Improving Accuracy for Statistical Simulation

Davy Genbrugge, Lieven Eeckhout, Koen De Bosschere

Supervisor(s): Koen De Bosschere

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.

I. 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 collected 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. The main advantage of statistical simulation is that the synthetic trace is very small, only one million of instructions at most.

Previous work on statistical simulation however considers simple memory data flow modeling (only the cache miss rates are modeled). We show that it is important to model memory data flow more accurately. In addition we model: delayed hits, load forwarding, and the correlation that may exist between cache misses.

This paper is organised as follows. First we explain statistical simulation. Second we discuss the proposed memory data flow model. Then we show some results and finally we conclude.

II. 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.

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

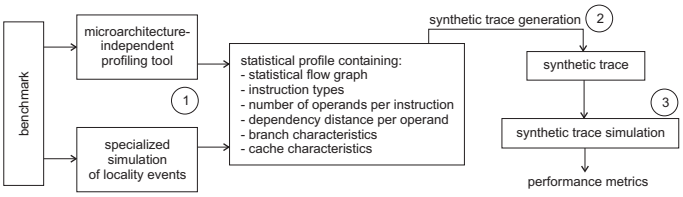

Fig. 1. Statistical simulation: general framework.

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.

III. Memory Data Flow Model

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.

A. Delayed hits

The caches in today's microprocessors typically are non-blocking caches [2], [4], which 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, e.g. delayed hits. A delayed hit is a load hit aliasing with a preceding load miss; the load hit then sees a part of the penalty of the preceding load miss when the first load hasn't completed, i.e. the data hasn't been loaded into the cache line. 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.

B. Load forwarding

Out-of-order execution of load instructions is another very important source for performance gain in out-of-order microprocessors. 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.

C. 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), e.g., when a loop walks over an array. Cache miss correlation results in specific cache miss patterns that have an important impact on overall performance.

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. By doing so, a more accurate statistical profile is collected, and a more representative synthetic trace can be generated.

IV. Preliminary Results

Figure 2 shows the procentual performance prediction error of the SPECcpu2000 benchmarks for our baseline processor configuration. The IPC prediction error is computed as

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

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:

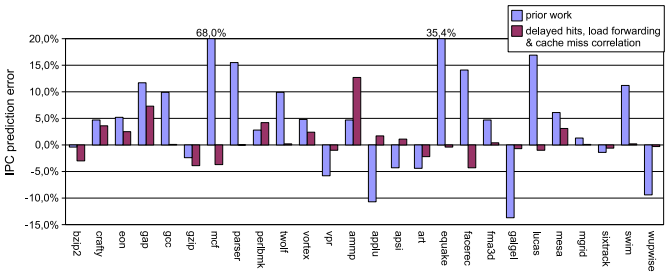

Fig. 2. The proposed memory data flow modeling yields good accuracy for the baseline processor configuration.

- 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.

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%.

V. Conclusion

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. 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.