Appears in the proceedings of the
Computer Architecture Evaluation using Commercial Workloads (CAECW-02)
February 2, 2002

Evaluating Non-deterministic Multi-threaded Commercial Workloads

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

Computer Sciences Department
University of Wisconsin — Madison
http://www.cs.wisc.edu/multifacet/

Abstract

Full-system simulation is increasingly used to evaluate the performance of commercial workloads on future multiprocessor designs. However, challenges such as simulation slowdown, sizing constraints, and workload tuning impede the development of commercial workloads for timing simulators. We describe how we address these challenges in our development of four commercial workload benchmarks.

This paper introduces non-deterministic workload behavior as another potential challenge in timing simulation. Non-determinism refers to the sensitivity of the system's timing to small changes in its parameters. This problem is nearly universally ignored because most simulators (including ours) are deterministic: they produce the same timing result every time, for given a workload and system parameters. However, we find that small changes in the memory latency can cause large changes in run-time (nearly 10%). We propose a methodology that uses pseudo-random perturbations and standard statistical techniques to compensate for these non-deterministic effects. Finally, we provide evidence that commercial workloads have different characteristics over time, further supporting a sampled simulation methodology.

1 Introduction

Commercial workload performance is an important metric for shared-memory multiprocessor computer systems. Full-system timing simulation is increasingly used to evaluate the performance of these workloads on future multiprocessor designs [11, 17]. However using commercial workloads in simulation environments requires addressing issues such as their long run times, large memory and disk requirements, and the complexity involved in tuning them.

Our workload development methodology (similar to prior work [5]) entails reducing simulation times, validating speed-ups, and scaling down the applications (as necessary). To reduce simulation time, we use simulation *checkpoints* to store a snapshot of the memory and disks after a long warm-up period. We use a *transaction-based* methodology that increases the accuracy of shorter simulation runs by reducing the start and end transient effects. To construct these workloads, we first tune them on an existing hardware platform, and then we load the *exact disk images* into our full-system simulation environment. These generic workload development concerns are addressed in Section 2, and the specifics of each workload is described in Section 3. We describe the target system and simulation infrastructure in Section 4, and present the properties of our workloads in Section 5.

The results in Sections 6 and 7 show that non-determinism exists in tuned multi-threaded commercial workloads, and that neglecting this behavior can result in incorrect conclusions. This problem is almost universally ignored because most simulators (including ours) are deterministic: they produce the same result every time, given the same workload and system parameters. Unfortunately, small changes in system parameters can expose the inherent non-determinism of the workload. For example, consider reducing the L2 cache miss latency of a *base* system by one cycle to produce an *enhanced* system. Intuitively, this enhancement should improve performance. However, this small change may result in completely different execution paths (e.g., due to lock races or external event timing), possibly resulting in worse performance. We demonstrate that this problem does indeed occur, then describe a methodology that uses pseudo-random perturbations and standard statistical measurement techniques to address this issue.

2 Workload Development

This section addresses the methodology we used to develop our commercial workload benchmarks. First, we describe how we shorten simulation time using checkpoints and a transaction-based measurement methodology. Second, we describe the two-step process of setting up and validating applications on an existing machine, then porting the applications into our simulation environment. Third, we describe our method for scaling down the workloads for tractable simulation.

2.1 Reducing Simulation Run Times

Running end-to-end simulations of commercial workloads would result in prohibitive run-times, due to the slowdown present in system-level multiprocessor simulation. For example, the TPC-C specification requires

that the benchmark runs for at least two hours on a real multiprocessor system [21]. Using a uniprocessor to simulate TPC-C on a 16-processor system would take more than 133 days, assuming a 1600x slowdown (100x per-processor). To limit simulation time, we use our simulator's checkpoint facility to capture the architectural state of the simulated system at the end of a warm-up period (which can take days to complete). All timing runs then start from the exact same checkpoint.

The commercial workloads presented in this paper are *throughput-oriented*. To measure throughput in a real system, one counts the number of transactions completed in a fixed time interval. For example, the TPC-C benchmark specification measures performance by the number of transactions completed per minute (tpmC) [21]. To measure throughput in our simulation environment, we instrument these workloads to signal the simulator at the end of each transaction using a special instruction. We then measure the amount of time it takes to complete a fixed number of transactions.

However, cold-start transients occur when transactions are executed before the system has reached a steady-state condition (e.g., the database buffer pool does not contain a frequently accessed index page). End transients occur when most processors become idle while they wait for the last several transactions to complete. To minimize these effects we warm-up the system, then take a checkpoint. Starting from this point, we measure the (simulated) time to complete a specified number of transactions. For example, benchmarking N transactions means time is measured and all processors continue execution until the Nth transaction completes. Even as this transaction completes, other transactions are in flight. By simulating a sufficiently large number of transactions, we reduce the effect of these partially complete transactions on the throughput. As we define transactions on a per benchmark basis, the work necessary to complete one varies between benchmarks. Two transactions in the same benchmark may represent significantly different amount of work (e.g., the "New Order" and the "Order Status" transactions in the OTLP benchmark).

2.2 Setup and Validation of Workloads

We set up and tuned our applications on a real multiprocessor machine, with the goal of developing workloads with reasonable multiprocessor speedup, and measuring microarchitectural statistics to be used in comparison with our simulation results. Using real machines makes setup much faster than under simulation, and permits rapid performance testing with different parameters. Due to the complexity of commercial workloads, they require a considerable amount of tuning in order to have reasonable speed-ups and scale-ups on multiprocessor systems and to avoid load imbalance.

We used a number of techniques to evaluate the applications' speed-ups and scale-ups, including hardware counters and operating system utilities. During this process, we used a Sun E6000 machine with sixteen 248-MHz UltraSparcII processors, each with a 1 MB unified L2 cache. UltraSparc processors have hardware performance counters that can be used to measure architectural events on a per-processor basis. We measured workload characteristics using these counters and calculated the Instruction Per Cycle (IPC) and cache miss rates as observed on the real machine. We tuned the workloads by seeking maximum speed-ups, measured by wall-clock run times for benchmark runs on a limited number of processors. For this purpose, we used the Solaris *psrset* tool to restrict given processes to run only on a subset of the available processors. Figure 1 shows the speedup of our set of workloads when running on 1, 2, 4, 8, and 16 processors. We also used the Solaris tool *mpstat* to measure the fraction of time spent by each processor in user, kernel, idle, or I/O wait mode. These utilization statistics allowed us to detect load-imbalance problems and verify that all processors are being sufficiently utilized (with less than 10% idle or I/O wait time). After this validation step was complete, we imported the exact disk images into our full-system simulation environment.

2.3 Scaling Down Workloads

Full-size commercial workloads can have memory requirements of many gigabytes, and secondary storage requirements of several terabytes. These memory and disk requirements often stress execution on native systems, and it is not currently possible to simulate such large systems. Moreover, simulators usually run on host workstations that are much less powerful than server systems. For example, our studies are conducted on PC-based Linux systems that have a 32-bit virtual address space limit.

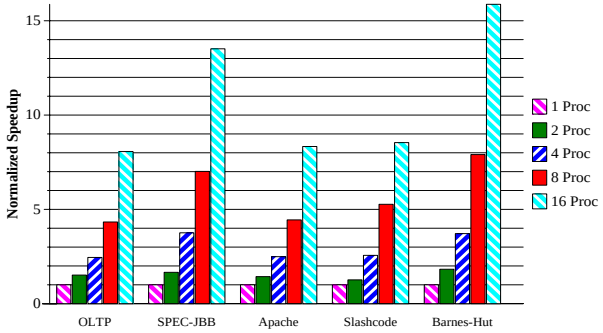

Figure 1. Workload parallel speedups

For these reasons, it is necessary to scale down the workloads so that they can be run in our simulation environment. We scaled down the workloads using trial and error to find the largest configuration of a workload that would run adequately in our simulator. Then we tested the performance of the scaled-down workloads on real hardware, to verify that the scaling has a minimal effect on workload behavior. For example, our OLTP benchmark (based on TPC-C) uses a 10-warehouse 1-GB database on five raw disks with a single log disk. Real TPC-C benchmark setups are normally two orders of magnitude larger in terms of the database size and number of disks used. Our setup on the Sun E6000 machine suffered a throughput penalty (in terms of tpmC numbers) of less than 30% compared to a 100-warehouse, 100 GB database constructed on 35 raw disks and 10 log disks. However, our simulation environment currently limits us to studying scaled-down versions of workloads.

3 Workloads

This section summarizes the set of workloads evaluated in this paper. This set includes two database on-line transaction processing applications, two web-server applications, and a scientific benchmark for comparison purposes.

3.1 OLTP

The TPC-C [22] benchmark models the database activity of a wholesale supplier, with many concurrent users performing business transactions against the database. The supplier operates out of a number of warehouses and their associated sales districts. The benchmark can be scaled by increasing the number of warehouses, but the database maintains fixed ratios of 10 sales districts per warehouse and 3000 customers per district. Transactions performed are of five transaction types, all related to the order-entry environment. Performance is measured by the number of "New Order" transactions performed per minute (tpmC), subject to certain constraints.

Our OLTP workload is based on the TPC-C v3.0 benchmark, but we scale down the data set. We use IBM's DB2 V7.2 EEE database management system and an IBM benchmark kit to build the database and model users. We build a 1 GB 10-warehouse database on five raw disks, and we use one additional dedicated disk for the database log. The TPC-C consistency requirements on the sizes of tables were maintained. We set the TPC-C client think time to be zero. We set the disk I/O latency in the simulator to be low and fixed (10 microseconds), emulating the high performance I/O subsystem in high-end servers.

We simulate 8 users per processor (e.g., 128 users on 16 processors), similar to Stets et al. [19]. Users are simulated using drivers from the IBM benchmark kit. A different process is started for each user. Each simulated user randomly executes transactions according to the TPC-C transaction mix specifications using a private random number generator. The database was warmed up by running for 10,000 transactions before taking measurements. Our results were based on runs of 1,000 transactions, unless otherwise specified. All completed transactions are measured, even those that do not satisfy some timing constraints in the original TPC-C benchmark specification.

3.2 SPECjbb

Java-based middleware applications are increasingly used in modern e-business infrastructure. SPECjbb is a Java program emulating a 3-tier system with emphasis on the middle tier. It fully implements the middle tier business logic. SPECjbb is inspired by the TPC-C benchmark and loosely follows the TPC-C specification for its schema, input generation, and transaction profile. SPECjbb runs in a single Java Virtual Machine (JVM) in which threads represent terminals in a warehouse. Each thread independently generates random input (tier 1 emulation) before calling transaction-specific business logic. The business logic operates on the data held in binary trees of java objects (tier 3 emulation). The specification states that the benchmark does no disk I/O or network I/O.

We used Sun's HotSpot 1.4.0 Server JVM and Solaris's native thread implementation. The system heap size was set to 1.8GB to avoid as much garbage collection as possible. Our experiments used 24 threads and 24 warehouses, with a data size of approximately 500 MB. The system was warmed up for 100,000 transactions, and our results are based on runs of 100,000 transactions.

3.3 Apache

Apache is a popular open-source web server used in many internet/intranet environments. In this benchmark, we focus on static web content serving.

We compiled Apache 1.3.19 on Solaris with GCC version 2.95.3. We made two compile-time changes to improve performance. First, we set Apache to use POSIX mutexes to serialize server processes waiting on `accept()`. Second, we set the dynamic module limit to zero, reducing the memory usage. We compiled and configured the Apache server according to the Apache group's performance notes [14].

We use the Scalable URL Request Generator (SURGE) [4] as the client. SURGE generates a sequence of static

URL requests which exhibit representative distributions for document popularity, document sizes, request sizes, temporal and spatial locality, and embedded document count. We ran 10 SURGE client threads per processor, and set the client think time to be zero.

Our experiments used a repository of 2000 files, with total size of approximately 50 MB, generated by SURGE using its default parameters. The system was warmed up for 80,000 transactions, and our results were based on runs of 2,500 transactions.

3.4 Slashcode

Dynamic web content serving has become increasingly important for web sites that serve large amount of information. Serving dynamic content is essential for online stores, instant news, and community message board systems. Our Slashcode benchmark is developed to represent these workloads.

Slashcode is an open-source dynamic web message posting system used by the popular slashdot.org message board system of the Linux user community. We used Slashcode 2.0, Apache 1.3.20, and Apache's mod_perl module 1.25 on the server side. MySQL 3.23.39 is used as the database engine. The server content is a snapshot from the slashcode.com site, and it contains approximately 3000 messages, with a total size of 5 MB. This benchmark is not database-oriented, so the size of the content has a small impact on system behavior. Most of the run time is spent on dynamic web page generation.

Autoslash is a multi-threaded user emulation program we developed to simulate user browsing and posting behavior. Each user independently and randomly generates browsing and posting requests to the server according to a transaction mix specification. There are 3 simulated users per processor. The system was warmed up for 240 transactions before taking measurements. Our results are based on runs of 50 transactions. Both server and client are compiled with Sun's WorkShop C 6.1 with aggressive optimization.

3.5 Barnes-Hut

For comparison, we selected one application from the SPLASH-2 [23] benchmark suite: Barnes-Hut with 64K bodies. The benchmark was compiled with Sun's WorkShop C 6.1 with profile-based feedback and uses the PARMACS shared-memory macros used by Artiaga et al. [3]. The macro library was modified to enable user-level synchronization through test-and-set locks rather than POSIX-thread library calls. We began measurement at the start of the parallel phase to avoid measuring initialization and thread forking.

4 Target System & Simulation Infrastructure

This section presents our simulation model of a target system and details of our simulation infrastructure.

4.1 Target System Model

We model a 16-node system similar to the Sun E10000 [7]. Each node contains a processor, caches, 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 single crossbar switch for the interconnection network to connect the nodes, with a delay of 50 ns for each interconnection network traversal (which includes wire propagation, synchronization, and routing). We selected 80 ns for memory 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 an integrated processor and first level cache model that would complete four billion instructions per second if the memory system beyond the L1 caches was perfect. This establishes the relative speed of the memory system with respect to the processors, and is representative of a 2 GHz processor with a perfect L2-cache that has an IPC of 2. L2 caches are modeled as being 4 MB 4-way set associative with 64-byte blocks. We assume a 1 GHz system clock, hence the system cycle time is 1 ns.

4.2 Full-System Simulation

We use Simics [1], a full-system multiprocessor simulator, to simulate the same commercial workloads we set up on the real hardware. Simics is a system-level architectural simulator developed by Virtutech AB that is capable of booting unmodified commercial operating systems and running unmodified applications. We configured Simics to model an E6000-like SPARC V9 target system running unmodified Solaris 8. Simics is only a functional simulator by default, but it supports extensions to model timing. We use Simics' functional processor model to model a simple blocking processor that executes all instructions in one simulated processor cycle. We assume a processor clock that is four times the frequency of the system clock (i.e., 4 GHz). We extended this simple timing model with a memory hierarchy simulator that accurately models memory reference timing.

4.3 Memory System Simulator

Our memory system simulator, *Ruby*, processes requests from Simics and blocks memory operations on cache

Table 1. Workload properties

Workload	Memory blocks touched (64 bytes)	Unique miss PCs	L2 cache misses per 1000 instructions	Supervisor misses (% of total)	Time Spent in Kernel (% of total)
OLTP	57 MB	12136	3.0	43%	28%
SPECjbb	353 MB	8163	3.2	15%	1%
Apache	102 MB	10214	2.9	82%	84%
Slashcode	173 MB	17009	1.1	48%	43%
Barnes-Hut	16 MB	3413	0.3	16%	3%

misses. A blocked Simics processor will complete no instructions until Ruby explicitly un-blocks the processor when the miss processing completes. This allows Ruby to capture timing-dependent race conditions and lock contention that cannot be captured using a trace-driven methodology. Ruby supports a broad range of coherence protocols, which are specified using our table-driven specification methodology [18]. The specification is codified using our domain specific language *SLICC* (Specification Language for Implementing Cache Coherence) and software tools to generate the C++ source code for the protocol state machines used in Ruby. Using this methodology, our simulations capture timing races and state transitions (including transient states) of the coherence protocols in cache and memory controllers.

5 Workload Properties

Table 1 presents some properties of our workloads from simulations of a 16-processor system. This table corroborates previous results by showing that our OTLP benchmark spends approximately one-quarter of its time in the kernel [5], and that commercial applications have significantly worse cache performance characteristics compared to the Barnes-Hut scientific benchmark.

In the following sections, we present non-deterministic effects that we observed in the OLTP benchmark. The four commercial workloads studied in this paper exhibit these effects to different extents. Ranked from most variable to least variable, the commercial workloads are slashcode, OLTP, Apache, and SPECjbb. We chose to present OLTP's behavior as representative of these workloads.

6 Commercial Workload Non-determinism

When measuring real systems, researchers normally make multiple measurements and use standard statistical techniques to ensure statistically significant results [2]. The goal of this methodology is to ensure that uncontrolled experimental factors do not lead to false conclusions. For example, if the paging daemon runs in one execution but not in any others, using this methodology should isolate this non-systematic effect.

Conversely, when (most) researchers perform simulation studies, they (implicitly) assume that *all* experimental factors are controlled and present results from a *single* simulation run. This assumption seems plausible, since a deterministic simulator will always return the same result for a given combination of workload and system parameters. However, just because a simulator is deterministic does not mean that the workload is also deterministic. While a single-threaded, user-level application running on a uniprocessor may be deterministic, a multi-threaded commercial workload on a multiprocessor is not.

Workload non-determinism can be caused by small timing variations that cause the application or operating system to take different execution paths. On a uniprocessor, a small difference in the arrival time of a device interrupt may result in a radically different operating system scheduling decision. On a multiprocessor, synchronization races may cause the application or operating system to acquire locks in different orders. The outcome of memory races and scheduling decisions potentially leads to divergent executions, which may yield widely varying results for the simulated execution time after the completion of the same number of transactions, or may even execute different mixes of transactions. Since the amount of work required to complete a transaction in a given workload can vary, executing a different mix of transactions would likely result in different simulated execution times.

We performed an experiment that shows that small changes in memory latencies can affect process scheduling even in a deterministic uniprocessor simulation environment. We ran two OLTP simulations starting from the same checkpoint and artificially introduced instruction cache misses every 100 instructions. In the first run, these additional cache misses are introduced at instructions 0, 100, etc., while in the second they are introduced at instructions 50, 150, etc. While these two runs have the same cache miss rate, they report up to 9% dif-

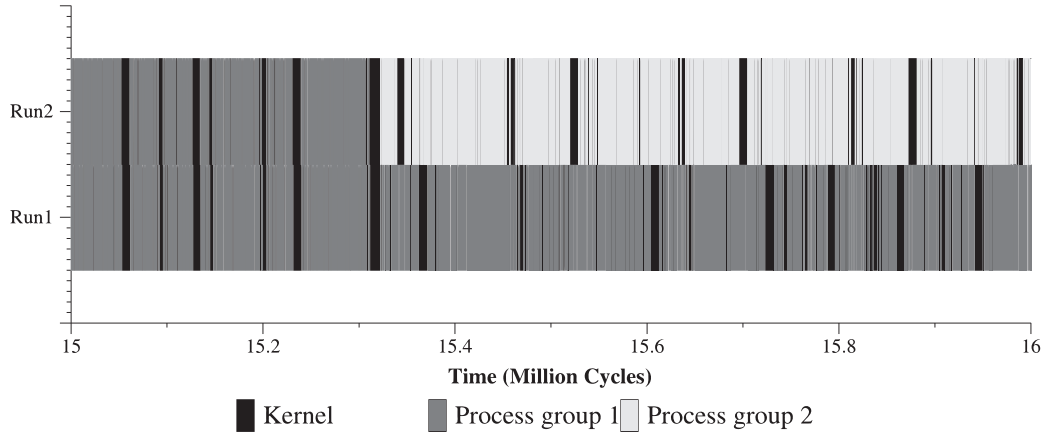

Figure 2. OS scheduling decisions are affected by memory latency

ference in simulated execution time after 2000 transactions. By instrumenting the simulator to report which processes are scheduled, we were able to show that the OS makes different scheduling decisions. Figure 2 shows a snapshot of the execution of these two runs. Run1 shows the execution of a process in process group 1, alternating between executing in kernel mode (black) or user mode (dark grey). In this same interval for Run2, the OS swaps out process group 1 and schedules a process of process group 2 (light grey). This snapshot shows the interval in which the initial divergence occurred. Both runs scheduled the same process groups prior to this snapshot, but the scheduling decisions were completely different beyond this point of divergence.

To mitigate the effects introduced by non-determinism, we run multiple simulations for each particular hardware configuration, and use their mean simulated execution time as our performance metric. We illustrate below how this methodology can be used to effectively separate systematic improvements from random effects caused by non-determinism.

Figure 3 presents statistics of the simulated execution time, gathered from a set of fifteen 1000-transaction OLTP runs on two different cache configurations. For each configuration, the left and right columns show the minimum and maximum (simulated) runtime, respectively. The center columns show the mean, with error bars indicating the standard deviation. Each run begins from the same checkpoint, but the simulator randomly introduces small perturbations in the memory system. On each L2 cache miss, the latency is randomly increased by a uniform random number between zero and seven nanoseconds. This perturbation effectively increases the contention-free memory access time to 183.5 ns and 128.5 ns, for memory and cache-to-cache misses respectively. Since the OLTP workload misses in the L2 cache no more than four times per 1000 instructions, the worst-case variation in CPI should be no more than 4%. And since there are millions of misses in each run, the law of large numbers [2] suggests that the actual variation should be much less. However, this perturbation results in widely varying execution times, whose range is approximately 10% of the mean. The minimum execution time for the direct-mapped case is lower (better) than the maximum for the set-associative case. If a researcher performed only a single simulation run for each case, s/he might draw the erroneous conclusion that the direct-mapped cache performs better than the set-associative cache. By performing multiple runs with random perturbations, statistically significant results can be obtained, that is the enhanced configuration performs better than the base case.

7 Workload Variability

Our transaction-oriented methodology simulates a given workload for a fixed number of transactions. A trade-off is made between simulation time and accuracy. Longer simulations amortize cold-start effects (e.g., cold

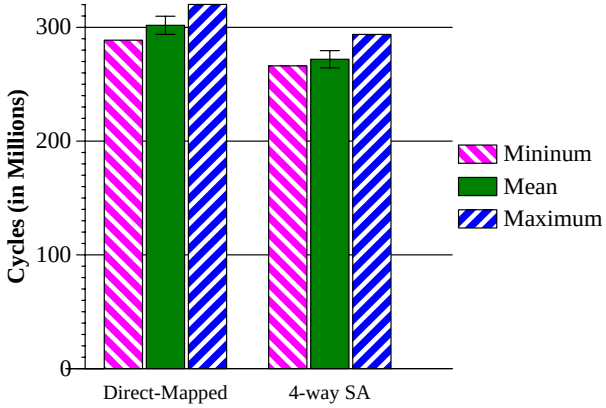

Figure 3. Execution time variations for two different cache configurations

Table 2. Some OLTP properties for different simulation lengths

Number of simulated transactions	200	400	600	800	1000	1200
System cycles per transaction	4.09	4.21	4.27	4.41	4.57	4.58
System cycles per instruction	0.78	0.74	0.72	0.70	0.66	0.66
L2 misses per 1000 instructions	3.79	3.50	3.39	3.23	2.99	2.98
L2 supervisor mode misses (%)	41.8	41.4	41.1	42.0	42.4	42.3
L2 misses per transaction (thousands)	19.78	19.99	20.11	20.48	20.70	20.73
Simulation runtime (hours)	1.92	3.88	5.89	8.02	10.18	12.23

caches) and smooth out variations due to heterogeneous transactions. To quantify this effect, we evaluated our workloads for different numbers of simulated transactions. Table 2 shows some architectural characteristics, computed from an average of twenty OLTP simulations on a 16-processor system. We had initially hoped that the runtime statistics would converge as we ran longer simulation. However, as shown in Figures 4, 5, and 6, even long simulations reveal that the workload exhibits different characteristics over time.

Using the OLTP workload, we simulated 8,000 transactions on a 16-processor system. These graphs show the average results of twenty separate runs, starting from the same checkpoint, measured every 200 transactions. The error bars indicate the standard deviations for each interval. Figure 4 shows there is variability during the run in the throughput of the system (the number of cycles to complete transactions). Figure 5 shows that the number of cache misses necessary to complete transactions is relatively stable across the run. Figure 6 shows there is considerable variability in the number of instructions executed to complete transactions in the OLTP benchmark.

Clearly a single short simulation run cannot capture the wide spectrum of the commercial workloads' behavior. Time-sampling is a well-known technique that may prove valuable to complete an architectural study within a reasonable simulation time [10, 12]. We intend to explore this further in future work.

8 Related Work

Prior work has studied commercial workloads for their architectural and micro-architectural characteristics, and has used them for simulation studies and for performance evaluations. The characterization studies can be classified by the level of detail in the processor model (in-order or out-of-order), their tools (real hardware or simulation), and the workloads studied. Our work distinguishes itself from these related works as it uses newer benchmarks, such as SPECjbb, larger configurations (in general), and studies these workloads' sensitivities to timing changes.

Barroso et al. [5] is an influential characterization study of commercial workloads on multiprocessors using both hardware counters, and in-order simulation tools. Ranganathan et al. [16] uses user-level out-of-order simulation to show that database workloads running on sequentially consistent systems can perform within 10-15% of release consistent systems. Keeton et al. [9] study the effects of out-of-order speculative execution on multiprocessor database workloads using hardware performance counters. Dedicated snooping hardware

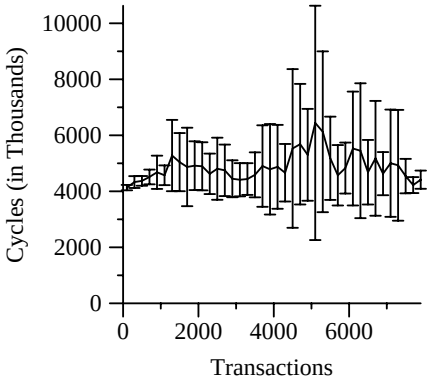

Figure 4. Cycles per transaction

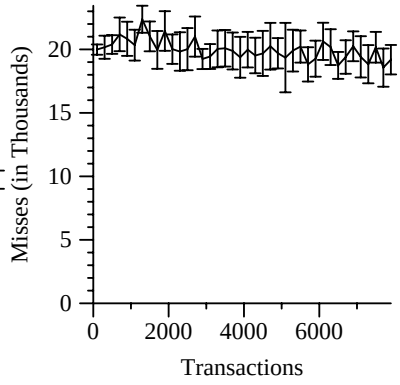

Figure 5. Misses per transaction

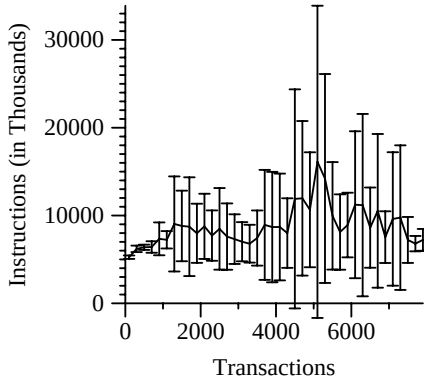

Figure 6. Instructions per transaction

has been used to study the memory system performance of commercial workloads running in real hardware systems [13]. The web-based on-line transaction processing benchmark TPC-W has also been studied in prior work [6]. The memory system performance of Decision Support System (DSS) workloads in multiprocessors have been characterized by Trancoso et al. [20].

In our simulation methodology we use an in-order processor model. Pai et al. [15] demonstrate that a key characteristic in shared-memory multiprocessor simulation is how the memory system is modelled. Durbhakula et al. [8] show this principle can be applied to approximate an out-of-order processor model using a simple processor model trading, achieving a significant speedup and introducing relatively little error.

9 Conclusions

This paper describes the methodology we adopt in developing four multiprocessor commercial workloads into benchmarks for simulation studies. In this methodology, we perform the setup of these workloads on real hardware for tuning and evaluation, then import them into our simulation environment. We provide a detailed explanation of our workloads and our simulation environment. We present some speed-up measurements of the tuned commercial workloads on the real hardware, and workload-specific tuning issues.

In order to deal with the non-determinism present in multiprocessor simulations, we proposed a transaction-based approach for simulation runs, that helps decrease cold-start and end transient effects. We demonstrate that even a fully deterministic simulation of a uniprocessor can exhibit non-deterministic behavior due to OS scheduling. We present how an incorrect conclusion can be reached if the issue of non-determinism is ignored. We demonstrate that commercial applications can exhibit different characteristics, and that these characteristics could lead to a large variation in simulation results. We propose a methodology for coping with this non-determinism by introducing minor perturbations in memory system latencies, and running multiple simulations for the same hardware configuration. This approach can lead to statistically significant results for these non-deterministic workloads.

Acknowledgements

This work is supported in part by the National Science Foundation with grants EIA-9971256, CDA-9623632, and CCR-0105721, an IBM Graduate Fellowship (Martin), an Intel Graduate Fellowship (Sorin), two Wisconsin Romnes Fellowships (Hill and Wood), and donations from Compaq Computer Corporation, IBM, Intel, and Sun Microsystems.

References

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

[2] Arnold O. Allen. *Probability, Statistics, and Queueing Theory.* Academic Press, second edition, 1990.

[3] Ernest Artiaga, Nacho Navarro, Xavier Martorell, and Yolanda Becerra. Implementing PARMACS Macros for Shared Memory Multiprocessor Environments. Technical report, Polytechnic University of Catalunya, Department of Computer Architecture Technical Report UPC-DAC-1997-07, January 1997.

[4] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of the 1998 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 151–160, June 1998.

[5] 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*, pages 3–14, June 1998.

[6] Harold W. Cain, Ravi Rajwar, Morris Marden, and Mikko H. Lipasti. An Architectural Evaluation of Java TPC-W. In *Proceedings of the Seventh IEEE Symposium on High-Performance Computer Architecture*, pages 229–240, January 2001.

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

[8] Murthy Durbhakula, Vijay S. Pai, and Sarita V. Adve. Improving the Accuracy vs. Speed Tradeoff for Simulating Shared-Memory Multiprocessors with ILP Processors. In *Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture*, pages 23–32, January 1999.

[9] 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*, pages 15–26, June 1998.

[10] 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.

[11] Peter S. Magnusson et al. SimICS/sun4m: A Virtual Workstation. In *Proceedings of Usenix Annual Technical Conference*, June 1998.

[12] Margaret Martonosi, Anoop Gupta, and Thomas Anderson. Effectiveness of Trace Sampling for Performance Debugging Tools. In *Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 248–259, May 1993.

[13] 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.

[14] Apache Performance Notes. http://httpd.apache.org/docs/misc/perf-tuning.html.

[15] Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve. The Impact of Instruction-Level Parallelism on Multiprocessor Performance and Simulation Methodology. In *Proceedings of the Third IEEE Symposium on High-Performance Computer Architecture*, pages 72–83, February 1997.

[16] Parthasarathy Ranganathan, Kourosh Gharachorloo, Sarita Adve, and Luis Barroso. Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 307–318, October 1998.

[17] Mendel Rosenblum, Stephen A. Herrod, Emmett Witchel, and Anoop Gupta. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel and Distributed Technology: Systems and Applications*, 3(4):34–43, 1995.

[18] Daniel J. Sorin, Manoj Plakal, Mark D. Hill, Anne E. Condon, Milo M.K. Martin, and David A. Wood. Specifying and Verifying a Broadcast and a Multicast Snooping Cache Coherence Protocol. *IEEE Transactions on Parallel and Distributed Systems*, To appear.

[19] Robert Stets, Luiz Andre Barroso, Kourosh Gharachorloo, and Ben Verghese. A Detailed Comparison of TPC-C versus TPC-B. In *Third Workshop on Computer Architecture Evaluation Using Commercial Workloads in conjunction with HPCA-6*, January 2000.

[20] Pedro Trancoso, Josep-L. Larriba Pey, Zheng Zhang, and Josep Torrellas. The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors. In *Proceedings of the Third IEEE Symposium on High-Performance Computer Architecture*, pages 250–260, February 1997.

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

[22] Transaction Processing Performance Council. TPC Benchmark C, Draft Specification, Revision 4.0.q, August 1999.

[23] 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*, pages 24–37, June 1995.