Statistical Simulation of Chip Multiprocessors

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

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

ABSTRACT

According to Moore's law the number of transistors on a single chip doubles every 18 months. To respond to this law recent evolutions in computer architecture resulted in multiple processor cores integrated on a single chip, called a chip multiprocesor (CMP), sharing hardware components such as caches, I/O, *etc*. Current performance evaluation tools which allow for fast and accurate performance estimation, such as statistical simulation, merely focus on unicore systems. The need rises to extend these methodologies so that they become applicable for chip multiprocessor systems. In this paper we address some extensions that need to be made to statistical simulation so that it is still useful when designing a new CMP.

KEYWORDS: Performance modeling; Statistical simulation; CMP; Cache modeling

1 Introduction

Designing a new computer architecture is a very time consuming activity. A major reason is that architects heavily rely on simulators, that have a slowdown factor up to several orders of magnitude compared to real hardware execution. In addition a large number of parameters can be varied resulting in a huge design space. As such, detailed simulation is no longer feasible to explore the design space and find the design points of interest. As for chip multiprocessors, that are even more complex than traditional unicore systems, the simulation time increases and the design space grows. All this leads to the need for faster performance evaluation tools without loosing too much accuracy.

Recently, statistical simulation was shown to be a fast and accurate performance estimation technique, which makes it suitable for efficiently culling design spaces. The basic idea is to collect a number of program characteristics from a real program execution in a so called statistical profile. A synthetic trace is generated from this profile which is then simulated on a trace-driven statistical simulator. Current state of the art in statistical simulation merely targets unicore processors. With the increasing interest for multicore processors it is necessary to examine the applicability of statistical simulation for chip multiprocessors. In our current work, we are focusing on extending statistical simulation for modeling CMPs with shared L2 caches running multiprogrammed workloads.

We will first revisit the statistical simulation methodology for unicore processors. We will then describe the extensions needed to statistical simulation so that it becomes applicable for CMPs. Finally we conclude and take a glance at future extensions.

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 [Eeck04, Genb06]. We make a distinction between microarchitecture-*in*dependent characteristics and microarchitecture-dependent

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

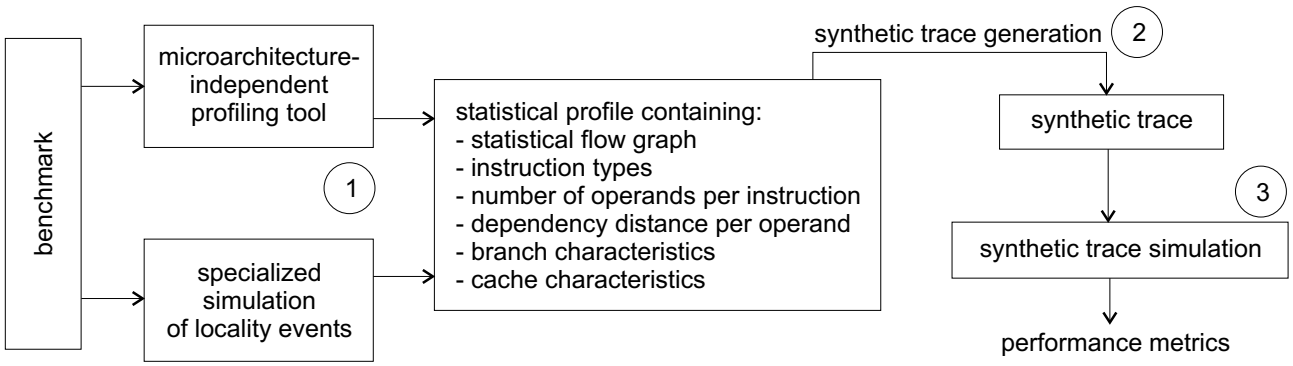

Figure 1: Statistical simulation: general framework.

characteristics. We then use this profile to generate a *synthetic trace* consisting of only a million of instructions. In the final step we simulate this synthetic trace on a statistical simulator which yields us performance metrics such as IPC. In the following subsections we discuss all three steps in more detail.

2.1 Statistical profiling

The key structure in the statistical profile is the *statistical flow graph (SFG)* [Eeck04] 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, both microarchitecture-*in*dependent characteristics and microarchitecture-dependent characteristics, along the nodes of the SFG. This allows for modeling program characteristics that are correlated with control flow path behavior. The microarchitecture-*in*dependent characteristics are:

- the instruction mix, classified into 12 synthetic instruction classes according to their semantics;
- for each instruction operand, the RAW dependency distance distribution [Eeck04];
- for each load instruction, the RAW memory dependency distance distribution [Genb06].

The microarchitecture-dependent characteristics, related to locality events, are:

- for a branch instruction, the probability for (i) a taken branch, (ii) a fetch redirection and (iii) a branch miss prediction;
- for each instruction, the instruction cache miss probabilities (instruction TLB, L1 instruction cache and L2 instruction cache);
- for each memory operation, separate cache miss probabilities (data TLB, L1 data cache and L2 data cache) depending on its global cache miss history;
- for each memory operation, the *missed cache line reuse distance* distribution [Genb06] depending on its global cache miss history.

2.2 Synthetic trace generation

The second step in the statistical simulation methodology is to generate a synthetic trace from the statistical profile; therefore we use a random number generator with the cumulative distribution functions of the various program characteristics. 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, I-cache miss info, D-cache miss info (in case of a memory operation), RAW memory dependency (in case of a load) and branch miss info (in case of a branch).

2.3 Synthetic trace simulation

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. Also the statistical simulator is very simple as it does not need to model branch predictors nor cache hierarchies. This type of information is directly obtained from the synthetic instructions.

3 Statistical simulation of CMPs

A typical configuration for a contemporary CMP is to have an L2 cache that is being shared by different processors. As a result, independent threads can affect each other's performance through the shared L2 cache. Labeling the synthetic memory operations as L2 hits and L2 misses as done for uniprocessor statistical simulation is no longer a viable solution because the L2 miss rate for each of the threads is affected by the other threads running on the CMP. In following subsections we will discuss how to extend statistical simulation for dealing with shared L2 caches in CMPs.

3.1 L2 cache profiling

For every memory reference, also those references that were L1 hits, we determine the L2 cache set that is to be accessed. In addition, we also determine the LRU stack distance in the given set. The maximum stack distance is $(a + 1)$ with a being the associativity of the L2 cache. Accessing the LRU stack at distance $(a+1)$ means that a miss occurred in an a-way set-associtive cache. In fact, for every static memory reference we collect the distribution of the accessed sets and the LRU stack depths. These statistics are collected conditionally on the history of basic blocks and conditionally on the cache miss history. However, we no longer keep track of a real cache miss history, we now include the LRU stack depth in the history instead of the L2 cache access outcome.

3.2 Statistical simulation of a shared L2 cache

In contrast to the statistical simulator for unicore processors we effectively simulate the shared L2 cache in the statistical simulator for CMPs. As such we can compute the L2 cache behavior for multiple threads sharing the L2 cache. A cache line in the shared L2 cache contains the following information:

- stored thread ID: This allows the statistical simulator to keep track of the owner of a cache line;
- valid bit stating whether the cache line is valid;
- cold bit stating whether the cache line has been accessed;
- a dirty bit stating whether the cache line has been written by a store operation — this only applies to write-back caches.

Simulating the shared L2 cache proceeds as follows. A reference labeled as an L1 miss accessing set s at LRU stack depth d in the L2 cache will be a hit in the L2 cache in case there are at least d valid

cache lines in set s for which the stored thread IDs equal the ID of the thread doing the access. The access will result in a miss if the above condition is not satisfied.

Note that since we are simulating the shared L2 cache an appropriate warmup of the cache is required, otherwise the L2 cache would suffer from a large number of cold misses. We could make the synthetic trace longer to solve this problem, however this would affect the usefulness of statistical simulation which is to provide fast performance estimates. To warmup the cache we first initialize all cache lines as being cold by setting the cold bit. We apply 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 unset. In the case a memory reference accesses a cache line at depth d and there are at least d cold cache lines in the given set, the reference is also considered a hit. The referenced cache line's cold bit as well as the cold bits of $d-1$ other cache lines are then unset. In addition, assuming inclusive caches, we also warm the L2 cache with L2 set and L2 LRU stack depth info from references that hit in the L1 caches.

4 Summary and Future Work

The topic of this paper was to extend statistical simulation for simulating multiprogrammed workloads running on chip multiprocessors. The main issue to deal with is that CMPs share structures such as L2 caches. Modeling this sharing in statistical simulation requires that the methodology be extended so that the interaction in the shared structures between the independent threads is appropriately modeled. We are currently in the evaluation process of the proposed mechanism.

References

[Eeck04] L. EECKHOUT, R. BELL, B. STOUGIE, K. DE BOSSCHERE, AND L. JOHN. Control Flow Modeling in Statistical Simulation for Accurate and Efficient Processor Design Studies. In *ISCA '04: Proceedings of the 31st Annual International Symposium on Computer Architecture*, June 2004.

[Genb06] D. GENBRUGGE, L. EECKHOUT, AND K. DE BOSSCHERE. Accurate Memory Data Flow Modeling in Statistical Simulation. In *ICS '06: Proceedings of the 20th ACM International Conference on Supercomputing*, June 2006.