Workload Generation for Microprocessor Performance Evaluation

SPEC PhD Award (Invited Abstract)

Luk Van Ertvelde
Ghent University, Belgium
lvertvel@elis.ugent.be

Thesis Supervisor: Lieven Eeckhout
Ghent University, Belgium
leeckhou@elis.ugent.be

ABSTRACT

This PhD thesis [1], awarded with the SPEC Distinguished Dissertation Award 2011, proposes and studies three workload generation and reduction techniques for microprocessor performance evaluation. (1) The thesis proposes code mutation, a novel methodology for hiding proprietary information from computer programs while maintaining representative behavior; code mutation enables dissemination of proprietary applications as benchmarks to third parties in both academia and industry. (2) It contributes to sampled simulation by proposing NSL-BLRL, a novel warm-up technique that reduces simulation time by an order of magnitude over state-of-the-art. (3) It presents a benchmark synthesis framework for generating synthetic benchmarks from a set of desired program statistics. The benchmarks are generated in a high-level programming language, which enables both compiler and hardware exploration.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: Modeling of computer architecture; C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling Techniques*

General Terms

Design, Performance, Measurement, Experimentation

Keywords

Workload Characterization, Workload Generation, Sampled Simulation

1. INTRODUCTION

Microprocessors have drastically advanced over the years, from scalar in-order execution processors to complex superscalar out-of-order and multi-core processors. The ever-increasing microarchitectural complexity necessitates benchmark programs to evaluate the performance of a new microprocessor, hence, organizations such as SPEC, EEMBC, etc., released standardized benchmark suites. Although this has streamlined the process of performance evaluation, computer architects and engineers still face several important benchmarking challenges.

1. Benchmarks should be representative for the (future) applications that are expected to run on the target computer system; however, it is not always possible to select a representative benchmark suite for at least three reasons. For one, standardized benchmark suites are typically derived from open-source programs because industry hesitates to share proprietary applications, and open-source programs have the advantage that they are portable across different platforms. The limitation though is that these benchmarks may not be representative for the real-world applications of interest. Secondly, existing benchmark suites are often outdated because the application space is constantly evolving and developing new benchmark suites is extremely time-consuming and costly. Finally, benchmarks are modeled after existing applications that may be less relevant by the time the product hits the market.

2. Coming up with a benchmark that is short-running yet representative is another major challenge. Contemporary application benchmark suites like SPEC CPU2006 execute trillions of instructions in order to stress contemporary and future processors in a meaningful way. If we also take into account that during microarchitectural research a multitude of design alternatives need to be evaluated, we easily end up with months or even years of simulation time. This may stretch the time-to-market of newly designed microprocessors. Hence, it is infeasible to simulate entire application benchmarks using detailed cycle-accurate simulators.

3. Finally, a benchmark should enable both (micro)architecture and compiler research and development. Although existing benchmarks satisfy this requirement, this is typically not the case for workload generation techniques that reduce the dynamic instruction count in order to address the simulation challenge. These techniques often operate on binaries which eliminates their utility for compiler exploration and instruction-set architecture exploration.

2. CONTRIBUTIONS

This dissertation [1] proposes three novel benchmark generation and reduction techniques to address the aforementioned challenges. In particular, code mutation addresses the proprietary nature of application codes; sampled simulation using NSL-BLRL reduces the long simulation times of contemporary benchmarks; finally, benchmark synthesis reduces simulation time and hides proprietary information in the reduced workloads.

2.1 Code Mutation

We first propose code mutation [2, 4] to stimulate sharing of proprietary applications between third parties in academia and industry. Code mutation is a novel methodology that mutates a propri-

ICPE'12, April 22-25, 2012, Boston, Massachusetts, USA
ACM 978-1-4503-1202-8/12/04.

etary application to complicate reverse engineering so that it can be distributed as an application benchmark among several parties. These benchmark mutants hide the functional semantics of proprietary applications while exhibiting similar performance characteristics. We therefore exploit two observations: (i) miss events have a dominant impact on performance on contemporary microprocessors, and (ii) many variables of contemporary applications exhibit invariant behavior at run time. More specifically, we compute program slices for memory access operations and/or control flow operations trimmed through constant value and branch profiles. Subsequently, we mutate the instructions not appearing in these slices through binary rewriting. The end result is a benchmark mutant that can serve as a proxy for the proprietary application during benchmarking experiments by third parties.

Our experimental results using SPEC CPU2000 and MiBench benchmarks show that code mutation is an effective approach that mutates (i) up to 90% of the binary, (ii) up to 50% of the dynamically executed instructions, and (iii) up to 35% of the at-run-time-exposed inter-operation data dependencies. In addition, the performance characteristics of the mutant are very similar to those of the proprietary application across a wide range of microarchitectures and hardware implementations.

Code mutation will mostly benefit companies that develop (embedded) microarchitectures and companies that offer (in-house built) services to remote customers. Such companies are reluctant to distribute their proprietary software. As an alternative, they can use mutated benchmarks as proxies for their proprietary software to help drive performance evaluation by third parties as well as guide purchasing decisions of hardware infrastructure. Being able to generate representative benchmark mutants without revealing proprietary information can also be an encouragement for industry to collaborate more closely with academia, i.e., it would make performance evaluation in academia more realistic and therefore more relevant for industry. Eventually, this may lead to more valuable research directions. In addition, developing benchmarks is both hard and time-consuming to do in academia, for which code mutation may be a solution.

2.2 Sampled Simulation: NSL-BLRL

Code mutation conceals the intellectual property of an application, but it does not lend itself to the generation of short-running benchmarks. Sampled simulation on the other hand reduces the simulation time of an application significantly. The key idea of sampled simulation is to simulate only a small sample from a complete benchmark execution in a detailed manner (a sample consists of one or more sampling units). The performance bottleneck in sampled simulation is the establishment of the microarchitecture state (caches, branch predictor, etc.) at the beginning of each sampling unit. The unknown microarchitecture starting image at the beginning of a sampling unit is often referred to as the cold-start problem.

We address the cold-start problem by proposing a new cache warmup method, namely NSL-BLRL [5, 6] which builds on No-State-Loss (NSL) and Boundary Line Reuse Latency (BLRL) for minimizing the cost associated with cycle-accurate processor cache hierarchy simulation in sampled simulation. The idea of NSL-BLRL is to establish the cache state at the beginning of a sampling unit using a checkpoint that stores a truncated NSL stream. NSL scans the pre-sampling unit and records the last reference to each unique memory location. This is called the least-recently used (LRU) stream. This stream is then truncated to form the NSL-BLRL warmup checkpoint by inspecting the sampling unit for determining how far in the pre-sampling unit one needs to go back to accurately warm up the cache state for the given sampling unit.

This approach yields several benefits over prior work: substantial simulation speedups compared to BLRL (up to 1.4× under fast-forwarding and up to 14.9× under checkpointing) and significant reductions in disk space requirements compared to NSL (on average 30%), for a selection of SPEC CPU2000 benchmarks.

2.3 HLL Benchmark Synthesis

Although code mutation can be used in combination with sampled simulation to generate short-running workloads that can be distributed to third parties without revealing intellectual property, there are a number of limitations. The most important limitation is that this approach operates at the assembly level, and as a result, it cannot be used for compiler exploration and ISA exploration purposes. We therefore propose a novel benchmark synthesis framework that generates synthetic benchmarks in a high-level programming language.

The benchmark synthesis framework [3, 4] aims at generating small but representative benchmarks that can serve as proxies for other applications without revealing proprietary information; and because the benchmarks are generated in a high-level language, they can be used to explore the architecture and compiler space. The methodology to generate these benchmarks comprises two key steps: (i) profiling a real-world (proprietary) application (that is compiled at a low optimization level) to measure its execution characteristics, and (ii) modeling these characteristics into a synthetic benchmark clone. To capture a program's control flow behavior in a statistical way, we introduce a new structure: the Statistical Flow Graph with Loop information (SFGL).

We demonstrate good correspondence between the synthetic and original applications across instruction-set architectures, microarchitectures and compiler optimizations, and we point out the major sources of error in the benchmark synthesis process. We verified using software plagiarism detection tools that the synthetic benchmark clones indeed hide proprietary information from the original applications.

We argue that our framework can be used for several applications: distributing synthetic benchmarks as proxies for proprietary applications, drive architecture and compiler research and development, speed up simulation, model emerging and hard-to-setup workloads, and benchmark consolidation.

3. REFERENCES

[1] L. Van Ertvelde. *Workload Generation for Microprocessor Performance Evaluation*. PhD thesis, Ghent University, Belgium, 2010.

[2] L. Van Ertvelde and L. Eeckhout. Dispersing proprietary applications as benchmarks through code mutation. In *The International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 201–210, 2008.

[3] L. Van Ertvelde and L. Eeckhout. Benchmark synthesis for architecture and compiler exploration. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 106–116, 2010.

[4] L. Van Ertvelde and L. Eeckhout. Workload reduction and generation techniques. *IEEE Micro*, 30(6):57–65, 2010.

[5] L. Van Ertvelde, F. Hellebaut, and L. Eeckhout. Accurate and efficient cache warmup for sampled processor simulation through NSL-BLRL. *The Computer Journal*, 51(2):192–206, 2008.

[6] L. Van Ertvelde, F. Hellebaut, L. Eeckhout, and K. De Bosschere. NSL-BLRL: Efficient cache warmup for sampled processor simulation. In *Proceedings of the Annual Simulation Symposium (ANSS)*, pages 168–177, 2006.