Accurate Throughput Prediction of Basic Blocks on Recent Intel Microarchitectures

Extended Abstract

Andreas Abel and Jan Reineke
{abel,reineke}@cs.uni-saarland.de
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany

ABSTRACT

Tools to predict the throughput of basic blocks on a specific microarchitecture are useful to optimize software performance and to build optimizing compilers. In recent work, several such tools have been proposed. However, the accuracy of their predictions has been shown to be relatively low.

To a significant degree, these inaccuracies are due to elements and parameters of the pipelines of recent CPUs that are not taken into account by previous tools. A primary reason for this is that the necessary details are often undocumented. In this paper, we build more precise models of relevant components by reverse engineering using microbenchmarks. Based on these models, we develop a simulator for predicting the throughput of basic blocks. In addition to predicting the throughput, our simulator also provides insights into how the code is executed.

Our tool supports all Intel Core microarchitecture generations released in the last decade. We evaluate it on an improved version of the BHive benchmark suite. On many recent microarchitectures, its predictions are more accurate than the predictions of state-of-the-art tools by more than an order of magnitude.

1 INTRODUCTION

Techniques to predict the performance of software are, for example, required to build compilers, to optimize code for specific platforms, and to analyze the worst-case execution time of software in safety-critical systems. Such techniques are also useful in the area of security, where performance differences may be used for side-channel attacks.

A specific problem in this area that has received a lot of attention in recent work is to predict the performance of basic blocks, under the assumption that all memory accesses lead to cache hits. A popular performance predictor from this class is Intel's IACA [9]. It is, for example, used by the Kerncraft tool [8] to construct Roofline [17] and ECM [16] models. It has also been used by Bhattacharyya et al. [4] to build a speculative execution attack.

However, IACA is closed source, does not support recent microarchitectures, and has been discontinued by Intel in 2019. A number of alternatives were proposed in recent work, such as OSACA [11, 12], llvm-mca [14], CQA [5], Ithemal [13], and DiffTune [15].

The accuracy of the throughput predictions of previous tools compared to measurements on the actual hardware has been shown to be relatively low; Chen et al. [6] found that the average error of the predictions of previous tools compared to measurements is between 9% and 36%.

We have identified two main reasons that contribute to these discrepancies between measurements and predictions:

(1) the microarchitectural models of previous tools are not detailed enough;
(2) evaluations were partly based on unsuitable benchmarks, and on incompatible throughput definitions.

In this paper, we address both of these issues. We first develop a pipeline model that is significantly more detailed than those of previous tools. Based on this model, we then implement a simulator that can compute throughput predictions for basic blocks.

To compare the predictions of our simulator to previous tools, we propose an improved version of the BHive benchmark suite [6]. We show that on this benchmark suite, our tool is significantly more accurate than all previous tools.

2 PARAMETRIC PIPELINE MODEL

We have developed a new pipeline model that is applicable to all Intel Core microarchitectures released in the last ten years; we discovered that the differences between these microarchitectures can be captured by a relatively small number of parameters.

Our model is significantly more detailed than any previous models described in the literature. As the information required to build such a model is often undocumented, we have developed microbenchmarks for reverse engineering the necessary details using hardware performance counters. For evaluating these microbenchmarks, we use nanoBench [2] and an extension to nanoBench that provides cycle-by-cycle performance data, similar to Brandon Falk's "Sushi Roll" approach [7].

In particular, we developed detailed models of the following components, which were mostly ignored by previous work: the decoders and predecoders, the *Decoded Stream Buffer* (DSB), the *Microcode Sequencer* (MS), the *Loop Stream Detector* (LSD), micro and macro fusion, the *renamer* (including *move elimination*), the *reorder buffer*, the *scheduler*, and the assigment of µops to execution ports.

3 THROUGHPUT PREDICTOR

Based on the model from the previous section and latency, throughput, and port usage data from uops.info [1], we have implemented a tool, called uiCA ("uops.info Code Analyzer"), that simulates the execution of basic blocks on Intel Core microarchitectures. In addition to providing throughput predictions, uiCA can also generate a table that contains the port usage for each instruction, and it can output a timeline that shows what happens in each cycle.

4 BENCHMARKS AND MEASUREMENTS

To evaluate and compare our simulator to previous approaches, we need a set of suitable benchmarks. Chen et al. [6] proposed the BHive benchmark suite, which is designed specifically to evaluate basic block throughput predictors on x86 systems. The BHive suite contains basic blocks that were extracted from applications from different domains, including numerical computation, databases, compilers, machine learning, and cryptography. In addition to that, Chen et al. in the same paper also propose a profiling tool to measure the throughput of such basic blocks using hardware performance counters. Chen et al.'s tool was used to measure the throughputs of the basic blocks that were used to train Ithemal [13] and DiffTune [15].

While Chen et al.'s benchmark suite and measurement framework appear to be suitable for evaluating the work presented in our paper, we discovered a number of issues with their approach that can lead to incorrect or misleading results.

In particular, the notion of throughput their measurement tool is based on is different from the notion of throughput that previous tools like IACA use. IACA considers the throughput of a basic block to be the average number of clock cycles per iteration when executing the basic block repeatedly in a loop at a steady state, assuming that all memory accesses are cache hits. Thus, it is intended to be used to analyze basic blocks that end with a branch instruction that jumps back to the beginning of the loop; all examples in IACA's user guide [9] are of this form. On the other hand, Chen et al.'s measurement framework only considers basic blocks that do not end with a branch instruction. They consider the throughput to be the average number of cycles (per execution of the basic block) when unrolling the basic block a sufficient number of times to reach a steady state. An important difference between these two definitions is that in the first case, the µops of many benchmarks are delivered by the DSB or the LSD (see Section 2), whereas in Chen et al.'s measurement tool, all µops have to go through the decoders, which can be significantly slower.

In order to enable a more meaningful comparison with previous tools, we have created a variant of the BHive benchmark suite in which all benchmarks end with a decrement instruction followed by a branch instruction. We omitted several classes of benchmarks from this suite which either have an ambiguous throughput or which are not meaningful when executed in a loop. In the following, we will call the resulting benchmark suite $BHive_L$; it contains more than 330,000 benchmarks.

We have also extended Chen et al.'s measurement tool to support evaluating the benchmarks in $BHive_L$, and we have made several improvements that enable more accurate and less biased measurements.

5 EVALUATION

We compare the predictions of our tool on $BHive_L$ to several popular previous tools on the Skylake microarchitecture (for Ithemal, we omitted the branch instruction from the benchmarks, as this tool cannot analyze code that ends with a branch instruction). For this comparison, we use the same metrics that were used in [6, 15]:

- The mean absolute percentage error (MAPE) of the predictions relative to the measurements.
- Kendall's Tau coefficient [10], which is a measure for how well the pairwise ordering is preserved.

Table 1: Comparison of different tools on $BHive_L$

	$BHive_L$	
Predictor	**MAPE**	**Kendall's Tau**
uiCA	0.38%	0.9894
Ithemal	13.66%	0.7582
IACA 3.0	14.26%	0.8290
IACA 2.3	8.42%	0.8477
OSACA	11.25%	0.8045
llvm-mca-10	12.01%	0.8015
DiffTune	104.88%	0.6426
CQA	7.44%	0.8847
Baseline	10.03%	0.7999

As a baseline for the throughput of a benchmark, we use

$$max\left(1, \frac{n-1}{4}, \frac{m_r}{2}, m_w\right)$$

Here, n is the number of instructions in the benchmark, and m_r and m_w are the number of memory read and write accesses performed by the benchmark. For the Skylake microarchitecture, this baseline constitutes a lower bound on the execution time of basic blocks that end with a decrement and branch instruction, as at most 4 µops can be issued per cycle (we use $n-1$ instead of n as the decrement and branch instructions are macro-fused), and at most two memory read operations and one memory write operation can be performed per cycle; we use 1 as an additional lower bound, as the benchmarks in $BHive_L$ cannot run faster than one iteration per cycle due to the read-after-write dependency of the decrement operation.

Table 1 shows the results of our evaluation. The MAPE of our tool is lower by more than an order of magnitude compared to the best previous tool; also, Kendall's Tau coefficient is significantly higher.

Interestingly, the predictions of several previous tool are below the baseline. For the machine learning-based approaches, retraining them on our modified benchmark suite might improve their accuracy.

We also evaluated our tool on eight other Intel Core microarchitectures. On all of these, it provided the most accurate predictions. In many cases, the MAPE was lower by an order of magnitude or more compared to previous tools.

6 CONCLUSIONS

We have developed a new simulator to predict the throughput of basic blocks. Our evaluation demonstrates that modeling microarchitectural details that were considered to be rather insignificant by previous work, is in fact crucial for accurate predictions.

Unlike recently proposed machine learning-based techniques, our approach is not based only on end-to-end measurements, which gives a higher confidence that our model is not overfitting to measurement errors. Our results show that training and evaluating predictors on measurements obtained with the same, potentially biased, measurement methodology may result in misleading conclusions.

7 FURTHER INFORMATION

More details on the work described in this extended abstract can be found in [3]. Our tool uiCA is available as open source on GitHub[1].

[1] https://github.com/andreas-abel/uiCA

REFERENCES

[1] Andreas Abel and Jan Reineke. 2019. uops.info: Characterizing Latency, Throughput, and Port Usage of Instructions on Intel Microarchitectures. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Providence, RI, USA, April 13-17, 2019* (Providence, RI, USA) *(ASPLOS '19)*. ACM, 673–686. https://doi.org/10.1145/3297858.3304062

[2] Andreas Abel and Jan Reineke. 2020. nanoBench: A Low-Overhead Tool for Running Microbenchmarks on x86 Systems. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. https://doi.org/10.1109/ISPASS48437.2020.00014

[3] Andreas Abel and Jan Reineke. 2021. Accurate Throughput Prediction of Basic Blocks on Recent Intel Microarchitectures. *CoRR* abs/2107.14210 (2021). arXiv:2107.14210 [cs.PF] https://arxiv.org/abs/2107.14210

[4] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. 2019. SMoTherSpectre: Exploiting Speculative Execution Through Port Contention. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. ACM, New York, NY, USA, 785–800. https://doi.org/10.1145/3319535.3363194

[5] A. S. Charif-Rubial, E. Oseret, J. Noudohouenou, W. Jalby, and G. Lartigue. 2014. CQA: A code quality analyzer tool at binary level. In *21st International Conference on High Performance Computing (HiPC)*. 1–10. https://doi.org/10.1109/HiPC.2014.7116904

[6] Yishen Chen, Ajay Brahmakshatriya, Charith Mendis, Alex Renda, Eric Atkinson, Ondrej Sykora, Saman Amarasinghe, and Michael Carbin. 2019. BHive: A Benchmark Suite and Measurement Framework for Validating x86-64 Basic Block Performance Models. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. http://groups.csail.mit.edu/commit/papers/19/ithemal-measurement.pdf

[7] Brandon Falk. 2019. Sushi Roll: A CPU research kernel with minimal noise for cycle-by-cycle micro-architectural introspection. https://gamozolabs.github.io/metrology/2019/08/19/sushi_roll.html

[8] Julian Hammer, Jan Eitzinger, Georg Hager, and Gerhard Wellein. 2017. Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels. In *Tools for High Performance Computing 2016*. Springer International Publishing, 1–22. https://doi.org/10.1007/978-3-319-56702-0_1

[9] Intel Corporation 2017. *Intel Architecture Code Analyzer User's Guide*. Intel Corporation. https://software.intel.com/content/dam/develop/external/us/en/documents/intel-architecture-code-analyzer-3-0-users-guide-157552.pdf Document Number: 321356-001US.

[10] M. G. Kendall. 1938. A New Measure of Rank Correlation. *Biometrika* 30, 1/2 (1938), 81–93. https://doi.org/10.2307/2332226

[11] Jan Laukemann, Julian Hammer, Georg Hager, and Gerhard Wellein. 2019. Automatic Throughput and Critical Path Analysis of x86 and ARM Assembly Kernels. In *IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. https://doi.org/10.1109/PMBS49563.2019.00006

[12] J. Laukemann, J. Hammer, J. Hofmann, G. Hager, and G. Wellein. 2018. Automated Instruction Stream Throughput Prediction for Intel and AMD Microarchitectures. In *IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE Computer Society, Los Alamitos, CA, USA, 121–131. https://doi.org/10.1109/PMBS.2018.8641578

[13] Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. 2019. Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, Long Beach, California, USA, 4505–4515. http://proceedings.mlr.press/v97/mendis19a.html

[14] LLVM Project. 2021. llvm-mca — LLVM Machine Code Analyzer. https://llvm.org/docs/CommandGuide/llvm-mca.html

[15] A. Renda, Y. Chen, C. Mendis, and M. Carbin. 2020. DiffTune: Optimizing CPU Simulator Parameters with Learned Differentiable Surrogates. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 442–455. https://doi.org/10.1109/MICRO50266.2020.00045

[16] Holger Stengel, Jan Treibig, Georg Hager, and Gerhard Wellein. 2015. Quantifying Performance Bottlenecks of Stencil Computations Using the Execution-Cache-Memory Model. In *Proceedings of the 29th ACM International Conference on Supercomputing* (Newport Beach, California, USA) *(ICS '15)*. Association for Computing Machinery, New York, NY, USA, 207–216. https://doi.org/10.1145/2751205.2751240

[17] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. https://doi.org/10.1145/1498765.1498785