small chance (1 in 400 for our 20-run experiment) that the execution paths followed by single simulations may lead to one or the other of these conclusions.

Table 1. Summary of Experiment 1

Configurations Compared (Superior Configuration).	WCR (%).
Direct Mapped vs. (2-way SA)	24%
Direct Mapped vs. (4-way SA)	10%
2-way SA vs. (4-way SA)	31%

Table 2. Summary of Experiment 2

Configurations Compared (Superior Configuration).	WCR (%).
16-entry vs. (32-entry) ROB	18%
16-entry vs. (64-entry) ROB	7.5%
32-entry vs. (64-entry) ROB	26%

4.1.2. Experiment 2: Microarchitectural Design

In this experiment, we simulated twenty 50-transaction runs for our OLTP workload using TFsim [25] to compare the performance of microarchitectural configurations that differ only in the reorder buffer size. These configurations have reorder buffers of 16, 32 and 64 entries, respectively. The expected conclusion from this experiment is that runtime decreases when the ROB size increases.

Figure 6 shows the average (with error bars), maximum, and minimum number of cycles per transaction for the three microarchitectural configurations. The averages confirm the expected conclusion[3]. But again, the result ranges overlap, leaving the possibility of an incorrect conclusion. Table 2 presents the WCR values for single comparison experiments. It shows, for example, that 26% of the possible experiment pairs lead to the wrong conclusion that a 32-entry ROB configuration outperforms a 64-entry configuration.

4.1.3. Summary

These two simple experiments clearly illustrate that ignoring variability can lead to erroneous conclusions in a significant percentage of single-simulation experiments. Note that a high WCR usually implies that two configurations are close in performance, and that it may not be possible to conclude that one outperforms the other. In Section 5, we discuss using standard statistical techniques, i.e., confidence intervals and hypothesis testing, to determine when it is safe to draw conclusions.

3. TFsim models a 4-wide out-of-order superscalar processor, resulting in a lower number of cycles per transaction than Experiment 1.

Table 3. Summary of space variability for different benchmarks

Benchmark.	Barnes.	Ocean.	ECPerf.	Slashcode.	OLTP.	Apache.	SPECjbb.
#transactions	1	1	5	30	1000	5000	60,000
Coefficient of Variation	0.16%	0.31%	1.40%	3.60%	0.98%	0.88%	0.26%
Range of Variability	0.59%	1.13%	5.30%	14.45%	3.85%	3.94%	1.10%

4.2. Simulated Space Variability

To understand the scope of space variability in multi-threaded workloads, we examine various benchmarks and simulation run lengths. In this section, twenty simulation runs were used for each data point, using the simple processor model. We use the mean of the simulated runtimes as the performance metric, and the coefficient of variation to estimate the magnitude of space variability. We define another metric, the *range of variability*, as the difference between the maximum and the minimum runtimes, taken as a percentage of the mean. The higher the range of variability, the more likely one is to make an incorrect conclusion.

4.2.1. Space Variability and Different Benchmarks

We compare the space variability across our seven benchmarks on a 16-processor system. The number of transactions executed for each benchmark vary from 1 (for Barnes-Hut and Ocean, where the whole benchmark is viewed as one transaction) to 60,000 for SPECjbb. The transaction counts were selected to limit the simulation runtime for each benchmark run to less than ten hours (except ECPerf).

Figure 7 presents the mean, error bars, and extremes for the seven benchmarks. Table 3 shows the number of simulated transactions, the coefficient of variation, and the range of variability for each benchmark. Table 3 (last row) shows that variability ranges from less than 1% for Barnes-Hut to more than 14% for Slashcode. The range of variability exceeds 3% for four out of the five commercial benchmarks, even for these relatively long runs with the simple processor model. In addition, these results show that OLTP (which we use throughout the paper) is not an extreme case in terms of space variability.

4.2.2. Space Variability and Run Lengths

Experiments on real systems show that space variability of multi-threaded workloads decreases with an increase in the observation interval. Table 4 confirms this result for our simulation experiments. As the number of simulated OLTP transactions increases from 200 to 1000, the coefficient of variation and the range of variability decrease, indicating less variability. However, the decrease in variability comes at the expense of longer simulation times.

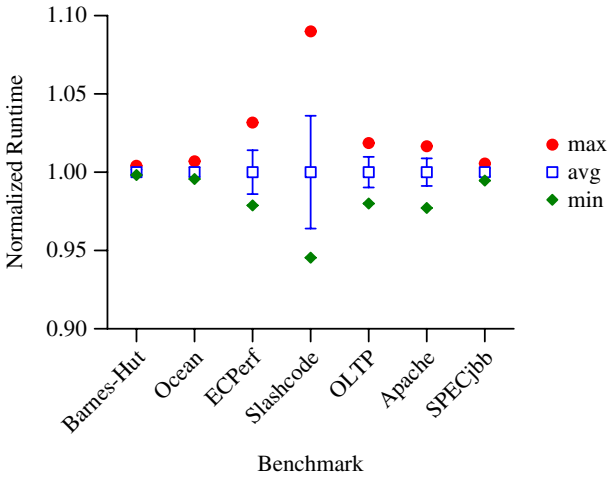


Figure 7. Variabililty of different benchmarks

4.3. Simulated Time Variability

Section 2.2 confirmed the well-known result that multi-threaded workloads running on real systems exhibit different characteristics over time. To demonstrate this phenomenon in simulation experiments, we conducted ten 40,000-transaction OLTP runs (about one month of simulation time for each run, using the simple processor model). Partial results were produced every 200 transactions. Figure 8 presents the average and standard deviation (error bars) for the number of cycles per transaction, where each data point represents 200 transactions. This figure shows that OLTP exhibits different characteristics over time, with cycles per transaction varying by up to 27%.

Figure 9 shows the average (with error bars) for 20 runs starting from ten different starting points in the workload lifetime of OLTP and SPECjbb. Figure 9a

Table 4. OLTP space variability for different run lengths

#Simulated Transactions.	200.	400.	600.	800.	1000.
Coefficient of Variation	3.27%	2.87%	2.16%	1.53%	0.98%
Range of Variability	12.72%	10.40%	7.65%	5.47%	3.86%
Average Runtime (1 simulation) in hrs.	1.79	3.62	5.48	7.36	9.26
Total Runtime (20 simulations) in hrs.	35.82	72.35	109.64	147.13	185.11

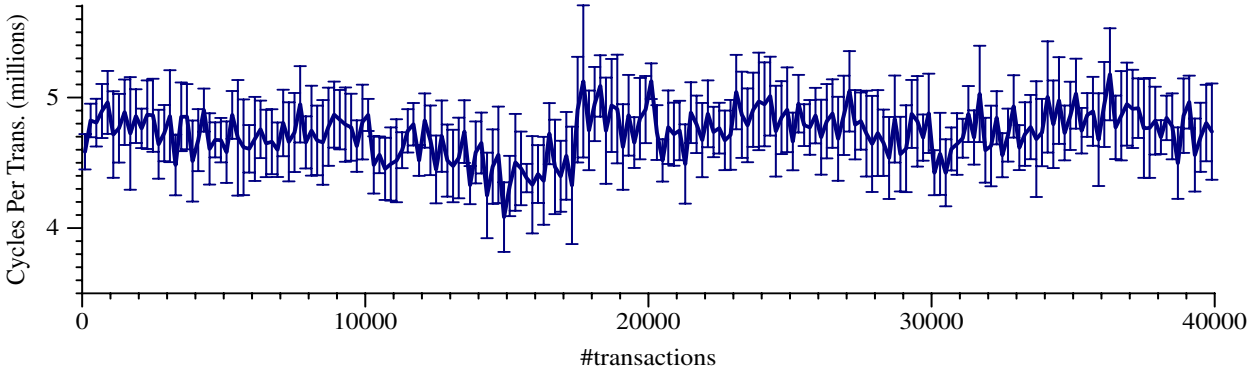


Figure 8. Time variability for different phases of long OLTP runs

shows results for 200-transaction OLTP runs, and Figure 9b shows results for 5,000-transaction SPECjbb runs. The OLTP results show that performance depends critically on which checkpoint is used to start the simulation. The difference between the average cycles per transaction of runs starting from the 30K and 40K checkpoints is more than 16%. For SPECjbb, which showed almost no space variability (standard deviation of runs starting from the same checkpoint is negligible), the difference between runs starting from the 100K and 400K checkpoints is more than 36%. This shows that time variability can be an issue even for benchmarks with almost no space variability.

For workloads that exhibit this behavior, time-sampling approaches are necessary to decrease the probability of reaching incorrect conclusions, while enabling architectural studies to be completed within reasonable simulation time limits [17,21]. Section 5 discusses using short runs from multiple checkpoints to estimate the average workload performance on a given configuration.

5. Statistical Simulation Methodology

Classical statistics provides a wealth of techniques for coping with variability. In this section, we apply a few of those methods to account for variability while keeping simulation time within reasonable bounds.

5.1. Accounting for Space Variability

Averaging the results of multiple trials forms the basis of classical experiment designs. The intuition behind this approach is that the coefficient of variation (our estimate of space variability) decreases when the sample size (number of runs) increases. To account for space variability, we need to obtain enough runs to increase confidence in our conclusions, i.e., to decrease the probability of drawing wrong conclusions.

We apply two standard statistical techniques—confidence intervals and hypothesis testing—to estimate the probability of drawing wrong conclusions when comparing two configurations. Two types of errors exist in such conclusions: errors concerning the direction of the relationship (i.e., which configuration performs better); and errors related to the magnitude of the difference (i.e., speedup of one configuration over the other). The *wrong conclusion probability* estimates the probability of drawing wrong conclusions about the direction of the relationship, which is our focus in this paper. Confidence intervals place a conservative upper bound on the wrong conclusion probability, but it can also help estab-

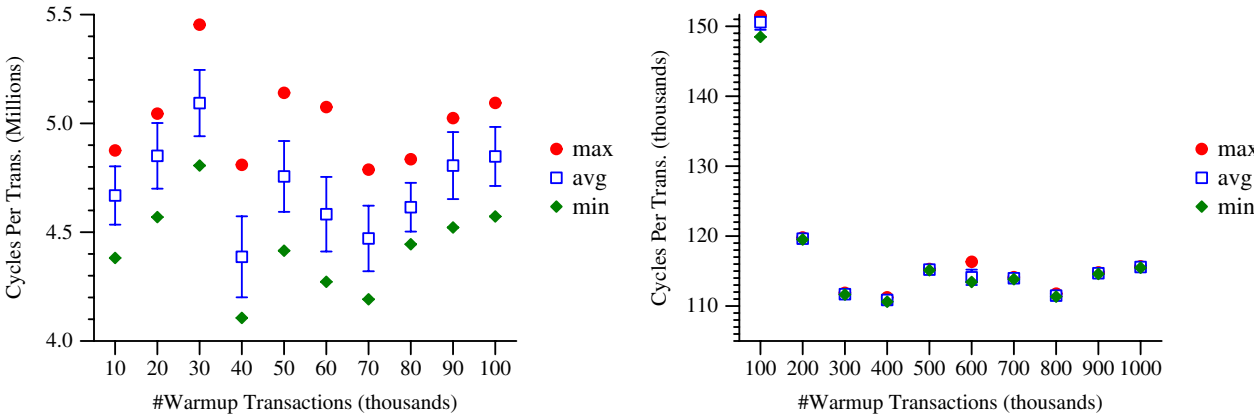


Figure 9. OLTP and SPECjbb performance from multiple starting points

lish approximate bounds on the magnitude of the difference (not discussed here). Hypothesis testing, on the other hand, provides a tighter, more accurate estimate of the wrong conclusion probability, but does not help—in the simple form described in this section—in establishing the magnitude of the difference.

5.1.1. Confidence Intervals

A *confidence interval* (CI) is defined as the range of values that is expected to include a population parameter (e.g., mean) [12]. The *confidence probability* is the probability that the true population parameter will fall inside the confidence interval. For example, if the mean cycles per transaction for OLTP lies in the interval between 4 and 5.5 million with confidence probability 99%, we are 99% certain that the true mean lies within that interval. The confidence interval for the mean of a normally distributed infinite population is given by [7]:

$$\bar{y} - \frac{ts}{\sqrt{n}} \leq mean \leq \bar{y} + \frac{ts}{\sqrt{n}}$$

where $\bar{y}$ is the sample mean, s is the sample standard deviation, n is the sample size, and t is the value of the normal deviate corresponding to the desired confidence probability (obtained from the student's t-distribution with *(n-1)* degrees of freedom if the sample size is less than 50, and from the normal distribution otherwise). Values for t can be obtained from standard statistical tables. We can reach a tighter confidence interval by increasing the sample size, n, or decreasing the confidence probability, thus decreasing t.

Confidence intervals can be used to estimate an upper bound on the wrong conclusion probability when comparing two alternatives. If the confidence intervals of the two alternatives do not overlap, the probability of reaching a wrong conclusion will be at most *(1-p)*, where p is the confidence probability[4]. Figure 10 shows the 95% confidence intervals for the data in Experiment 2 (Section 4.1.2). As expected, the confidence intervals get tighter as the sample size increases. Confidence intervals for 20 runs do not overlap, which implies that the probability of reaching a wrong conclusion is less than 5% (compared to the 26% WCR for single experiments). For the smaller sample sizes, the results are not statistically significant (at the 95% confidence level), because the confidence intervals overlap. Note that if we reduce the confidence probability to 90%, a sample size of 15 becomes statistically significant, but there remains (at most) a 10% chance of reaching a wrong conclusion.

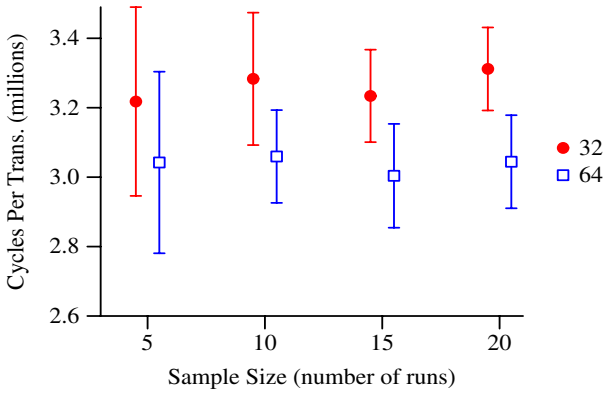


Figure 10. 95% confidence intervals using different sample sizes for 32 and 64-entry ROBs

4. Comparison conclusions are correct only if both means were inside their respective CIs (p^2 probability), or if one mean was inside its CI and the other was outside but in the opposite direction to the first mean's CI *(p(1-p))*. Thus, the probability of reaching a wrong conclusion is bounded by: $1-(p^2+p(1-p)) = 1-p$. We ignore the case where both means are outside their CIs due to its small probability.

Estimating the Sample Size. When we design a simulation experiment, we need to estimate the number of runs needed to obtain statistically significant results. Assuming an infinite population, the sample size can be estimated according to population parameters and the desired level of precision (or maximum allowed error) of the results. For example, if we want to limit the relative error of the estimated population mean to r (compared to the true mean), the sample size required is estimated using:

$$n = \left(\frac{tS}{r\bar{Y}}\right)^2$$

where $\bar{Y}$ and S are the true population mean and standard deviation, and t is the normal deviate corresponding to the desired confidence probability [7]. Since the true values of $\bar{Y}$ and S are not known beforehand, an approximation for their values can be used from prior knowledge about the same population. The confidence probability is chosen according to be the desired bound on the wrong conclusion probability.

As an example, if we require the relative error in the number of cycles per transaction to be less than 4% (r = 0.04) with a confidence probability of 95% ($t \cong 2$), and using an estimate of $S/\bar{Y}$ = 9% (0.09) (which is the approximate coefficient of variation we observed for 50-transaction OLTP runs), the number of runs required for our OLTP benchmark is $(2*0.09/0.04)^2 \cong 20$. In comparison simulation experiments, r should be selected to be less than half the expected performance improvement between the configurations being compared, in order to avoid overlap in confidence intervals.

5.1.2. Hypothesis Testing

Hypothesis testing is another statistical technique that we can use to evaluate conclusions of simulation experiments. Hypothesis testing has a variety of forms that depend on the parameter under investigation and its distribution. In this section, we describe testing a hypothesis about the relationship between two means of a normally-distributed population [12]. We next illustrate the use of hypothesis testing in simulation experiments to determine a tighter upper bound on the wrong conclusion probability.

In Experiment 2 (Section 4.1.2), our conclusion was that the 64-entry ROB configuration is better than the 32-entry configuration, i.e., the mean runtime for a 32-entry ROB is greater than the mean runtime for a 64-entry ROB. Since this conclusion was based on the

sample means of runs, we want to test the hypothesis that there is actually no significant difference between the two *true means*[5]. The test hypothesis in this case is:

H_0: $\mu_{32} - \mu_{64} = 0$ (or $\mu_{32} = \mu_{64}$)

where μ_{32} and μ_{64} are the true mean runtimes of the 32-entry and 64-entry ROB, respectively. If we reject the test hypothesis, this means we accept the alternative hypothesis that μ_{32} is greater than μ_{64}. The significance level of the test (α) is determined by the error probability we can tolerate. In this setting, we want to avoid a *type I error*, defined as the probability of rejecting the test hypothesis when it is correct. This is equivalent to the probability that our conclusion (i.e., accepting the alternative hypothesis) is wrong.

If we assume that the runtimes for the two configurations are independent, normally distributed random variables with unknown variances, and we collect an equal number of runs (n) from both configurations, the test statistic to test the hypothesis (H_0) is given by [12]:

$$t = \frac{\bar{y}_{32} - \bar{y}_{64}}{\sqrt{\frac{s_{32}^2 + s_{64}^2}{n}}}$$

where $\bar{y}_{32}$, $\bar{y}_{64}$ are the sample mean runtimes for the 32-entry and 64-entry configurations; s_{32}^2 and s_{64}^2 are their sample variances. If the test statistic lies in the upper tail of the t-distribution with ($2n$-2) degrees of freedom at a certain significance level α, we reject the test hypothesis (i.e., accept the alternative hypothesis) at this level (Figure 11). In that case, the probability of drawing a wrong simulation conclusion—obtained from sample means—is less than α.

Using this method, we can estimate the probability of reaching a wrong conclusion for a certain simulation experiment. We calculate the test statistic and compare it against critical values of the t-distribution at various significance levels. The wrong conclusion probability will be bounded by the smallest significance level at which the test hypothesis is rejected.

Estimating the Sample Size. We can estimate the number of runs necessary to achieve a certain significance level by evaluating the test statistic for different numbers of runs, and selecting the minimum number required to reject the test hypothesis. Table 5 presents the necessary number of runs for the ROB experiment to achieve a certain upper bound on the wrong conclusion probability. Nine runs are necessary to limit the probability of drawing wrong conclusions to 0.05, and sixteen runs limit this probability to 0.005 (compared to the conservative 0.05 limit obtained from confidence intervals for twenty runs).

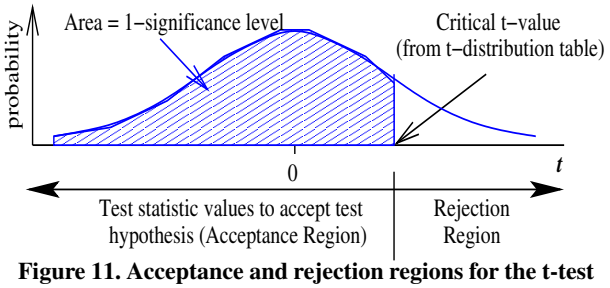


Figure 11. Acceptance and rejection regions for the t-test

Table 5. Number of runs needed for different significance levels

Significance Level (Wrong Conclusion probability).	#Runs Needed.
10%.	6.
5%.	9.
2.5%.	11.
1%.	13.
0.5%.	16.

5.2. Accounting for Time Variability

In order to account for time variability, the sample should include runs from multiple starting points distributed throughout the program runtime. However, determining the starting points for our sample in the workload's runtime is a challenge, since many workloads exhibit substantial time variability (Figures 8 and 9). While some methods have been developed to determine representative samples in single-threaded applications [32], these methods are not directly applicable to multi-threaded applications. Fortunately, sampling theory provides a broad array of techniques for selecting samples [7]. We focus on systematic sampling, where starting points are taken at fixed time intervals, and then apply techniques of Section 5.1 to estimate the wrong conclusion probability for a certain sample size.

In workloads that exhibit only one type of variability (e.g., SPECjbb exhibits only time variability), using the techniques in Section 5.1 is straightforward. However, if we need to account for both space and time variability, we have to determine which type is the greater source of error. The analysis of variance (ANOVA) [12] is a technique that can be used to estimate whether averages of runs from different groups are (statistically) the same. If averages are the same, variability between different groups can be attributed to the same effects causing variability within each group. Otherwise, there is a significant variability between averages of different groups (i.e., time variability) that cannot be attributed to the within-group variability (i.e., space variability). In other words, ANOVA tells us whether it is sufficient to use runs from a single starting point, or whether the sample should contain runs from many starting points.

We performed an ANOVA study on the groups of data points of OLTP and SPECjbb summarized in Figure 9. We considered different numbers of groups, group sizes and significance levels (0.1, 0.05 and 0.01). Our results, for both workloads, show that between-group variability is significant and cannot be attributed to the within-group variability. For both of these work-

5. The *true mean* here is the average runtime of all possible runs of a particular configuration. Since the number of runs is practically infinite, the average runtime for a sample of runs (*sample mean*) is used as an estimate of the true mean.

loads, time variability is significant, and simulations should be performed from different starting points. The number of different starting points should be selected in a similar manner to what was outlined in Section 5.1.

Our methodology can be improved in several directions. Given a fixed simulation budget (time allowed for all simulations), a tradeoff must be made between the length of each simulation and the number of simulations required to maximize the confidence probability (and minimize cold-start bias [17]). If the simulated system configuration has an impact on variability, ANOVA can be performed for different workload/system configuration combinations. Sampling techniques other than systematic sampling can be used to select representative time samples. These issues are left for future work.

6. Related Work

This work builds on Alameldeen et al. [1], a paper that identified commercial workload variability and solution directions. Some studies have evaluated commercial workloads performance on real systems using hardware counters [2, 16] or hardware emulation tools [28]. Prior simulation studies have evaluated multiprocessor system performance using execution-driven simulation of scientific applications [10, 38] and commercial workloads [27], or full system simulation of commercial workloads [2]. Other studies have evaluated the fidelity of simulation results for uniprocessors [8] and multiprocessors [13] compared to the actual hardware being modeled. Krishnan and Torrellas examined experimental errors in multiprocessor simulations due to simple processor models [18]. Cain et al. [5] discussed issues related to simulation precision and accuracy. In our infrastructure, we use TFsim to increase the simulation precision, and commercial workloads to increase accuracy.

Our work distinguishes itself from other studies by focussing on the variability phenomenon in simulation and providing a methodology to address it. Very few previous studies report results from multiple simulation runs to account for space variability [23, 24, 33]. Changes in program phase behavior were explored for SPEC benchmarks [20, 30]. Simulation errors introduced by selecting particular program phases were investigated by Sherwood et al. [31]. Statistical simulation based on program traces was used by Oskin et al. [29]. Some architectural studies used sampling techniques to estimate values of architectural parameters (e.g. [17, 21]), but not in the context of multi-threaded commercial workloads. Kuck et al. [19] defined a stability metric for the performance of different computational kernels running simultaneously on a multiprocessor, but did not discuss instability due to a single multi-threaded workload.

7. Conclusions

In this paper, we show that time and space variability are important phenomena that must be addressed in architectural simulation studies of multi-threaded workloads. We demonstrate that these phenomena exist in both real machine and simulation experiments, and provide evidence that operating system scheduling decisions are one significant source. We further show that the standard practice of ignoring variability in simulation can lead to incorrect conclusions in a significant percentage of microarchitectural and system design experiments.

We describe a simple methodology to compensate for variability, that combines pseudo-random perturbations, multiple simulations and standard statistical techniques. This methodology is intended to help architects determine when it is safe to draw conclusions from simulation experiments.

Acknowledgments

We thank Milo Martin and Dan Sorin who were the first to advocate multiple simulations to deal with variability. This work builds on the Multifacet simulation infrastructure, whose creators include Milo Martin and Dan Sorin (memory system simulator), Carl Mauer (TFsim), Ross Dickson, Pacia Harper, Kevin Moore, and Min Xu. We also thank Virtutech AB, the Wisconsin Condor group, the Wisconsin computer systems lab staff, David DeWitt, and Anastassia Ailamaki for their help and support. Thanks to Brad Beckmann, Mark Hill, Mikko Lipasti, Milo Martin, Carl Mauer, Min Xu and our anonymous reviewers for their suggestions.

References

[1] Alaa R. Alameldeen, Carl J. Mauer, Min Xu, Pacia J. Harper, Milo M.K. Martin, Daniel J. Sorin, Mark D. Hill, and David A. Wood. Evaluating Non-deterministic Multi-threaded Commercial Workloads. In *Proceedings of the Fifth Workshop on Computer Architecture Evaluation Using Commercial Workloads*, pp. 30–38, February 2002.

[2] Luiz A. Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory System Characterization of Commercial Workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 3–14, June 1998.

[3] Luiz Andre Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 282–293, June 2000.

[4] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel. A Multithreaded PowerPC Processor for Commercial Servers. *IBM Journal of Research and Development*, 44(6):885–898, November 2000.

[5] Harold W. Cain, Kevin M. Lepak, Brandon A. Schwartz, and Mikko H. Lipasti. Precise and Accurate Processor Simulation. In *Proceedings of the Fifth Workshop on Computer Architecture Evaluation Using Commercial Workloads*, pp. 13–22, February 2002.

[6] Alan Charlesworth. Starfire: Extending the SMP Envelope. *IEEE Micro*, 18(1):39–49, Jan/Feb 1998.

[7] William G. Cochran. *Sampling Techniques*. John Wiley & Sons, third edition, 1977.

[8] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. Measuring Experimental Error in Microprocessor Simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp. 266–277, July 2001.

[9] Karel Driesen and Urs Holzle. Accurate Indirect Branch Prediction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 167–178, June 1998.

[10] S. Dwarkadas, J. R. Jump, and J. B. Sinclair. Execution-Driven Simulation of Multiprocessors: Address and Timing Analysis. *ACM Transactions on Modeling and Computer Simulation*, 4(4):314–338, 1994.

[11] Avinoam Nomik Eden and Trevor Mudge. The YAGS Branch Prediction Scheme. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 69–77, June 1998.

[12] Harry Frank and Steven G. Althoen. *Statistics: Concepts and Applications*. Cambridge University Press, first edition, 1994.

[13] 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 Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[14] Stephan Jourdan, Tse-Hao Hsing, Jared Stark, and Yale N. Patt. The Effects of Mispredicted-Path Execution on Branch Prediction Structures. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 58–67, October 1996.

[15] Martin Karlsson, Kevin E. Moore, Erik Hagersten, and David A. Wood. Memory Characterization of the ECperf Benchmark. In *Second Annual Workshop on Memory Performance Issues (WMPI), in conjunction with ISCA-29*, 2002.

[16] Kimberly Keeton, David A. Patterson, Yong Qiang He, Roger C. Raphael, and Walter E. Baker. Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 15–26, June 1998.

[17] R. E. Kessler, Mark D. Hill, and David A. Wood. A Comparison of Trace-Sampling Techniques for Multi-Megabyte Caches. *IEEE Transactions on Computers*, 43(6):664–675, 1994.

[18] Venkata Krishnan and Josep Torrellas. A Direct-Execution Framework for Fast and Accurate Simulation of Superscalar Processors. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 286–293, October 1998.

[19] D. Kuck, E. Davidson, D. Lawrie, A. Sameh, C. Q. Zhu, A. Veidenbaum, J. Konicek, P. Yew, K. Gallivan, W. Jalby, H. Wijshoff, R. Bramley, U. M. Yang, D. Padua P. Emrath, R.Eigenmann, J. Hoeflinger, G. Jaxon, Z. Li, T. Murphy, and J. Andrews. The Cedar System and an Initial Performance Study. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp. 213–223, May 1993.

[20] Thierry Lafage and Andre Seznec. Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream. In *3rd Annual Workshop on Workload Characterization*, September 2000.

[21] Subhasis Laha, Janak H. Patel, and Ravishankar K. Iyer. Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems. *IEEE Transactions on Computers*, 37(11):1325–1336, 1988.

[22] Peter S. Magnusson et al. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, February 2002.

[23] Milo M. K. Martin, Daniel J. Sorin, Anastassia Ailamaki, Alaa R. Alameldeen, Ross M. Dickson, Carl J. Mauer, Kevin E. Moore, Manoj Plakal, Mark D. Hill, and David A. Wood. Timestamp Snooping: An Approach for Extending SMPs. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 25–36, November 2000.

[24] Milo M. K. Martin, Daniel J. Sorin, Mark D. Hill, and David A. Wood. Bandwidth Adaptive Snooping. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, pp. 251–262, January 2002.

[25] Carl J. Mauer, Mark D. Hill, and David A. Wood. Full System Timing-First Simulation. In *Proceedings of the 2002 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 108–116, June 2002.

[26] Ann Marie Grizzaffi Maynard, Coletter M. Donnelly, and Bret R. Olszewski. Contrasting Characteristics and Cache Performance of Technical and Multi-User Commercial Workloads. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 145–156, October 1994.

[27] Ashwini Nanda, Yiming Hu, Moriyoshi Ohara, Caroline D. Benveniste, Mark E. Giampapa, and Maged Michael. The Design of COMPASS: An Execution Driven Simulator for Commercial Applications Running on Shared Memory Multiprocessors. In *Proceedings of the 12th International Parallel Processing Symposium*, March 1998.

[28] Ashwini Nanda, Kwok-Ken Mak, Krishnan Sugavanam, Ramendra K. Sahoo, Vijayaraghavan Soundararajan, and T. Basil Smith. MemorIES: A Programmable, Real-Time Hardware Emulation Tool for Multiprocessor Server Design. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[29] Mark Oskin, Frederic T. Chong, and Matthew Farrens. HLS: Combining Statistical and Symbolic Simulation to Guide Microprocessor Designs. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 71–82, June 2000.

[30] Timothy Sherwood and Brad Calder. Time Varying Behavior of Programs. Technical report, UC San Diego Technical Report UCSD-CS99-630, August 1999.

[31] Timothy Sherwood, Erez Perelman, and Brad Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 3–14, September 2001.

[32] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 45–57, October 2000.

[33] Daniel J. Sorin, Milo M.K. Martin, Mark D. Hill, and David A. Wood. SafetyNet: Improving the Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pp. 123–134, May 2002.

[34] Systems Performance Evaluation Cooperation. SPEC Benchmarks. http://www.spec.org.

[35] Transaction Processing Performance Council. TPC-C. http://www.tpc.org/tpcc/.

[36] Transaction Processing Performance Council. TPC Benchmark C, Standard Specification, Revision 5.0, February 2001.

[37] Virtutech AB. Simics Full System Simulator. http://www.simics.com/.

[38] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 24–37, June 1995.