

The RASE (Rapid, Accurate Simulation Environment) for Chip Multiprocessors

John D. Davis, Cong Fu, James Laudon
Sun Microsystems
{john.d.davis, cong.fu, james.laudon}@sun.com

Abstract

We present RASE, a full system high performance simulation methodology for simulating complex server applications and server class chip multiprocessors enabled with fine-grain multithreading (CMTs). RASE combines application knowledge, operating system information, and data access patterns with an instruction stream from a highly-tuned, scalable steady-state benchmark [5][22] to generate multiple representative instruction streams that can be mapped to a variety of CMT configurations. We use execution-driven simulation to generate instruction streams for M processors and store them as instruction trace files (several billion instructions per processor) that can be post-processed and augmented for larger than M processor system simulation. We use SPEC JBB2000, TPC-C, and an XML server benchmark to compare the performance estimates of RASE to a reference prototype CMT system. By varying M, we find that our trace-driven simulation methodology predicts within 5% of the instructions per cycle (IPC) of the reference hardware for the applications. Without post-processing the traces, in the best cases, the performance prediction accuracy degrades to 20-40% of the real IPC for instruction traces that require a high replication factor.

1. Introduction

Silicon process technology scaling has been both a blessing and curse for computer architects. Moore's law enables computer architects to conjure complicated designs and systems-on-a-chip (SOC) as a result of an seemingly unlimited supply of transistors. Likewise, software engineers have been able to harness the copious resources available to them creating ever larger and more complicated applications. Unfortunately, the increased system size, complex software and hardware interactions, and multiple processors have made performance estimation and system validation more complex. Computer architects cannot decouple the cache hierarchy or other important on-chip or off-chip peripheral devices from the processor infrastructure. Current and future processor architectures require all aspects of the system working in concert in a simulation environment, to produce accurate performance estimates. This is an even more challenging process when developing a new architecture without readily available software, operating system or applications support, or similar hardware to leverage for a simulation environment. Finally, computer architects must possess the ability to evaluate design trade-offs in

a reasonable time frame, which is complicated by chip multiprocessor designs that have linear run-time increases on single-threaded system simulators.

This paper presents RASE, a Rapid, Accurate Simulation Environment for simulating fine-grain multithreaded chip multiprocessors (CMTs) running commercial applications. First, the commercial application is examined to make sure it scales in a nearly linear fashion across the range of system sizes that will be explored. Several large commercial server applications such as Spec JBB2000, TPC-C, and XML Test scale in this fashion [24][26][25]. Second, execution-driven simulation of the actual, full-scale application running on the actual, full-scale system is performed for several application/system sizes, where the size of the largest application and system configuration simulated is limited by the speed of the execution-driven simulation and the patience of the architect (our patience ran out at two months of wall-clock time for the largest configuration). Next, a very detailed comparison is performed between the simulated performance and the actual performance. Extensive effort is taken to validate the real reference systems against their simulated counterparts. This validation is time consuming, but once completed, provides the foundation for being able to quickly and accurately evaluate complex architectural tradeoffs. In the final step, very long instruction traces are generated from the M processors in the largest application and system configuration. Combining the instruction stream, stored as a trace, with application, operating system, and memory information, provide the fundamental information for scaling the application. Because these applications scale linearly and have homogeneous behavior, the traces can be used to rapidly evaluate systems of up to M processors. Furthermore, for these steady-state applications [5][22], we have developed a trace post-processing technique that uses the application specific information and memory information to replicate traces for $N > M$ traces for larger systems. Instead of scaling the application down to fit the simulation environment, we use a simulation infrastructure that can artificially scale to larger systems by estimating the requirements of these systems before and extending the application characteristics during simulation. This modified trace-driven simulation technique is fast, but has the drawback of executing an instruction stream that is potentially not a legal ordering of the actual application. As we will show in this paper, the errors introduced by using trace-driven simulation are small, and as a result,

performance of the actual application on the actual hardware matches the performance of the traces on the simulated hardware, even though the system we are simulating is very different from the system that was used to generate and validate the traces.

We present an evaluation of RASE, a high performance multi-configuration simulation technique for a CMT. Our company leverages the Simics [19] simulation environment and adds a proprietary processor and memory system simulator for a family of CMTs. We compare the results from a Niagara instantiation of RASE to a real Niagara processor running the same applications. In the next section, we present an overview of the processor and simulation infrastructure and techniques. Section 3 presents the simulator validation methodology, followed by the benchmark details in Section 4. Section 5 presents the benchmark results comparison of the real and simulated systems for SPEC JBB2000, XML Test, and TPC-C results. Related work, conclusions and future work are given in the final sections.

2. The Hardware and Simulator

Rapid full system simulation is required to make accurate performance predictions to quantify design trade-offs. With ever increasing system complexity, accurate peripheral (memory, disk, etc.) device simulation can impact the performance predictions as much as the processor model. We briefly describe a chip multiprocessor with fine-grain multithreading (CMT) targeting server class applications. Before the system was available, a simulation environment was required that could provide accurate performance estimates for architecture trade-offs within a reasonable timeframe. We developed RASE, a Rapid, Accurate Simulation Environment that combines custom processor and memory simulation modules with the devices and simulation infrastructure from the Simics full system simulator [19]. This section describes the reference hardware, simulator, and related simulation methodology.

2.1. The Hardware

Sun Microsystems has developed Niagara, a 32-way multithreaded SPARC® processor [1]. This processor exploits 32 threads, a high bandwidth memory system, and is targeting commercial server workloads [1]. Figure 1 shows the die photo of the Niagara processor. Threads are clustered into groups of 4 and share a single-issue integer datapath pipeline and related resources (IDP), caches, and TLBs. Each thread maintains a unique register state and exclusive sets of instruction and store buffers. Niagara has a six-stage pipeline, adding a thread selection stage to the canonical five-stage pipeline: Instruction Fetch, Thread Selection, Decode, Execute, Memory, and Writeback. Each thread can be logically thought of as a conventional processor thread and this paper uses the term processor and thread synonymously.

2.2. The Simulator

The combination of complex applications and large

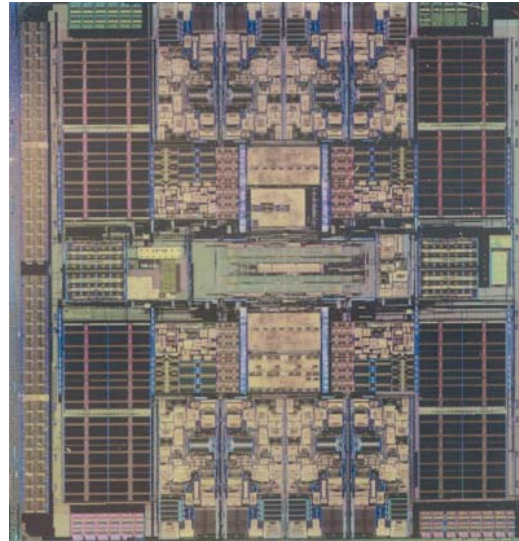


Figure 1: Niagara, a 32-way multithreaded (MT) SPARC® processor with fine-grain multithreading supporting 4 threads per core, exclusive primary data and instruction cache, a shared second-level cache, memory controllers, I/O subsystems, and other subsystems.

full system simulation make the long simulation run time of execution-driven simulation unrealistic for many design trade-off studies. We have developed a detailed performance simulator called SimCMT. SimCMT is a cycle-based performance simulator that models most of the micro-architecture details of Niagara: the threads, processor cores, pipelines, Level 1 and 2 (L1/L2) caches, crossbar, DRAM controllers and various buffers and queues. All blocks and their related parameters can be configured at both compile time and run-time, enabling large design space searches [10]. SimCMT also supports multiple clock domain simulation enabling single-simulator full system simulation of system-on-a-chip (SOC) components like the DRAM controllers.

Another important feature of SimCMT is the built-in statistics module in each block. This module periodically dumps out raw statistics about the specific block, e.g., some of the statistics for each L2 bank include total number of accesses, total number of misses, total number of write-backs, and number of misses that hit on the fill queue. For each miss request, multiple time stamps are recorded to indicate how many cycles are spent in the crossbar, L2 pipeline, miss queues, and DRAM, providing the full round trip time. The raw statistics are then post-processed to generate a more readable format. These statistics have proven to be vital in analyzing and understanding the results from the simulations.

SimCMT simulation can be run in two modes. One mode is to run with an external instruction simulator such as Simics, which provides the instruction and data references to SimCMT. SimCMT in turn feeds timing information in terms of stall cycles back to Simics. This execution driven mode has the advantage of accurate reference traces that vary along with the architecture changes. The down side of the approach is the

extremely long simulation time. Another mode is to run with an instruction trace that is collected offline from a system simulator. The traces include complete system activities ranging from instruction traces (including data accesses), traps, TLB records, processor state changes, Direct Memory Accesses (DMAs), and register values. One advantage of the trace-driven mode is that the trace needs to be collected only once and can be reused. We simulate the full processor pipeline and memory system with the instruction traces. The speed-up of trace-driven simulation compared to execution-driven simulation is about 20X. Another advantage is that the same trace provides a constant input to the simulator when several architectures are being simulated and evaluated. This reduces the variability in the comparisons and the analysis can be more isolated and focused on the architecture differences.

SimCMT has been extensively used throughout the duration of the design and implementation phase of Niagara. A wide range of (micro-)architecture variables such as number of cores, number of pipelines per core, number of threads per pipeline, cache sizes, line sizes, associativity, replacement strategies, and various buffer/queue sizes have been evaluated and finalized through extensive simulations. SimCMT's combination of the highly detailed simulation infrastructure and the highly detailed raw statistics provide the performance evaluation platform for quick and accurate architecture evaluation and understanding.

2.3. Instruction Stream Generation

Figure 2 illustrates the instruction stream generation and simulation flow. We start by setting up these workloads on a system simulator, such as Simics [19], with a configuration that has the proper resource requirements based on the targeted performance including the following resources: the number of CPUs, the number of hard drives, RAM, and selection of operating system and third-party applications. (1) Simics, running in fast mode, is used to execute the benchmark until it reaches steady state, where a checkpoint is taken (2). After the checkpoint, we can plug in Niagara specific models for an execution-driven simulation or generate a trace that can be later used for trace-driven simulation. Given a valid trace file, a standalone SimCMT can be used in trace-driven mode (3) to evaluate the system performance. Regardless of the simulation mode, a common raw output can be analyzed (4) and the results can be used to modify the system model and repeat the simulation methodology (5), either execution or trace driven.

Steps 1 and 2 generate instruction streams that are representative of the original workload across various hardware configurations, making them suitable for a variety of targeted architectures. We created a Simics module to generate trace files by writing every PC and corresponding instruction to a file for each processor or thread. Branch prediction, physical address to virtual address mappings, kernel vs. user code and other related information about thread state is written to this file and compressed to save disk space. We also use *pmap* to get address space information about the processes. This defines the shared and

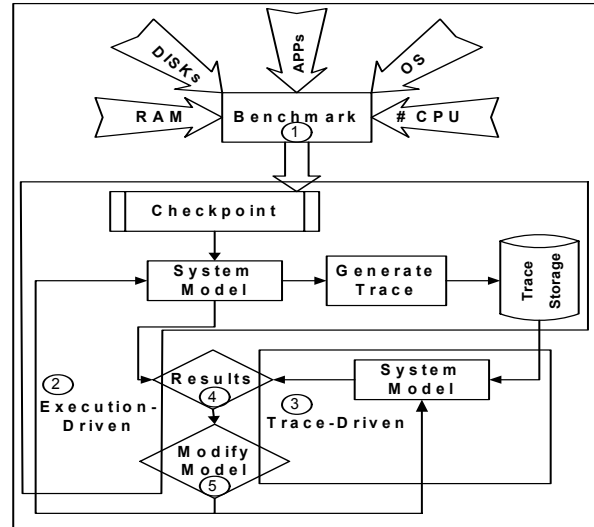


Figure 2. The steps required for trace generation and execution or trace driven simulation. (1) Tune the benchmark for the target hardware, (2) use execution-driven mode to reach the benchmark steady state, take a checkpoint, continue execution-driven simulation and/or generate trace file, (3) trace-driven simulation using input trace file and system model, (4) common result analysis framework, and (5) model modifications.

non-shared memory regions to be used for thread replication for larger subsequent simulations. These trace files therefore capture the memory access behavior and the instruction mix of the commercial server benchmarks used for the architecture studies.

2.4. Scaling Large Simulations

Niagara has 32 threads, which exceeds the number of threads in our instruction stream file, because the largest applications we decided to simulate in a reasonable amount of wall clock time contained only 16 threads. Generating larger trace files would be prohibitive due to time to set up, reach the benchmark's steady state operating region, and collect the trace. In order to replicate threads to enable scaling the industry-grade benchmark to simulate high aggregate thread configurations, we use the address mapping of the processes. We assume text is shared, as well as the kernel data structures. The application address space is divided into shared and non-shared region based on the *pmap* info and some application specific information, i.e., the system global area (SGA) for Oracle or the JVM heap for SPEC JBB2000. We use this benchmark specific information to "remap" old threads to new threads with the same benchmark behavior at different points in the processor trace file. The long trace files enable non-overlapping instruction execution of the original threads and replicated threads, as shown in Figure 3. These steady-state benchmarks [5][22] do the same classes of transactions repetitively and the replicated threads capture that same general benchmark behavior.

In general, an X -way trace file can be expanded to simulate Y threads, when $Y > X$, by replicating and shifting the address space of the trace threads. The addresses of the non-shared address space, like the stack, were simply shifted to remove replication-induced artificial sharing, while the shared regions, e.g., text and shared global memory, were not shifted. We shift the upper address bits by Xoring them with the replication id and maintaining the correct physical to virtual address mappings. The replication id ranges from 0 to the ceiling of Y/X . The cpu or thread ids are distinguished by the offset pointers into the original instruction stream file. We insure non-overlapping execution of the replicated threads by selecting an appropriate offset.

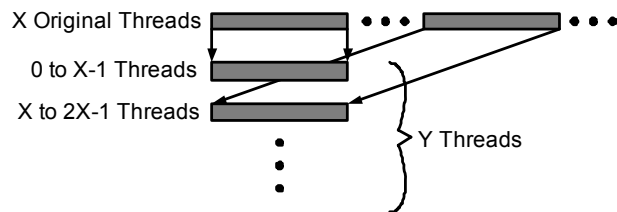


Figure 3. Thread replication from a trace with X threads for non-overlapping trace-driven execution of Y threads.

Thread replication has the potential to lead to benchmark behavior that is not characteristic of the benchmark. This could be a result of benchmark variability, lock contention, or other software and benchmark setup problems. However, we did not observe time variability, as described in [3], as these benchmarks exhibit a steady behavior during their large measurement window [5][22]. Furthermore, space variability exists between runs on the same systems, but we are able to reproduce benchmark performance within a few percent of the average for multiple runs of these highly tuned benchmarks. As our data demonstrates, CMT performance on the real applications can be better characterized by our RASE methodology combined with detailed hardware/software correlation and long traces than by an alternative methodology of application scaling and multiple short execution-driven runs [2][3].

When using thread replication to simulate larger systems, we are trying to balance constructive and destructive interference in the simulation, in this case cause by false sharing and increased conflict misses. These effects should be present in these types of simulations, but we want to mitigate the artificial exaggeration of these effects caused by thread replication. We preserve shared address spaces to remove any false conflict misses. However, conflict misses are introduced by shifting the non-shared memory region while still preserving all the low order bits. This causes more conflict misses and in some cases, the simple replication mechanism increases conflict misses in the L2 cache resulting from the lower order bits of the replicated threads mapping to the same set index. A simple index hashing scheme is used that hashes the replication id with the set index to reduce the conflict miss rate, thereby balancing the performance degradation introduced by the increased conflict misses

due to address shifting. This basically replicates the page coloring that the operating system would normally do.

3. Benchmark Details

To estimate real application performance, we have selected SPEC JBB2000, TPC-C, and XML Test server benchmarks to assess the CMT's performance. SPEC JBB2000 emulates a 3-tier system emphasizing the Java server-side performance. This benchmark focuses on middleware business logic [24]. TPC-C is an online transaction processing benchmark based on an order-entry system [26]. We concentrate on the server component of TPC-C for this study. This complicated benchmark can have extreme resource requirements [16][14][4][1][23]. XML Test is a multithreaded XML processing benchmark developed at Sun Microsystems [25]. XML Test performs both streaming and tree building parsing, which replicate application servers that provide web services and simultaneously process XML documents. Unlike SPEC JBB2000, XML Test is a single tier system benchmark; the test driver is part of the worker thread.

All of these benchmarks lack multiple phase execution, thus recording the contiguous instruction streams on a per thread basis captures the complete system performance and the overall benchmark characteristics. In contrast, benchmarks like SPEC CPU2000 require sampling techniques to capture the various phases of execution [21]. SPEC JBB2000 uses the J2SE 1.5 JVM with a 3.5 GB heap running on Solaris 9 with 2 to 16 warehouses to collect a 2 to 16-processor instruction trace file, respectively. Likewise, XML Test uses the J2SE 1.5 JVM, but with a smaller 2.5 GB heap. For TPC-C, we use 3,000 warehouses with a 28 GB SGA and 176 9 GB disks coupled with commercial database management and volume manager software running on Solaris 9. Both clients and servers are simulated, but only the server instruction traces are used in this study.

Each trace contains several billion instructions per process thread, or over 30 billion instructions in aggregate in steady state. All traces are collected during the valid measurement time after the benchmarks have ramped up and completed the benchmark specified warm-up cycle, on real hardware. This is 2 minutes of wall clock time for SPEC JBB2000, several million transactions for TPC-C, and 10 minutes for XML Test. We have observed significant variation in benchmark performance during the ramp-up period, but little variation once in steady state as mentioned in [3]. All benchmarks are highly tuned with less than 1% system idle time, and show negligible variability during the measurement period with respect to performance and instruction mix as described in Section 4.1.

4. RASE Validation

RASE validation can be broken up into three parts, the instruction stream files, thread replication, and the simulator, SimCMT. First, we validate the trace files by comparing steady-state execution of the full-sized

application on a large-scale commercial system to the same full-sized application running on a SimCMT model of the large-scale commercial system. Second, by using multiple configurations of the same benchmark, we are able to generate multiple instruction stream files and compare the thread replication methodology. Furthermore, we split the largest instruction stream file with 16 threads into multiple smaller instruction stream files that were then replicated to compare to the original non-replicated simulation results. Finally, we validated the SimCMT model of the Niagara system by comparing the performance of the SimCMT model against the actual Niagara RTL. The highly detailed system model is parameterized and can easily evolve with the processor design. With each major change of the Niagara architecture, we re-evaluated the SimCMT model to make sure it continued to match the performance of the Niagara RTL.

4.1. Trace Validation

Extensive effort was performed to validate the workload setups and the traces generated from them. The full-size workloads were run on a large SMP system as well as on the SimCMT model of that system. The first degree of validation compared system hardware counters to simulated counters for primary and secondary cache misses and instruction mixes. In addition, comparisons of TLB miss rates, context switch frequencies, system calls, and intervals between contexts, to name a few metrics, were performed to make sure the real and simulated systems matched at the workload level before starting any SimCMT simulation.

We have observed approximately 1% difference in IPC when comparing execution-driven and trace-driven full system simulations for the same CMP model. As expected, miss rates for the two methodologies differ more, but we have not observed a relative difference greater than 5%. We also investigated the instruction mix by process or thread basis over a range of system configurations, from 2 threads up to 16 threads, for the benchmarks. This decomposes each configuration (2, 4, 8, or 16 threads) into its constituent group of single threads. We analyzed the instruction mix of each thread for each configuration and compared it to the aggregate average, all threads for all configurations (2+4+8+16 threads or the combined 30 total threads). Figure 4 shows this aggregate average instruction mix \pm one standard deviation compared to the average instruction mix (ALU, conditional branches, unconditional branches, loads, stores, and overall user-level instructions) for the 4 threads, 8 threads, and 16 threads in a 4, 8, and 16 processor configuration, respectively, running TPC-C.

When examining the TPC-C's instruction mix, we noticed that most of the processor instruction streams were consistent, especially with regard to the percentage of load and store instructions. This instruction mix consistency is important because the loads and stores act as performance throttles via cache misses and page faults. Figure 4 illustrates the much larger standard deviation due to a few instructions streams with more kernel instructions compared to the majority of the instruction streams with more user-level instructions. This is due to tuning TPC-C to bind the interrupt handling routines to a few processors.

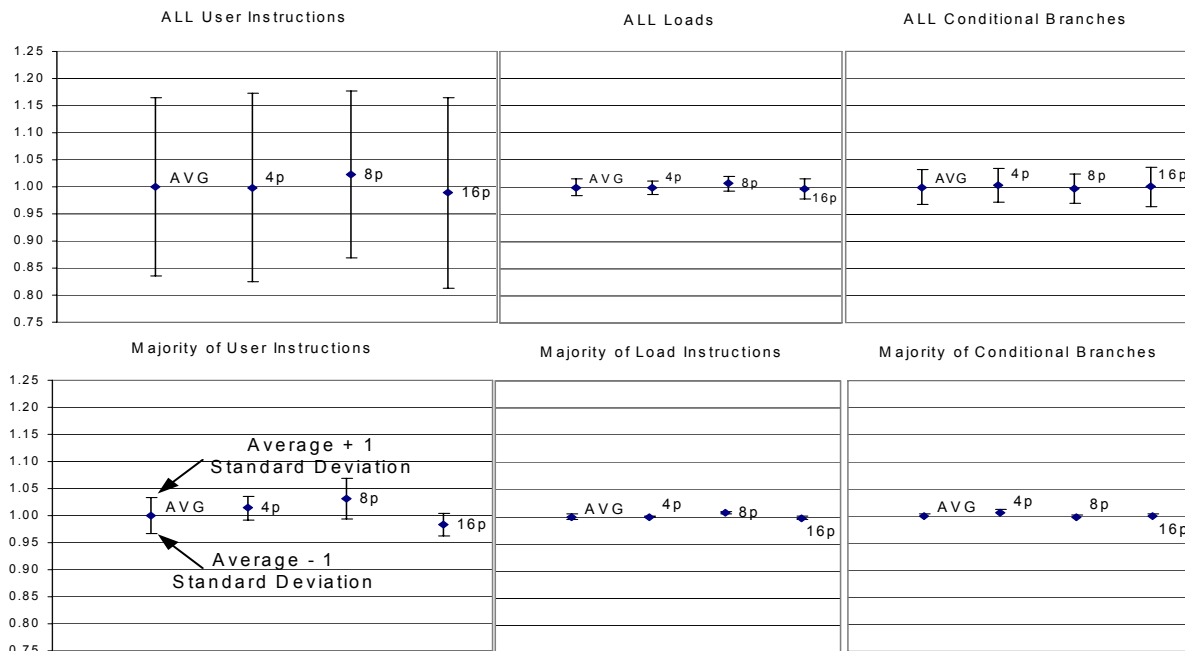


Figure 4. Each graph compares the overall average (AVG) \pm 1 standard deviation instruction mix component with the average for a particular TPC-C configuration for 4, 8, and 16 processor from the execution-driven simulation that generates the trace files. The top set of graphs is for all TPC-C instruction streams. The bottom set of graphs accounts for 3, 6, and 14 of the instruction streams for 4, 8, and 16 instruction streams total, respectively.

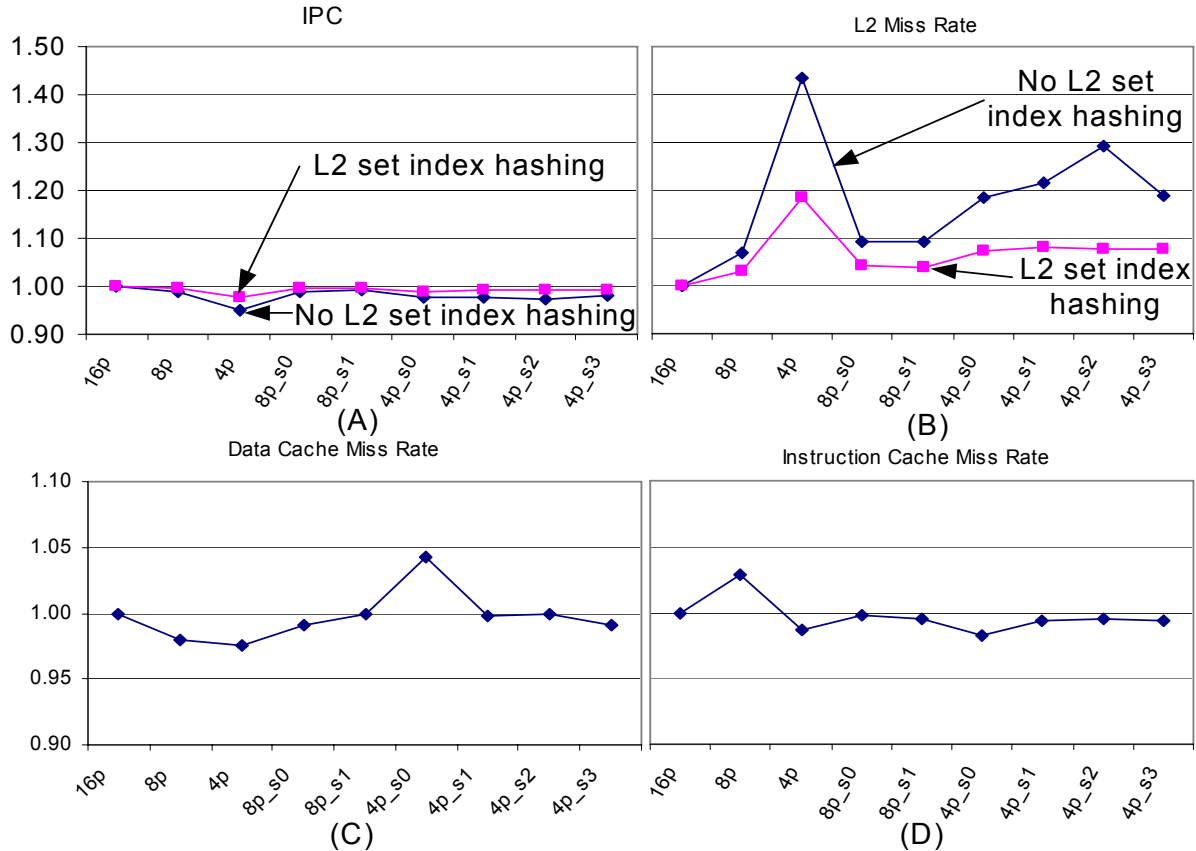


Figure 5. Using a 16 thread CMP, a 16 thread execution driven simulation is compared to thread replication used for smaller instruction streams with normalized results from 8 threads, 4 threads and the 16 thread instruction stream split into 2 8 thread and 4 4thread instruction streams. This illustrates the impact thread replication has on the IPC (A), L2 miss rate (B), data cache miss rate (C), and instruction cache miss rate (D).

This thread decomposition was another way to verify that the instruction streams for the benchmark scaled and that the scaled versions were optimized. On average, across all 30 processor/thread instruction streams for SPEC JBB2000 (threads from all 2p, 4p, 8p, 16p configurations), the user level instructions accounted for over 98% of all instructions in each individual stream for these trace files. Likewise, the overall instruction mix was constant as the highly tuned application scaled. The same observation held true for XML Test, another Java based benchmark that can scale linearly with system size. For both of these benchmarks, the instruction mix variation was less than 5% compared to the overall average.

4.2. Thread Replication Validation

Predicting the performance of large-scale systems is very difficult because of the complexity of the benchmarks and the time required to generate results using execution driven methods. Using the three instruction streams generated from 4 to 16 processor SPEC JBB2000 configurations, we evaluated thread replication using the same CMP model, 4 processors with each processor supporting 4 threads. We used the 16 thread execution driven results and compared the IPC, instruction and data cache miss rate and L2 cache

miss rate to trace-driven simulation using the instruction streams derived from an 8 thread, 4 thread, and the 16 thread instruction stream split into 2-8 thread (8p_s0 and 8p_s1) and 4-4 thread (4p_s0, 4p_s1, 4p_s2, and 4p_s3) instruction streams. Figure 5 (B) shows the L2 miss rate normalized to the 16 thread simulation results. As the number of threads in the trace file decreases, the cache miss rate increases by more than 40% compared to the execution-driven simulation due to increased conflict misses. Furthermore, even taking the 16 thread instruction stream file and decomposing it into smaller instruction stream files is detrimental to the L2 miss rate, but not as bad when comparing the 16 thread instruction stream decomposed in to 4-4 thread streams (4P_s*) with a normal 4 thread stream (4p). The bottom line in Figure 5 (B) illustrates how a simple L2 set index hashing scheme can reduce the L2 cache miss rate, thereby enabling the performance prediction of small instruction streams to approximate the L2 cache behavior of large instruction trace files.

Primary caches and IPC are less affected by the replication-introduced errors, with the latter having the largest effect. Figure 5 shows the variation in the normalized IPC (A) and primary cache miss rates (C) and (D). In this case, the top line in Figure 5 (A) shows the IPC improvement when using the L2 set index hashing. Multithreading in the CMP core helps hide the

long latency effects of cache misses thereby reducing the IPC degradation due to L2 cache misses. The bottom line in Figure 5 (A) shows the IPC without using this post-processing technique on the instruction streams.

As shown in Figure 5, thread replication can introduce significant performance prediction inaccuracy for highly replicated instruction streams in the L2 miss rate. The addition of set index hashing further improves the accuracy of the trace-driven simulation results versus the execution driven simulation results. Even without the use of set index hashing, the CMP performance predictions are within 5% for larger replication factors.

4.3. Simulator vs. RTL Validation

Multiple stages of validation have been performed to guarantee SimCMT's accuracy. There were three main methods used for validation, ranging from the use of small, directed diagnostics, cycle-by-cycle comparison to the hardware RTL, and waveform comparisons. As the micro-architecture specification developed and evolved, so did SimCMT. By using small, directed diagnostics, short instruction sequence snippets, various parts of SimCMT were validated by examining the cycle-by-cycle debugging log. Comparing SimCMT's output with the micro-architecture specification provided the first stage of validation.

The second stage of validation started when most of the RTL code became available. More complicated performance diagnostics that target different blocks were run in both RTL simulation and SimCMT simulation. The log files from both simulations were compared side by side using comparison tools. The coarsest metric used for comparison was the total number of cycles executed. These code segments ranged from an average of 15 cycles to well over 100,000 cycles per iteration. We investigated all large differences between the RTL and SimCMT to determine whether RTL or SimCMT was not conforming to the specification. This process also acted as a safeguard for checking if the implementation was meeting the specification. In cases where SimCMT was deviating from the RTL model, SimCMT parameters could be adjusted to match the RTL.

Using this iterative process, we were able to match RTL within 2% on diagnostics that were single threaded and rarely accessed DRAM. The DRAM accesses tended to have a slightly larger error margin because SimCMT did not model the periodic DRAM refresh. About 50% of the deviations between SimCMT and RTL were a result of "off by one" counting errors. This type of error was the easiest to fix and were a result of the pipeline stages or other events being miscounted. This was most common in the longer floating-point operations and acknowledgements for cache events like hits and misses in the L2 cache. The timing correlation was very complicated for the L2 cache because of the multiple units involved in the point-to-point communication between the core and the L2 cache and the number of outstanding misses. Finally, there were a number of situations related to some traps that required complex sequences before a

thread could resume. This resume cycle latency due to some traps was also a source of deviation between SimCMT and RTL.

However, overall the goal for all diagnostics was to reduce the absolute difference below 5% for single-threaded tests and between 5-10% for multi-threaded tests. To further validate the fine details, we also had the capability in SimCMT to dump the simulation state into a format that was readable by a waveform tool called debussy. This output could be incorporated into the RTL signal dump and shown in waveforms. This provided the finest grain of comparison for correlations. As the next section demonstrates, these performance differences in the directed code sequences did not dramatically impacted the accuracy of overall benchmark performance predictions.

5. Simulation vs. Real Hardware

The final step was to validate the trace-driven results versus real Niagara hardware. The Niagara processor is a dramatic deviation from previous Sun Microsystems processors. This required a new simulation environment that was vastly different from current processors or systems, making pre-silicon simulator validation difficult. However, as we will show later, the RASE system accurately predicts the performance that was observed on Niagara, even though the application traces used were generated on a standard SMP built from multiple single-threaded processors.

The simulation speed directly impacts how quickly we can evaluate architectural trade-offs. RASE has significant up-front costs for producing high quality instruction stream trace files in a non-intrusive manner, but yields rapid and consistent results afterwards. The SimCMT simulation engine can output simulation results at user-specified intervals. This data can be analyzed, while SimCMT is running, in order to monitor the simulation progress. For commercial server steady-state benchmark, we find that SimCMT can warm the caches and related buffers and queues in as little as 10 million cycles for Niagara-like processors. This minimum number of simulation cycles depends on the complexity of the simulated system and does vary. Furthermore, we can observe architectural differences, like cache size, at the same timing granularity. In order to avoid any spurious perturbations that could lead to incorrect conclusions, simulation periods of at least 50 million cycles are required to ascertain architectural performance differences. For these steady state benchmarks, we can use small samples of the large trace file to produce performance results. Subsampling these large trace files makes using techniques like those used by Wunderlich et al. [27] unnecessary for large steady-state benchmarks once the trace file is generated. Those or other sampling techniques could be used for trace generation, but the smaller trace file could limit the number of configurations that could be simulated in the future due to the small instruction stream.

The XML Test trace files present both ends of the spectrum with regard to trace replication and

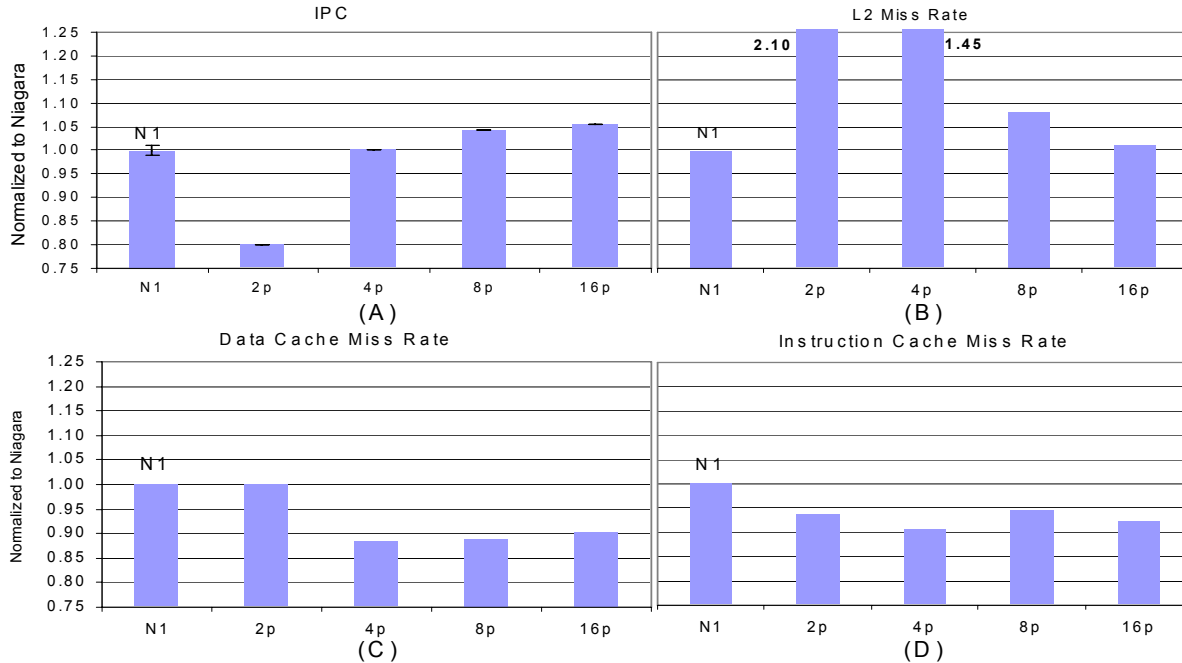


Figure 6: Niagara SPEC JBB2000 average IPC (A), L2 cache miss rate (B), data cache miss rate (C), and instruction cache miss rate (D) compared to RASE prediction using various initial instruction streams ranging in size from 2,4,8, to 16 processors.

performance prediction. At one extreme, we are replicating a single trace 31 times to produce a “representative” benchmark instruction stream. The single processor instruction trace has a very similar instruction mix compared to the 16 processor trace, but predicts a dramatic 40% performance degradation compared to XML Test running on Niagara. Niagara has 32 active threads that can produce very convoluted memory access patterns that cannot be understood without the assistance of a highly detailed simulation environment like RASE. Canonical single threaded processors have an explicit relationship with outstanding memory requests, because the processor stalls. However, Niagara was designed to tolerate memory latency by overlapping the execution of multiple threads. The 16p trace file is a better predictor of Niagara’s performance, but we can achieve very similar results from an overall performance perspective by enabling RASE to do intelligent modifications to the simulation environment to compensate for constructive and destructive interference from low thread-count trace files by using L2 set index hashing.

SPEC JBB2000 reinforces the lack of correlation between the accuracy of predicted cache miss rates and actual performance, as shown in Figure 6. We generated trace files for 2, 4, 8, and 16 processors for SPEC JBB2000 and compared RASE’s results to Niagara’s results. As expected, the 2p simulation had the largest absolute IPC difference. Overall, excluding the 2p simulation, all of the predicted performance results are within 5% of the average Niagara IPC. Ironically, the worst predictive simulation results in Figure 6 (A) matched the data cache miss rates the best in Figure 6 (C). This further illustrates how one might

be misled by miss rate calculations for multithreaded architectures.

Studying the shared L2 miss rate does provide some performance insight with regard to Figure 6’s performance comparison. The normalized L2 miss rate for all the simulation configurations compared to Niagara (N1) correlate well, with the 2p RASE results being the worst, and thereby explaining the IPC penalty. However, the 4p IPC results exactly match the hardware platform, but the 4p L2 miss rate is also off the chart. In the latter case, the miss rates in the L1 counterbalance the higher L2 miss rates.

Finally, as mentioned in Section 4.1, the TPC-C instruction mix is not uniform across all threads of instruction streams, unlike SPEC JBB2000 or XML Test. Therefore, we expect the small trace files to give less accurate performance prediction because they would be more skewed by the unbalanced kernel code distribution. TPC-C shows that thread replication must be adapted to each benchmark. When comparing the normalized IPC, data, instruction, and L2 cache miss rates with respect to the TPC-C on Niagara, we found that RASE is within 10% of Niagara for IPC and instruction and data cache misses and 5% for L2 miss rate using a 16p (thread) trace file.

The 4p TPC-C trace files also illustrate the poor predictability of small or scaled down simulation for large systems. Although we have developed methodologies to compensate for some of these issues, it injects another error source that potentially can lead you to incorrect conclusions. As expected, the simulation results using a higher thread count trace file more accurately predict performance.

Finally, multithreading complicates the performance estimates because of the complex memory access

patterns from interleaving multiple threads. Khalid advocates using both static and dynamic characteristics to correctly simulate applications [14]. RASE does this by incorporating information about the benchmarks, address space, and various detailed simulation modules. However, we believe that one static benchmark characteristic, the load/store instruction mix, must be well understood in order to produce accurate results. The Niagara architecture interleaves many memory operations, thus the load/store instruction mix must be representative of the benchmark or the resulting prediction will be very inaccurate. For the benchmarks presented in our study, we have observed the instruction mix to be well defined as we scale all the benchmarks, regardless of the system size.

6. Related Work

There is a large body of research on architecture simulation. Most recent simulation work for commercial servers deals with scaling the applications down [2][3][4][13]. Gibson et al. [12] points out the importance of modeling details of the design and validating against the hardware and specification. RASE addresses most of the shortcomings that [12] identifies as sources of simulation accuracy or error. Alameldeen et al. [4] presents a framework for simulating commercial workloads. To make the simulation feasible, they scaled down the workloads. It is not clear how they scale down the target system since stressing a more capable system requires a larger workload. In addition, there is no data showing how the simulation results compare against real hardware results. In contrast, we use large-scale applications and capture the application behavior in a trace file in order to scale up the applications for the target simulator. Basu et al. [8] takes a similar approach to ours, but they are not simulating large-scale commercial workloads due to the limitations of their tracing mechanism (from a live system). We use the execution-driven framework in RASE which does not perturb the system under observation [16]. Furthermore, Basu et al. [8] did not address how to scale up the trace to simulate larger configurations either. Khalid [14] uses a sampling approach to obtain the right instruction mix for target application for simulation. This work is focused on SPECCPU benchmarks instead of commercial workloads. However, Barroso et al. [6] used similar memory access pattern information to analyze commercial workload performance. We extend that work by scaling large-scale applications and doing full-system simulation.

There are many sampling techniques that have been proposed to improve simulation accuracy and speed [21][27]. We believe that these techniques are complimentary to the RASE framework. They could be incorporated to further reduce the simulation time required to produce accurate performance predictions. Binary translation simulation methods can yield even higher performance [8], but this methodology degenerates to execution-driven simulation if the simulated architecture varies greatly from host architecture, forcing excessive calls into the simulation library functions.

This paper presents a rapid and accurate approach to simulating highly threaded CMT processor architecture with a thoroughly validated, cycle-accurate SimCMT simulator and traces from highly tuned and validated commercial workloads. We also propose the thread replication strategy that utilizes remapped memory access patterns to simulate higher thread count architectures with minimal costs. Our approach is verified by comparing the simulation results for thread replication against a Niagara CMT system.

7. Conclusions and Future Work

We have presented a framework for large-scale full system simulation. By using a highly detailed system model, various levels of simulator validation, and large, representative trace files from highly tuned benchmarks, we are able to rapidly and very accurately predict multithreaded chip multiprocessor performance for a 32-way system using trace-driven simulation for commercial server applications. In order to simulate a large CMT architecture that implements more threads than the number of instruction streams available in the trace files, we scale up the trace file by replicating the instruction streams based on workload-specific memory access information. Our results have shown that when the ratio between the thread count in the target architecture and trace file is small (2:1), the simulation is able to predict within 5% of the IPC of the real hardware, and within 10% of the cache miss rates of the hardware. When the ratio becomes larger (>4), the method introduces more error. In an extreme case with a replication ratio of 32:1, we have observed a 20% IPC difference and 110% L2 miss rate difference. Therefore, it is still necessary to collect a trace that has a thread count within a factor of 4 of the target architecture, unless L2 index hashing or other additional techniques are incorporated into the simulation process. On the other hand, we have found that benchmarks that are not well tuned misrepresent the actual workload behavior resulting in poor performance prediction. One of the main reasons that this replication approach works well is that in our experience and in published results, we find that the large scale, highly tuned benchmarks are scaling rather well. As the system-under-test size increases, the performance on 64-bit systems scales almost linearly, in particular on Itanium and Power 5, for TPC-C and SPECJBB2000 [24][26]. Here, too TPC-C is an example of a benchmark that requires more specific benchmark application behavior knowledge to extend the naïve replication process to scale the benchmark properly. Using our methodology, the TPC-C results on hardware mirror those predicted by RASE.

There are many future directions for the RASE framework. Using techniques similar to [21][27], we would like to formalize a method to accurately predict performance for applications with phase behavior and thereby reduce simulation time for all application simulations. This applies to both commercial server workloads and other application domains. This would dramatically increase the scope of workloads that we could accurately simulate. Improving the thread replication process by integrating OS page remapping

and related process-creation memory management would address some of the artificial cache conflicts introduced by using low-thread count instruction trace files. We can achieve this by dynamically altering and inserting instruction streams into the existing trace files.

Acknowledgements

We would like to thank the rest of the Niagara performance team: Jeff Gibson, Venkatesh Iyengar, Hong-Men Su, and Jenn-Yuan Tsai for developing the simulator and their thoughtful comments on the work presented in this paper.

References

- [1] K. Aingaran, P. Kongetira, et al., "A 32-way Multithreaded SPARC® Processor," 16th Hot Chips Symposium, Aug. 2004.
- [2] A.R. Alameldeen, C. J. Mauer, et al., "Evaluating Non-deterministic Multi-threaded Commercial Workloads," Computer Architecture Evaluation using Commercial Workloads (CAECW), February 2002.
- [3] A.R. Alameldeen and D.A. Wood, "Variability in Architectural Simulations of Multi-threaded Workloads," 9th Int'l Symp. on High Performance Computer Architecture (HPCA), Feb. 2003.
- [4] R. Alameldeen; M.M.K. Martin, et al., "Simulating a \$2M commercial server on a \$2K PC," Computer , Volume: 36 , Issue: 2 , Feb. 2003 Pp:50 - 57
- [5] M. Annavaram, et al. "The Fuzzy Correlation between Code and Performance Predictability," MICRO-37, Dec. 2004
- [6] L. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," Proc. of the 25th Annual Int'l Symp. on Computer Architecture (ISCA), June 1998, pp: 3-14.
- [7] L. Barroso, K. Gharachorloo, R. McNamara, et al., "Piranha: a scalable architecture based on single-chip multiprocessing," ISCA-27, June 2000, pp: 282 -293.
- [8] S. Basu, S. Roy, R. Kumar, T. Fisher, B.E. Blaho, "Peppermint and Sled: tools for evaluating SMP systems based on IA-64 (IPF) processors," International Proceedings on Parallel and Distributed Processing Symposium, IPDPS 2002, 15-19 April 2002, pp: 54 - 63
- [9] R. Bedichek, "SimNow™: Fast Platform Simulation Purely in Software," 16th Hot Chips Symp., August 2004.
- [10] J. D. Davis, J. Laudon, and K. Olukotun, "Maximizing CMP Throughput with Mediocre Cores," Int'l Conference on Parallel Architectures and Compilation Techniques (PACT), Sept. 2005, pp. 51-62.
- [11] F. Eskesen, et al., "Performance Analysis of Simultaneous Multithreading in a PowerPC-based Processor," IBM Research Report, May 2002, RC22454.
- [12] J. Gibson, R. Kunz, et al. "FLASH vs. (Simulated) FLASH: Closing the Simulation Loop", In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp: 49-58, November 2000.
- [13] J. Huh, S.W. Keckler and D. Burger, "Exploring the Design Space of Future CMPs," PACT, Sept. 2001 pp. 199-210.
- [14] H. Khalid, "Validating trace-driven microarchitectural simulations," Micro, IEEE, Vol: 20, Issue: 6, Nov.-Dec. 2000, Page(s):76-82
- [15] S. Kunkel, B. Armstrong, P. Vitale, "System optimization for OLTP workloads," Micro, IEEE , Volume: 19 Issue: 3, May-June 1999 Page(s): 56 -64.
- [16] S. Kunkel, et al., "A performance methodology for commercial servers," IBM Journal of Research and Development, Vol. 44, Number 6, 2000.
- [17] J. Laudon, A. Gupta, and M. Horowitz, "Interleaving: A Multithreading Technique Targeting Multiprocessors and Workstations," Proc. of the 6th Int'l Symp. on Architectural Support for Parallel Languages and Operating Systems (ASPLOS), Oct. 1994, pp: 308-318.
- [18] J. Lo, L. Barroso, S. Eggers, K. Gharachorloo, et al., "An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors," ISCA-25, Jun 1998, pp: 39-50.
- [19] P. Magnusson, M. Christensson, et al., "Simics: A Full System Simulation Platform," Computer, February 2002, pp: 50-58.
- [20] P. Ranganathan, K. Gharachorloo et al., "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors," ASLPOS-8, Oct.1998, pp: 307-318.
- [21] T. Sherwood, S. Sair, and B. Calder, "Phase Tracking and Prediction," ISCA-30, June 2003.
- [22] L. Spracklen and S. Abraham, "Chip Multithreading: Opportunities and Challenges," HPCA-11, Feb. 2005
- [23] R. Stets, L.A. Barroso, et al., "A Detailed Comparison of TPC-C versus TPC-B," Third Workshop on CAECW, January 2000.
- [24] Standard Performance Evaluation Corporation, SPEC*, <http://www.spec.org>, Warrenton, VA
- [25] Sun Microsystems Inc., "XML Processing Performance in Java and .Net," http://java.sun.com/performance/reference/whitepapers/XML_Test-1_0.pdf
- [26] Transaction Processing Performance Council, TPC-*, <http://www.tpc.org>, San Francisco, CA
- [27] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, J. C. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," ISCA-30, June 2003
- [28] Qin Xiaohan; J. L. Baer, "A comparative study of conservative and optimistic trace-driven simulations," Simulation Symposium, 1995. Proceedings of the 28th Annual, 9-13 April 1995 Page(s): 42 - 50