Enhanced Statistical Simulation Framework: Accurate Memory Data Flow Model

Davy Genbrugge*, Lieven Eeckhout*,
Koen De Bosschere*,

* *ELIS, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium*

ABSTRACT

Microprocessor design is very time-consuming, due to the huge design space that needs to be explored in order to identify the optimal design. Detailed simulations, used to explore these design spaces, are fairly slow. For every design point of interest, several hundreds of billions of instructions per benchmark need to be simulated. Recently, statistical simulation was proposed to efficiently cull a huge design space. The basic idea of statistical simulation is to collect a number of program characteristics and to generate a synthetic trace from it. Simulating this synthetic trace is extremely fast as it contains a million instructions only.

We improve the statistical simulation methodology by proposing accurate memory data flow models. We model load forwarding, delayed cache hits and correlation between cache misses based on path info. Our experiments using the SPECcpu2000 benchmarks show a substantial improvement upon current state-of-the-art statistical simulation methods.

1 Introduction

Standard trace-driven performance modeling techniques for superscalar processors are very accurate, but require extremely long simulation times, especially as traces reach lengths in the billions of instructions. This is a well recognized problem and several researchers have proposed solutions to this problem. In this paper we address to statistical simulation [1, 5]. The basic idea of statistical simulation is very simple. A number of program characteristics are measured from a real program execution in a so called statistical profile. A synthetic trace is then generated from this statistical profile which is then simulated on a simple trace-driven statistical simulator.

Previous work on statistical simulation however considers simple memory data flow modeling. The statistics considered in previously proposed statistical simulation approaches show three important shortcomings. First, statistical simulation typically assigns hits and misses to loads, not taking into account delayed hits. A delayed hit is a load hit referencing the same memory address of a preceding load miss; the load hit then sees a part of the la-

[1]E-mail: {dgenbrug,leeckhou,kdb}@elis.UGent.be

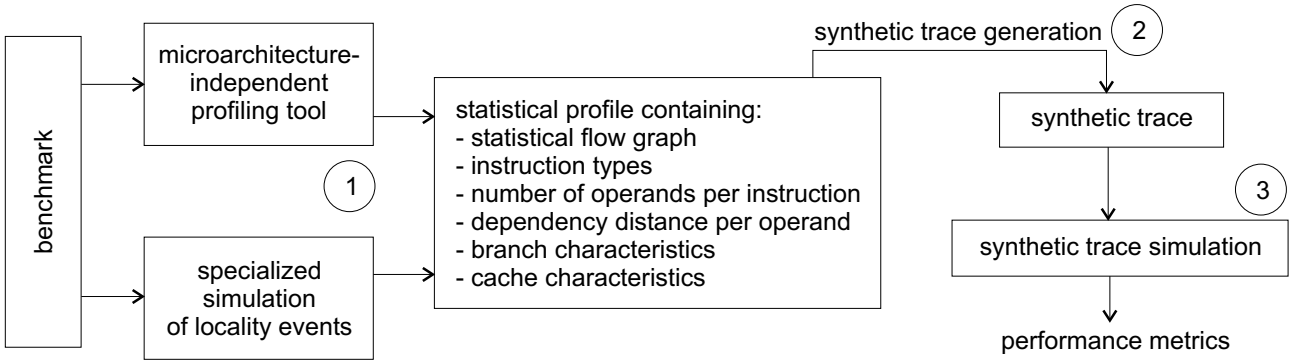

Figure 1: Statistical simulation: general framework.

tency of the preceding load miss, this happens in case the load miss did not return yet from the cache when the load hit issues. Second, none of the previously proposed statistical simulation approaches adequately model load bypassing and load forwarding, i.e., it is assumed that loads never alias with preceding stores. Third, none of this prior work models the correlation that may exist between cache misses. Cache miss correlation results in specific cache miss patterns that have an important impact on overall performance. In this paper, we address all three shortcomings by proposing how to extend statistical simulation for dealing with (i) delayed hits, (ii) load forwarding and (iii) cache miss correlation. We show that it is important to accurately model memory data flow and we provide techniques of how to achieve that.

2 Statistical simulation

Statistical simulation consists of three steps as shown in Figure 1.

We first measure a *statistical profile* which is a collection of important program execution characteristics. We make a distinction between microarchitecture-*in*dependent characteristics (*instruction mix*, *data-dependencies*) and microarchitecture-dependent characteristics (*branch statistics*, *cache statistics*). The key structure in the statistical profile is the *statistical flow graph (SFG)* [1] which represents the control flow in a statistical manner . In an SFG, the nodes are the basic blocks along with their basic block history. The edges in the SFG interconnecting the nodes represent transition probabilities between the nodes. The idea behind the SFG is to model all the other program characteristics along the nodes of the SFG. This allows for modeling program characteristics that are correlated with path behavior.

Subsequently, this statistical profile is used to generate a *synthetic trace* consisting of only a million of instructions. 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 describes 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).

In the final step, this synthetic trace is simulated on a statistical simulator which yields us performance metrics such as IPC. The important benefit of statistical simulation is that the synthetic traces are fairly short. The performance metrics such as IPC quickly converge to a steady-state value when simulating a synthetic trace. As such, synthetic traces containing a million of instructions are sufficient for obtaining accurate performance estimations.

3 Memory data flow modeling

As mentioned in the introduction, previously proposed statistical simulation approaches did not consider advanced memory data flow statistics. These approaches basically keep track of aggregate cache hit/miss information. In this section we detail the three additional features that we propose in this paper: (i) delayed hits, (ii) load forwarding and (iii) cache miss correlation.

Delayed hits The caches in today's microprocessors typically are non-blocking caches [2, 4]. Non-blocking caches allow for overlapping cache misses by putting aside load misses while servicing other load instructions. These in-overlap serviced load instructions can also be cache misses. Non-blocking caches have an important impact on overall performance. As such, it is important to model its impact on performance adequately.

In current statistical simulation frameworks however, only cache hits and cache misses are considered. In case of non-blocking caches, load instructions can see latencies that are different from the L1 access latency, L2 access latency and main memory access latency. Consider for example the case where a load accesses memory location A at time t_{100} and this is a cache miss to L2; the load thus finishes execution at time t_{120} in case the L2 access latency is 20 cycles. Assume now another load accessing the same memory location A at time t_{107}; this load will then see a load execution latency of 13 cycles. This is called a *delayed hit* or a *secondary miss*. Current statistical simulation frameworks will consider the delayed hit as a hit and will assign the L1 access latency to this load which is a serious underestimation of the load's execution latency.

In order to model delayed hits within statistical simulation, we compute the *read-after-read (RAR) memory dependency* distribution which measures the probability that a load operation reads the same memory location as one of the preceding load operations. This means that a probability is stored for a load i to read the same address as the preceding load $i-1$, or load $i-2$, or load $i-3$, etc. Note that this statistic is also measured within the context of an SFG.

Load forwarding Out-of-order execution of load instructions is another very important source for performance gain in out-of-order microprocessors. The goal is to execute load instructions as soon as possible (as soon as their source operands are ready) provided that read-after-write (RAW) dependencies through memory are respected. By doing so, load instructions possibly get executed before preceding store instructions.

Early out-of-order execution of loads is achieved in out-of-order microprocessors through two techniques, load bypassing and load forwarding [3]. Load bypassing refers to executing a load earlier than preceding stores; this is possible provided that the load address does not alias with those stores. In case the load aliases with a preceding store, i.e., there is a RAW dependency, load forwarding allows the load to retrieve its data directly from the store without accessing the memory hierarchy.

Modeling load bypassing and load forwarding can be done in statistical simulation by measuring the *RAW memory dependency* distribution. This distribution quantifies the probability that a load aliases with a preceding store $j-1$, $j-2$, $j-3$, etc. Again, this distribution is measured in the context of the SFG.

Cache miss correlation The third important memory data flow characteristic that we model is cache miss correlation. Cache miss correlation refers to the fact that the cache miss behavior of a particular load operation can be highly correlated with the cache miss behavior of (a) preceding load(s). Consider for example a loop that walks over an array. Each element in the array is 8 bytes long and a cache line is 32 bytes long. As a result, in case the array is not residing in the cache, a cache miss will occur every four iterations of the loop. This cannot be modeled accurately in the previously proposed statistical simulation frameworks. All iterations of this same loop will collapse in a single number, namely the cache miss rate (which is 75%) for that particular static load in the loop, i.e., no distinction is made between different iterations of the loop. Even the statistical flow graph is incapable of modeling this behavior accurately because the basic block history for that load is always the same sequence of basic blocks from the loops itself. This results in aggregate statistics that average over multiple executions of the same load which decreases the accuracy of statistical simulation. In other words, no distinction can be made between loop iterations with a load miss versus loop iterations with a load hit.

When modeling cache miss correlation we collect separate statistics for the cache miss behavior of a single static load instruction depending on its global cache miss history. The global cache miss history is a concatenation of the most recent cache hit/miss outcomes. In the above example where a loop walks over an array, cache miss correlation allows for making a distinction between the load operation that results in a cache miss and the other load operations that result in cache hits. The global cache miss history for the load miss looks like '0111' where a '0' denotes a cache miss and a '1' denotes a cache hit. The probability for a cache miss for that static load given its global cache miss history '0111' then equals 100%. The probability for a cache hit for that same static load then is 0% for the other global cache miss histories that occur for that load, namely '1011', '1101' and '1110'. By doing so, a more accurate statistical profile is collected, and a more representative synthetic trace can be generated.

4 Experimental setup

We used SimpleScalar/Alpha v3.0 in our experiments. The benchmarks along with their reference inputs used in this study are the SPECcpu2000 benchmarks. In order to reduce the simulation time when validating our improved statistical simulation framework, we considered 100M single (and early) simulation points as determined by SimPoint [6]. The processor model we use in this paper is detailed in Table 1.

5 Performance prediction

We now evaluate the performance prediction accuracy for statistical simulation enhanced with memory data flow modeling. Figure 2 shows the procentual performance prediction error of the SPECcpu2000 benchmarks for the baseline processor configuration. The IPC prediction error is computed as

$$IPC\ prediction\ error = \frac{IPC_{stat\_sim} - IPC_{det\_sim}}{IPC_{det\_sim}},$$

processor width	8 issue width, 8 decode width (fetch speed = 2), 8 commit width
IFQ	32-entry instruction fetch queue
RUU/LSQ	128/32 entries
functional units	8 int ALU's, 4 load/store units, 2 fp adders, 2 int and 2 fp mult/div units
I-cache	8KB, 2-way set-associative, 32-byte block, 2 cycles access latency
D-cache	16KB, 4-way set-associative, 32-byte block, 2 cycles access latency
unified L2 cache	1MB, 4-way set-associative, 64-byte block, 20 cycles access latency
I-/D-TLB	32-entry, 8-way set-associative, 4KB pages
memory	150 cycle round trip access
branch predictor	8K-entry hybrid predictor
speculative update	at dispatch time
misprediction penalty	14 cycles

Table 1: Baseline configuration used in this paper.

with IPC_{stat_sim} and IPC_{det_sim} the IPC for statistical simulation and detailed simulation, respectively. A positive error reflects an overestimation whereas a negative error reflects an underestimation. Figure 2 shows two bars per benchmark:

- The prior work bar corresponds to the previously proposed statistical simulation framework including the SFG as described in [1];
- The second bar shows the SFG enhanced with all three enhancements: delayed hits, load forwarding and cache miss correlation.

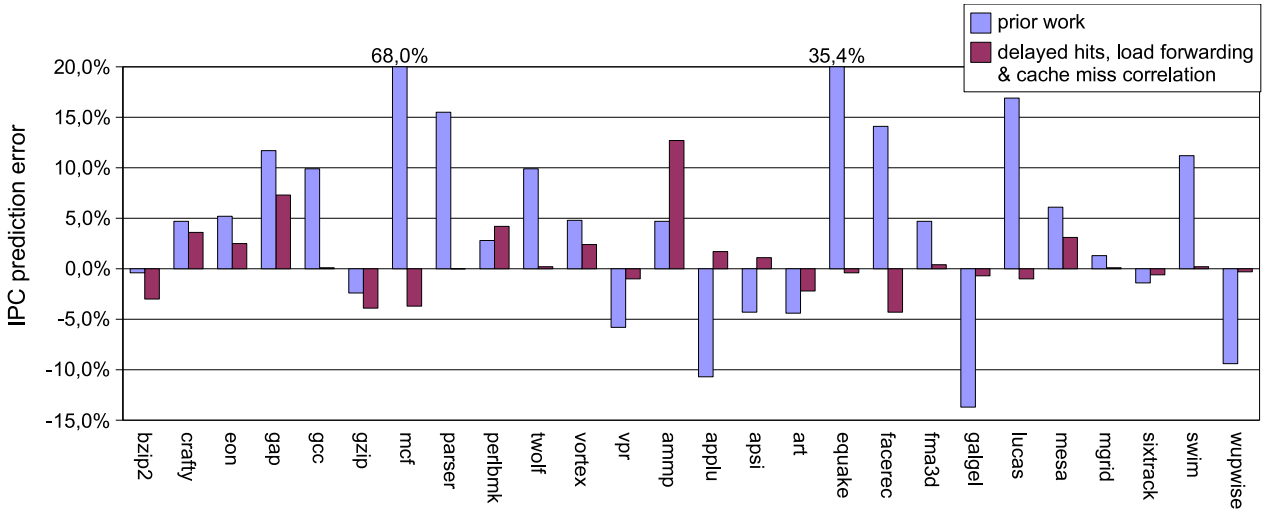

Figure 2: The proposed memory data flow modeling yields good accuracy for the baseline processor configuration.

As shown by the rightmost bars in Figure 2 the result is a highly accurate statistical simulation framework. The average prediction error goes down from 10.7% to 2.3%. The maximum error is observed for ammp (12.7%) which is substantially lower than the high errors observed for prior statistical simulation approaches, see for example mcf (68%). (Note that even without the outliers mcf and equake the average IPC prediction error goes down from 7.3% to 2.4%.)

Another note we would like to make is that for some benchmarks the IPC prediction error actually increases with improved memory data flow modeling, see for example bzip2, gzip, perlbmk and ammp. The reason is that in the original statistical simulation approaches, the inaccurate modeling of one program characteristic in one direction is averaged out with an inaccurate modeling of another program characteristic in the opposite direction. As a result, if memory data flow is now modeled more accurately, the inaccurate modeling of other program characteristics may become apparent.

6 Summary

Statistical simulation is a very fast simulation technique that only requires in the order of a million instructions per benchmark to make an accurate performance estimate. Previous work on statistical simulation however considered simple memory data flow models. In this paper we proposed to more accurately model memory data flow and we show how to do that. We model delayed hits, load forwarding and cache miss correlation. Our experimental results using the SPECcpu2000 benchmarks show that significant reductions in IPC prediction errors are obtained by more accurately modeling memory data flow characteristics.

References

[1] 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 *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA-31)*, pages 350–361, June 2004.

[2] K. I. Farkas and N. P. Jouppi. Complexity/performance tradeoffs with non-blocking loads. In *Proceedings of the 21st Annual International Symposium on Computer Architecture (ISCA-21)*, pages 211–222, Apr. 1994.

[3] M. Johnson. *Superscalar Microprocessor Design*. Prentice Hall, 1991.

[4] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *Proceedings of the Eighth Annual International Symposium on Computer Architecture (ISCA-8)*, pages 81–87, May 1981.

[5] S. Nussbaum and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, pages 15–24, Sept. 2001.

[6] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT-2003)*, pages 244–256, Sept. 2003.