Improving Instruction Fetch Throughput With Dynamic Structure of Fetch Pipeline

Reoma Matsuo
Nagoya University
matsuo@ando.nuee.nagoya-u.ac.jp

Ryota Shioya
The University of Tokyo
shioya@ci.i.u-tokyo.ac.jp

Hideki Ando
Nagoya University
ando@nuee.nagoya-u.ac.jp

ABSTRACT

The size of instruction working set has been growing in recent workloads such as server applications and WEB applications written in JavaScript, and instruction cache misses in such applications cause significant performance degradation [1, 2]. Unlike data cache misses, instruction cache misses are one of the critical performance bottlenecks in the workloads described above because out-of-order execution cannot hide an instruction cache miss latency.

Various instruction prefetchers have been proposed for mitigating the performance degradation due to instruction cache misses. Most of the state-of-the-art prefetchers achieve high coverage and accuracy by leveraging the characteristic that the stream of instruction cache misses are reproducible, but the storage costs for training the stream is significant [3, 4]. Prefetchers based on a branch target buffer or return address stack have been proposed to prefetch with a small hardware cost, but they cannot capture complex cache miss patterns[5, 6].

We propose a novel method to improve instruction fetch throughput by dynamically controlling a pipeline structure. The proposed method consists of the following two parts: 1) a pipeline assuming miss and 2) dynamic switching of fetch pipelines.

First, we propose a new pipeline structure called a pipeline assuming miss. The conventional instruction fetch pipeline

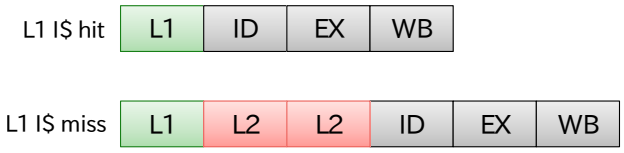

Figure 1: A conventional pipeline (a pipeline assuming hit)

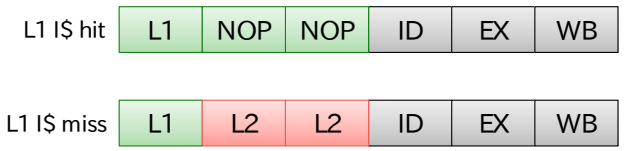

Figure 2: A pipeline assuming miss

MICRO51 SRC, October 2018, Fukuoka City, Japan
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

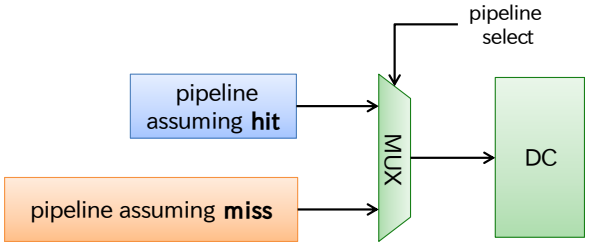

Figure 3: Proposed method: This architecture switches instruction fetch pipeline between a conventional pipeline (assuming hit) and a pipeline assuming miss

(we refer to this pipeline structure as a pipeline assuming hit) immediately sends instructions to the next stage when instructions are fetched from the L1 cache (Figure 1 depicts an instruction pipeline of a conventional pipeline). This pipeline structure does not assume L2 cache accesses, that is, this pipeline does not integrate L2 cache access stages into the fetch pipeline. Therefore, when an L1 instruction cache miss occurs, this pipeline has to stall the pipeline and wait for completion of the L2 cache access.

On the other hand, the pipeline assuming miss integrates L2 cache access stages into the pipeline structure (Figure 2 depicts an instruction pipeline of a pipeline assuming miss). Even when an L1 instruction cache hits, all instructions spend an L2 cache access latency in the pipeline before they are sent to the subsequent stage. As a result, even if an L1 cache miss is caused and an L2 cache is accessed, the subsequent instructions are fetched without stalling the pipeline.

However, in the pipeline assuming miss, since all instructions spend an L2 cache access latency, the pipeline length is increased and the branch misprediction penalty is significantly increased compared with that in the conventional pipeline.

To tackle this problem, we propose a method that dynamically switches between the pipeline assuming miss and the pipeline assuming hit (see Figure 3). The proposed method switches the pipeline on instruction cache miss and branch misprediction, and it can completely avoid the performance degradation due to the increased misprediction penalty by the pipeline assuming miss. We show that the proposed switching is ideal, which achieves the maximum performance improvement.

Our contributions are summarized as follows:

- We proposed a novel pipeline structure to mitigate performance degradation due to instruction cache misses. By dynamically controlling the fetch pipeline structure, this method avoids pipeline stalls on instruction cache misses and can significantly improve fetch throughput. Moreover, since the proposed method is orthogonal to an instruction prefetcher, our proposed method can work synergistically with prefetchers.
- The proposed method does not include any complex hardware for training access patterns and can be implemented with the simple switching control hardware. Moreover, while prefetchers can cause incorrect prefetches and they can degrade performance, our proposed method does not have such side-effects.
- We evaluated our proposed method with a full-system simulator. Evaluation results show that the performance improvement is up to 10.2% in SPECCPU 2006 and 28.5% in TPC-C.

KEYWORDS

Instruction Pipeline, Instruction Cache, Branch Prediction

ACM Reference Format:
Reoma Matsuo, Ryota Shioya, and Hideki Ando. 2018. Improving Instruction Fetch Throughput With Dynamic Structure of Fetch Pipeline. In *Proceedings of MICRO 51 ACM Student Research Competition (MICRO51 SRC)*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

REFERENCES

[1] L. Spracklen, Y. Chou, and S. G. Abraham, "Effective instruction prefetching in chip multiprocessors for modern commercial applications," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, February 2005, pp. 225–236.

[2] Y. Zhu, D. Richins, M. Halpern, and V. J. Reddi, "Microarchitectural implications of event-driven server-side web applications," in *Proceedings of the 48th International Symposium on Microarchitecture*, December 2015, pp. 762–774.

[3] M. Ferdman, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Temporal instruction fetch streaming," in *Proceedings of the 41st International Symposium on Microarchitecture*, November 2008, pp. 1–10.

[4] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive instruction fetch," in *Proceedings of the 44th International Symposium on Microarchitecture*, December 2011, pp. 152–162.

[5] A. Kolli, A. Saidi, and T. F. Wenisch, "RDIP: Return-address-stack directed instruction prefetching," in *Proceedings of the 46th International Symposium on Microarchitecture*, December 2013, pp. 260–271.

[6] R. Kumar, C. C. Huang, B. Grot, and V. Nagarajan, "Boomerang: A metadata-free architecture for control flow delivery," in *IEEE International Symposium on High Performance Computer Architecture (2017)*, February 2017, pp. 493–504.