Statistical Simulation of Chip Multiprocessors Running Multi-Program Workloads

Davy Genbrugge Lieven Eeckhout

ELIS Department, Ghent University, Belgium
Email: {dgenbrug,leeckhou}@elis.UGent.be

Abstract

This paper explores statistical simulation as a fast simulation technique for driving chip multiprocessor (CMP) design space exploration. The idea of statistical simulation is to measure a number of important program execution characteristics, generate a synthetic trace, and simulate that synthetic trace. The important benefit is that a synthetic trace is very small compared to real program traces.

This paper advances statistical simulation by modeling shared resources, such as shared caches and off-chip bandwidth. This is done (i) by collecting cache set access probabilities and per-set LRU stack depth profiles, and (ii) by modeling a program's time-varying execution behavior in the synthetic trace. The key benefit is that the statistical profile is independent of a given cache configuration and the amount of multiprocessing, which enables statistical simulation to model conflict behavior in shared caches when multiple programs are co-executing on a CMP. We demonstrate that statistical simulation is both accurate and fast with average IPC prediction errors of less than 5.5% and simulation speedups of 40X to 70X compared to the detailed simulation of 100M-instruction traces. This makes statistical simulation a viable tool for CMP design space exploration.

1 Introduction

Architectural simulation is a crucial tool in a computer designer's toolbox because of its flexibility, its ease of use and its ability to drive design decisions early in the design cycle. The downside however is that architectural simulations are very time-consuming. Simulating an industry-standard benchmark to completion for a uniprocessor design point easily takes a couple of weeks, even on today's fastest machines and simulators. Culling a large design space through architectural simulation of complete benchmark executions thus simply is infeasible. And this problem keeps on increasing over time. Given the current era of CMP design, there is a big quest for efficient simulation techniques for driving the design process of CMPs with multiple tens of cores integrated on a single chip.

Researchers and computer designers are well aware of the multi-core simulation problem and have been proposing various simulation methodologies for coping with it, such as sampled simulation [1, 7, 15, 17], or parallelized simulation and/or hardware accelerated simulation using FPGAs [12]. In this paper we take a different approach through statistical simulation. The idea of statistical simulation is to first measure a statistical profile of a program execution through (specialized) functional simulation. These statistics are then used to build a synthetic trace; this synthetic trace exhibits the same characteristics as the original program trace, by construction, but is much shorter than the original program trace, no more than a few millions of instructions. Simulating this synthetic trace then yields a performance estimate. Given its short length, simulating a synthetic trace is done very quickly.

Previous work has been exploring the statistical simulation paradigm extensively for uniprocessors [5, 8, 9, 11], and a single study [10] applied statistical simulation to multithreaded workloads running on shared memory multiprocessor systems. None of this prior work addresses the modeling of shared resources in chip multiprocessors though. This paper advances the statistical simulation methodology by proposing models for shared resources such as shared caches and off-chip bandwidth. This makes statistical simulation a viable fast simulation technique for quickly exploring chip multiprocessor design spaces. Note that we do not envision statistical simulation as a substitute to detailed simulation. We rather consider statistical simulation as a useful complement to detailed simulation at the earliest stages of the design: the design space can be culled using statistical simulation, and when a region of interest is identified, detailed but slower simulation can then be used to explore the region of interest in greater detail.

This paper makes the following contributions:

1-4244-1258-7/07/$25.00 ©2007 IEEE

- We extend the statistical simulation methodology to chip multiprocessors running multi-program workloads. To enable the accurate modeling of shared caches in chip multiprocessors, we collect statistics to capture the cache access behavior, such as set access probabilities and per-set LRU stack distance probabilities. Co-simulating synthetic traces annotated with set and LRU stack distance information then gives an accurate picture of the conflict behavior in shared caches.

- We show that in order to accurately model conflict behavior in shared resources, it is important to accurately model the time-varying program execution behavior. To this end, we collect a statistical profile and generate a synthetic mini-trace per instruction interval, and then subsequently coalesce these mini-traces to an overall synthetic trace.

- The cache set access and per-set LRU stack distance statistics make the statistical profile less microarchitecture-dependent. A single statistical profile for the largest cache of interest during the design space exploration can now be used to explore various cache configurations for a given cache line size whereas previous work requires a statistical profile for each cache configuration of interest.

- We demonstrate that the overall framework presented in this paper is accurate and efficient for quickly exploring the chip multiprocessor design space.

2 Statistical uniprocessor simulation (in a nutshell)

Statistical simulation is done in three steps. We first measure a *statistical profile* which is a collection of important program behavior characteristics. The statistical profile comprises program characteristics such as the instruction mix, the inter-instruction dependency (through registers and memory) distribution, the statistical control flow graph (transition probabilities between basic blocks), per-branch misprediction rates, per-load/store cache miss rates, etc.

Subsequently, this statistical profile is used to generate a *synthetic trace*. The synthetic trace is a linear sequence of synthetic instructions. Each instruction has an instruction type, a number of source operands, an inter-instruction dependency for each source operand (which denotes the producer for the given source operand), I-cache miss info, D-cache miss info (in case of a load), and branch miss info (in case of a branch). The locality miss events are just labels in the synthetic trace describing whether the load is an L1 D-cache hit, L2 hit or L2 miss, and whether the load generates a TLB miss. Similar labels are assigned for the I-cache and branch miss events. In the final step, this synthetic trace is simulated on a statistical simulator yielding performance metrics such as IPC.

The important benefit of statistical simulation is that a synthetic trace is very short. As such, synthetic traces containing no more than a few million of instructions are sufficient for obtaining converged performance estimates. For a more elaborate discussion on statistical simulation for uniprocessors, we refer to [5, 8, 9, 11].

3 Statistical CMP simulation

Statistical simulation as described in the previous section cannot be applied in a straightforward manner to model chip multiprocessors with shared resources. The reason is that one of the characteristics in a statistical profile is the cache miss rate for a given cache hierarchy. However, co-executing multiple programs on a chip multiprocessor with shared L2 and/or L3 caches will affect the cache miss rate, the amount of off-chip bandwidth requests, and thus overall performance. And the level of interaction between programs is greatly affected by the programs co-executing: the interaction may be minimal for some co-executing programs; for others, the amount of interaction can be substantial. Moreover, the level of interaction can be affected by the CMP cores' microarchitecture — the amount of interaction can be very different for one microarchitecture compared to another.

We now discuss how to extend statistical simulation for simulating multi-program workloads running on a CMP. This is done in three steps: (i) cache set and LRU stack profiling, (ii) statistical simulation of shared caches, and (iii) modeling time-varying program execution behavior.

3.1 Cache set and LRU stack profiling

In order to enable the simulation of shared L2 caches in a CMP, we collect two novel program characteristics in the statistical profile, namely the *cache set profile* and the *per-set LRU stack depth profile*. (Throughout the paper we refer to the shared cache as the L2 cache; extending our framework to model shared L3 caches is trivial.) For doing this, we assume a large L2 cache, i.e., the largest L2 cache one may be potentially interested in during design space exploration. The cache set and LRU stack depth profiles for smaller L2 caches can then be derived from the profile measured for the largest L2 cache, as will be discussed later. In our experimental setup, the largest L2 cache of interest is a 16-way set-associative 16MB L2 cache.

For every memory reference — this includes instruction addresses as well as data addresses — we determine the L2 cache set that is to be accessed in the largest L2 cache of interest. In addition, we also determine the LRU stack depth

in the given set. The maximum LRU stack depth kept track of during profiling is $(a+1)$ with a being the associativity of the largest L2 cache of interest.

The cache set and per-set LRU stack depth profile can be used to estimate cache miss rates for caches that are smaller than the largest L2 cache of interest. All accesses to an LRU stack depth larger than a will be cache misses in an a-way set-associative cache. Similarly, all accesses to sets s and $s+S/2$ for a cache with S sets will result in accesses to set s for a cache with $S/2$ sets.

The L2 cache set and LRU stack depth profiles are dependent on a given cache line size. In practice, there is only a few L2 cache line sizes of interest, which limits the number of L2 cache set and stack profiles that need to be computed.

3.2 Statistical simulation of a shared L2 cache

When generating a synthetic trace we probabilistically generate a cache set and LRU stack depth accessed for each memory reference based on the measured profiles. Simulating the synthetic trace on a CMP with a shared L2 cache then requires that we effectively simulate the L1 D-cache and the L2 cache. In statistical simulation for a uniprocessor system on the other hand, caches do not need to be simulated since cache misses are simply flagged as such in the synthetic trace; based on these cache miss flags, appropriate latencies are assigned. Statistical simulation of a CMP with a shared cache, on the other hand, requires that the caches are simulated in order to model shared cache conflict behavior.

Each cache line in the shared L2 cache contains the following information:

- The ID of the program that most recently accessed the cache line; we will refer to this ID as the *program ID*. This enables the statistical simulator to keep track of the program 'owning' the cache line.
- The set index of the set in the largest L2 cache of interest that corresponds to the given cache line; we will refer to this set index as the *stored set index*. In case the L2 cache being simulated has as many sets as the largest L2 cache of interest, the stored set index is the set index of the simulated cache itself. The stored set index will enable the statistical simulator to model cache lines conflicting for a given set in case the number of sets is reduced for the simulated cache compared to the largest cache of interest.
- A valid bit stating whether the cache line is valid.
- A cold bit stating whether the cache line has been accessed. The cold bit will be used for driving the cache warmup as will be discussed later.
- In case of a write-back cache, we also maintain a dirty bit stating whether the cache line has been written by a store operation.
- And finally, we also keep track of which instruction in the synthetic trace accessed the given cache line; this is done by storing the position of the instruction in the synthetic trace which we call the *instruction ID*.

Simulating the shared L2 cache then proceeds as follows assuming that all memory references are annotated with set information s and LRU stack depth information d for the largest cache of interest. We first determine the set s' being accessed in the simulated cache. The cache access is considered a cache hit in case there are at least d valid cache lines in set s' for which (i) the stored program IDs equal the ID of the program being simulated, and (ii) the stored set indices equal s. In case the above conditions do not hold, the cache access is seen as a cache miss. The most recently accessed cache block is put on top of the LRU stack for the given set.

An appropriate warmup approach is required for the L2 cache; without appropriate warmup, the L2 cache would suffer from a large number of cold misses. Making the synthetic trace longer could solve this problem, however, this would definitely affect the usefulness of statistical simulation which is to provide performance estimates from very fast simulation runs. As such, we take a different approach and warmup the L2 cache. The warmup technique that we use first initializes all cache lines as being cold by setting the cold bit in all cache lines. The warmup approach then applies a *hit-on-cold* strategy, i.e., upon the first access to a given cache line we assume it is a hit and the cold bit is set to zero. Thit hit-on-cold warmup strategy is simple to implement, and is fairly accurate.

During this work, we also found that it is important to model L1 D-cache write-backs during synthetic trace simulation because write-backs can have a significant impact on the conflict behavior in the shared L2 cache. This is done by simulating the L1 D-cache similar to what is described above for the L2 cache. We assume that an L1 D-cache write-back is an L2 cache miss in case all instruction IDs in the given L2 set are larger than the instruction ID of the cache line written back from L1 into L2; if not, it is assumed a hit.

3.3 Modeling time-varying behavior

A critical issue to the accuracy of statistical simulation for modeling CMP performance is that the synthetic trace has to capture the original program's time-varying phase behavior. The reason is that overall performance is affected by the phase behavior of the co-executing programs: the relative progress of a program is affected by the conflict

behavior in the shared resources [17]. For example, extra cache misses induced by cache sharing may slow down a program's execution. This one program running relatively slower may result in different program phases co-executing with the other program(s), which, in turn, may result in different sharing behavior, and thus faster or slower relative progress.

To model the time-varying behavior we divide the entire program trace into a number of instruction intervals; an instruction interval is a sequence of consecutive instructions in the dynamic instruction stream. We then collect a statistical profile per instruction interval and generate a synthetic mini-trace. Coalescing these mini-traces yields the overall synthetic trace. The synthetic trace then captures the original trace's time-varying behavior.

4 Experimental setup

We use the SPEC CPU2000 benchmarks with the reference inputs in our experimental setup. The Alpha binaries of the CPU2000 benchmarks were taken from the SimpleScalar website. We considered 100M single (and early) simulation points as determined by SimPoint [13, 14] in all of our experiments. The synthetic traces are 4M instructions long, unless mentioned otherwise — we provide a motivation for this in Section 5.3. For measuring the statistical profiles capturing time-varying behavior, we measure a statistical profile per 10M-instruction interval. From these ten statistical profiles, we then generate 10 400K-instruction mini-traces that are subsequently coalesced to form the 4M-instruction synthetic traces.

We use the M5 simulator [2] in all of our experiments. Our baseline core microarchitecture is a 4-wide superscalar out-of-order core with 64KB private L1 I- and D-caches with a 2-cycle access latency. When simulating a CMP, we assume that all cores share the L2 cache as well as the off-chip bandwidth for accessing main memory; the baseline L2 cache is an 8-way set-associative 4MB cache with a 12-cycle access latency; main memory has an access latency of 150 cycles. We model write buffers and MSHRs in our cache hierarchy. Simulation stops as soon as one of the co-executing programs terminates, i.e., as soon as one of the programs has executed 100M instructions in case of detailed simulation, or 4M instructions in case of statistical simulation.

5 Evaluation

We now evaluate the statistical simulation methodology proposed in this paper along three dimensions: (i) accuracy both in terms of a single design point as in terms of exploring a design space, (ii) simulation speed, and (iii) storage requirements for storing the statistical profiles on disk.

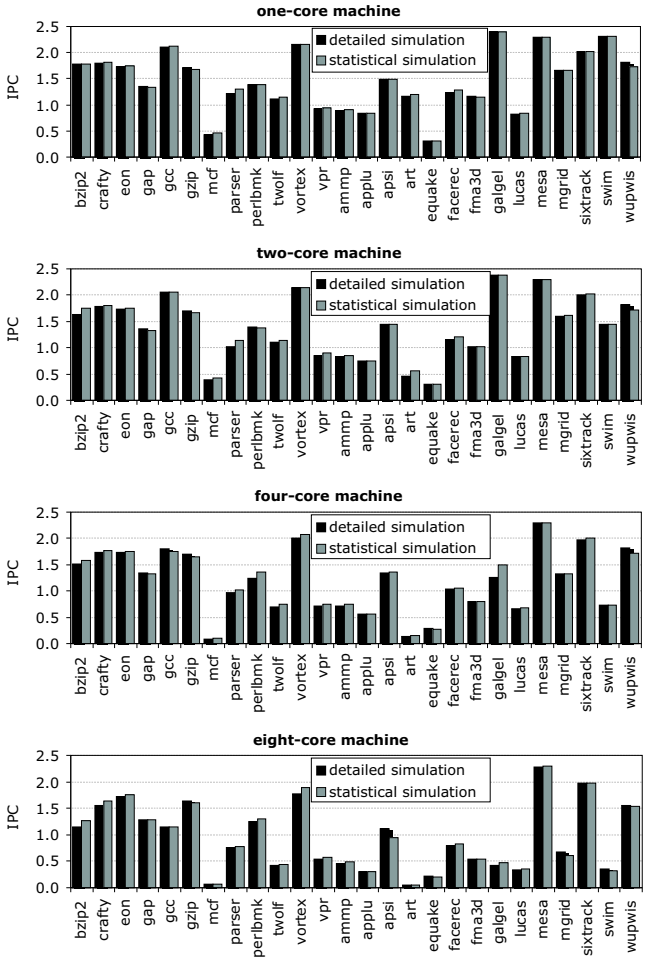

Figure 1. Evaluating the accuracy of statistical simulation for single-program and homogeneous multi-program workloads.

5.1 Accuracy

Homogeneous workloads. The top graph in Figure 1 evaluates the accuracy of statistical simulation for a single program running on a single-core processor. The average IPC prediction error is 1.6%; this is in line with our prior work [8]. The other three graphs in Figure 1 evaluate the accuracy when running homogeneous multi-program workloads on a multi-core with a shared L2 cache, i.e., multiple copies of the same program are executed simultaneously. The average prediction error for the two-core, four-core and eight-core machines are 3.1%, 4.5% and 4.6%, respectively. Statistical simulation is capable of accurately tracking the impact of the shared L2 cache and off-chip bandwidth on overall application performance: for some programs, resource sharing has almost no impact, see for example mesa; for other programs on the other hand, sharing has a large impact, see for example art, mgrid and swim.

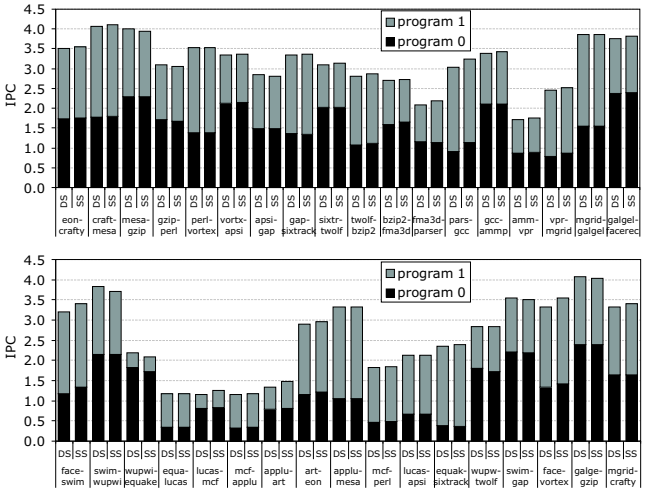

Figure 2. Evaluating the accuracy of statistical simulation for heterogeneous two-program workloads: detailed simulation (DS) vs. statistical simulation (SS).

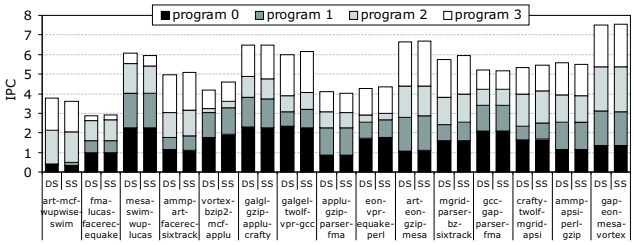

Figure 3. Evaluating the accuracy of statistical simulation for heterogeneous four-program workloads: detailed simulation (DS) vs. statistical simulation (SS).

Heterogeneous workloads. Figures 2 and 3 evaluate the accuracy of statistical simulation for randomly chosen heterogeneous two-program and four-program workloads, respectively. These figures show that statistical simulation not only accurately predicts overall system IPC throughput for heterogeneous workloads, it also accurately predicts per-program IPC values. For example, for the two-program workloads, the average throughput prediction error is 2.4% and the average per-program IPC prediction error is 3.4%.

Modeling time-varying behavior. As mentioned in Section 3.3, it is important to model a program's time-varying behavior in the synthetic trace when studying shared resources in CMPs through statistical simulation. This is experimentally evaluated in Figure 4 which shows the average IPC prediction error for heterogeneous two-program workloads (i) without time-varying behavior modeling versus (ii) with time-varying behavior modeling i.e., by generating and coalescing mini-traces. Modeling time-varying behavior reduces the IPC prediction error from 8.9% to 3.4% on average.

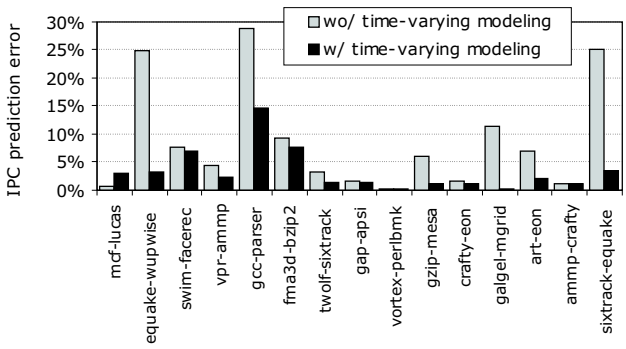

Figure 4. Evaluating the importance of modeling time-varying behavior.

5.2 Design space exploration

We now demonstrate the accuracy of statistical simulation for driving design space exploration, which is the ultimate goal of the statistical simulation methodology. To do so, we consider a design space with varying L2 cache configurations and a varying number of cores. We vary the L2 cache size from 1MB to 16MB with varying associativity from 2- to 16-way set-associative; the cache line size is kept constant at 64 bytes. And we vary the number of cores from 1, 2, 4 up to 8. This design space consisting of 56 design points is small compared to a realistic design space, however, the reason is that we are validating the accuracy of statistical simulation against detailed simulation for each of the design points. The detailed simulation for all those 56 design points was very much time-consuming, which is the motivation for statistical simulation in the first place.

Figure 5 shows a scatter plot with system IPC throughput through detailed simulation on the vertical axis versus system IPC throughput through statistical simulation on the horizontal axis. The two graphs in Figure 5 show two different heterogeneous eight-program mixes; one mix of L2-intensive benchmarks (on the left), and one mix of L2-intensive and non L2-intensive benchmarks (on the right). The IPC throughput estimates through statistical simulation show a close to perfect correlation with the IPC throughput numbers obtained from detailed simulation. The average IPC prediction error over the entire design space is 5.5%. We observed very similar trends for other two-, four- and eight-program mixes.

Cache design space exploration. Figure 6 illustrates the accuracy of statistical simulation for estimating the global shared L2 cache miss rate, i.e., the number of L2 misses divided by the number of L1 accesses. We consider a shared

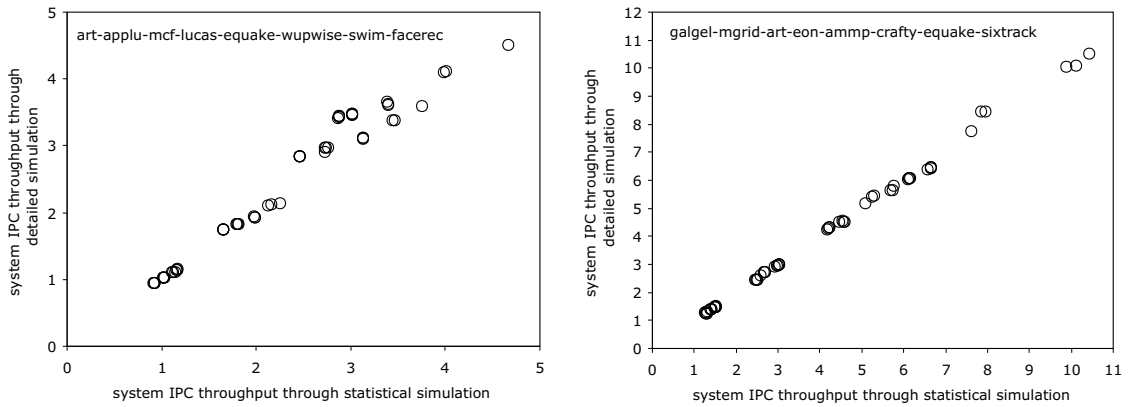

Figure 5. Evaluating the accuracy of statistical simulation for exploring CMP design spaces: measured system IPC throughput through detailed simulation versus estimated system IPC throughput through statistical simulation. The two graphs represent two eight-program workload mixes.

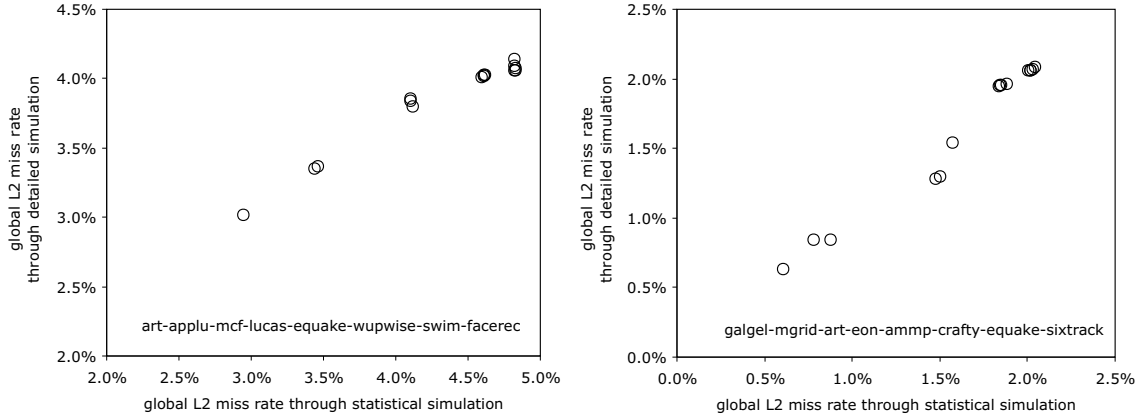

Figure 6. Evaluating the accuracy of statistical simulation for exploring the shared L2 cache design space: global L2 miss rate is shown for detailed simulation versus statistical simulation. The two graphs represent two eight-program workload mixes.

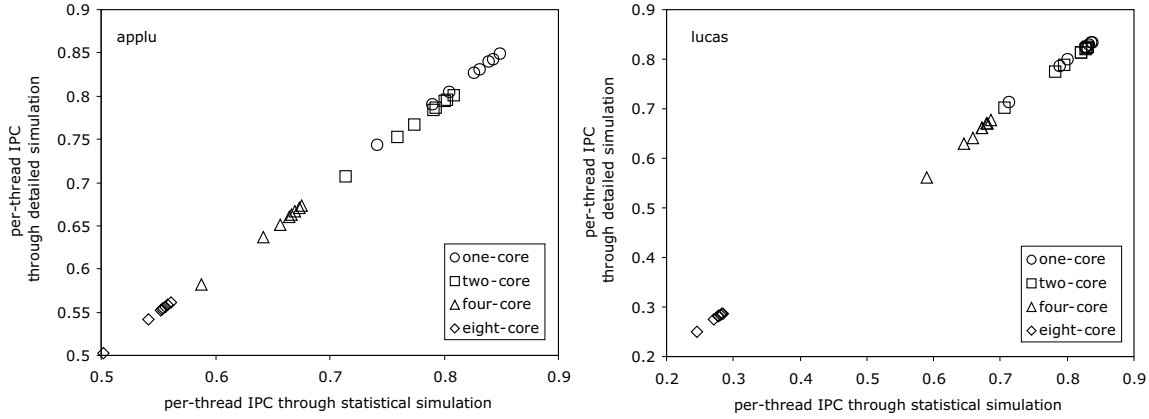

Figure 7. Evaluating the accuracy of statistical simulation for studying off-chip bandwidth: measured system IPC throughput through detailed simulation versus estimated system IPC throughput through statistical simulation. The two graphs represent applu **(left graph) and** lucas **(right graph).**

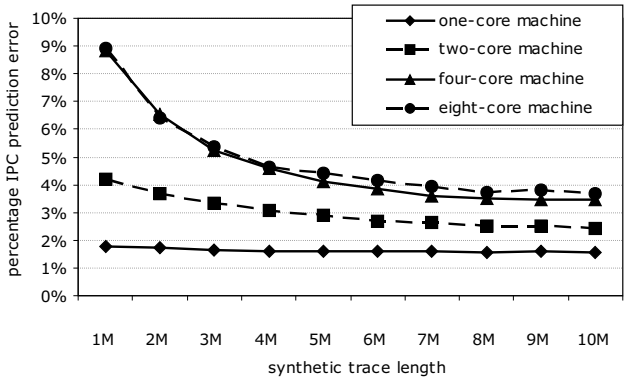

Figure 8. Percentage average IPC prediction error as a function of synthetic trace length.

L2 cache design space consisting of 15 design points by varying the L2 cache size from 1MB up to 16MB with varying associativity and number of sets. We show two eight-program workloads here, but obtained similar results for other workload mixes. Again, the overall conclusion is that statistical simulation accurately tracks performance differences across cache configurations and across a different number of cores. Note that these results were obtained from a single statistical profile, namely a statistical profile for the largest cache of interest, a 16MB 16-way set-associative cache. In other words, a single statistical profile is sufficient to drive a cache design space exploration.

Off-chip bandwidth experiments. Figure 7 shows system throughput while varying the off-chip bandwidth for two workloads that are sensitive to off-chip bandwidth, namely applu and lucas. The off-chip bandwidth design space is explored by varying the width (8-byte vs 16-byte) and frequency (333MHz, 666MHz, 999MHz and 1.3GHz). Again, we observe that statistical simulation tracks detailed simulation very accurately.

5.3 Simulation speed

Having shown the accuracy of statistical simulation for CMP design space exploration, we now evaluate its simulation speed. Figure 8 shows the average IPC prediction error as a function of the synthetic trace length. For a single-program workload, the prediction error stays almost flat, i.e., increasing the size of the synthetic trace beyond 1M instructions does not increase prediction accuracy. For multi-program workloads on the other hand, the prediction accuracy is sensitive to the synthetic trace length. The reason is that the shared cache requires more warmup to establish the conflict behavior in shared resources between multiple co-executing programs. However, once the synthetic trace contains more than 4M instructions, the prediction error remains almost flat. This observation motivated us to report all of our numbers using 4M instruction synthetic traces.

Recall that in these experiments we went from 100M instruction real program traces to 4M instruction synthetic traces. This is a 25X decrease in the dynamic instruction count. In our statistical simulator, this results in a 40X to 70X speedup in simulation time. The reason why the simulation time reduction is larger than the dynamic instruction count reduction is that the statistical simulator does not have to model all the functionality a detailed simulator has to do; for example, the statistical simulator does not model the branch predictor, the I-cache, etc.

5.4 Storage requirements

As a final note, the storage requirements are modest for statistical simulation. The statistical profiles when compressed on disk are 24MB on average per benchmark.

6 Related work

Statistical simulation has grown in interest over the recent years and has evolved from fairly simple models [3, 6] to more and more detailed models [5, 8, 9, 11]. The most accurate model to date [8] reports an average IPC prediction error of 2.3% for a wide superscalar out-of-order processor compared to detailed simulation. Those models are limited to single-core processor modeling though.

Nussbaum and Smith [10] extended the uniprocessor statistical simulation method to enable the modeling of multithreaded programs running on shared-memory multiprocessor (SMP) systems. To do so, they extended statistical simulation to model synchronization and accesses to shared memory. Cache behavior is modeled based on cache miss rates though; in other words, they do not model shared caches.

Chandra et al. [4] propose performance models to predict the impact of cache sharing on co-scheduled programs. The output provided by the performance model is an estimate of the number of extra cache misses for each thread due to cache sharing. These performance models are limited to predicting cache sharing effects, and do not predict overall performance.

Sampled simulation is a popular approach to speed up simulation. Van Biesbrouck et al. [15, 16, 17] propose the co-phase matrix for guiding sampled simultaneous multithreading (SMT) processor simulation running multi-program workloads. Ekman and Stenström [7] and Wenisch et al. [18] use random sampling and systematic sampling, respectively, to speed up multiprocessor simulation. Barr et al. [1] propose the Memory Timestamp Record (MTR) to store microarchitecture state (cache and directory state) at

the beginning of each sample as a checkpoint. The main advantage of statistical simulation over sampled simulation is that it requires even fewer instructions to simulate; as demonstrated in this paper, we reduce the simulation time by more than a factor 40X to 70X compared to sampled simulation using 100M instruction samples.

7 Conclusion

Simulating chip multiprocessors is extremely time-consuming. This is especially a concern in the earliest stages of the design cycle where a large number of design points need to be explored quickly. This paper proposed statistical simulation as a fast simulation technique for chip multiprocessors running multi-program workloads. In order to do so, we extended the statistical simulation paradigm (i) to collect cache set access and per-set LRU stack depth profiles, and (ii) to model time-varying program behavior in the synthetic traces. These two enhancements enable the accurate modeling of the conflict behavior observed in shared resources such as shared caches and off-chip bandwidth. Our experimental results showed that statistical simulation is accurate with average IPC prediction errors of less than 5.5% over a broad range of CMP design points, while being 40X to 70X times faster than detailed simulation of 100M instruction traces. This makes statistical simulation a viable fast simulation approach to CMP design space exploration.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable feedback. We are also grateful to Jos Delbar who did some preliminary experiments for this paper. Lieven Eeckhout is a postdoctoral fellow with the Fund for Scientific Research–Flanders (Belgium) (FWO Vlaanderen). This work is also partially funded by Ghent University/BOF under contract No. 01J14407, the HiPEAC Network of Excellence and the European SARC project No. 27648.

References

[1] K. C. Barr, H. Pan, M. Zhang, and K. Asanovic. Accelerating multiprocessor simulation with a memory timestamp record. In *ISPASS*, pages 66–77, Mar. 2005.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

[3] R. Carl and J. E. Smith. Modeling superscalar processors via statistical simulation. In *PAID, held with ISCA*, June 1998.

[4] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting inter-thread cache contention on a chip-multiprocessor architecture. In *HPCA*, pages 340–351, Feb. 2005.

[5] L. Eeckhout, R. H. Bell Jr., B. Stougie, K. De Bosschere, and L. K. John. Control flow modeling in statistical simulation for accurate and efficient processor design studies. In *ISCA*, pages 350–361, June 2004.

[6] L. Eeckhout and K. De Bosschere. Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces. In *PACT*, pages 25–34, Sept. 2001.

[7] M. Ekman and P. Stenström. Enhancing multiprocessor architecture simulation speed using matched-pair comparison. In *ISPASS*, pages 89–99, Mar. 2005.

[8] D. Genbrugge and L. Eeckhout. Memory data flow modeling in statistical simulation for the efficient exploration of microprocessor design spaces. *IEEE Transactions on Computers*, 2007. Accepted for publication.

[9] S. Nussbaum and J. E. Smith. Modeling superscalar processors via statistical simulation. In *PACT*, pages 15–24, Sept. 2001.

[10] S. Nussbaum and J. E. Smith. Statistical simulation of symmetric multiprocessor systems. In *ANSS*, pages 89–97, Apr. 2002.

[11] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining statistical and symbolic simulation to guide microprocessor design. In *ISCA*, pages 71–82, June 2000.

[12] D. A. Penry, D. Fay, D. Hodgdon, R. Wells, G. Schelle, D. I. August, and D. Connors. Exploiting parallelism and structure to accelerate the simulation of chip multi-processors. In *HPCA*, pages 27–38, Feb. 2006.

[13] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *PACT*, pages 244–256, Sept. 2003.

[14] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57, Oct. 2002.

[15] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Considering all starting points for simultaneous multithreading simulation. In *ISPASS*, pages 143–153, Mar. 2006.

[16] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Representative multiprogram workloads for multithreaded processor simulation. In *IISWC*, Oct. 2007.

[17] M. Van Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *ISPASS*, pages 45–56, Mar. 2004.

[18] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, July 2006.