Accurate and Fast Performance Modeling of Processors with Decoupled Front-end

Yuya Degawa, Toru Koizumi, Tomoki Nakamura, Ryota Shioya,
Junichiro Kadomoto, Hidetsugu Irie, Shuichi Sakai
Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
Email: {degawa, koizumi, tomokin, kadomoto, irie, sakai}@mtl.t.u-tokyo.ac.jp, shioya@ci.i.u-tokyo.ac.jp

***Abstract*—Various techniques, such as cache replacement algorithms and prefetching, have been studied to prevent instruction cache misses from becoming a bottleneck in the processor front-end. In such studies, the goal of the design has been to reduce the number of instruction cache misses. However, owing to the increasing complexity of modern processors, the correlation between reducing instruction cache misses and reducing the number of executed cycles has become smaller than in previous cases. In this paper, we propose a new guideline for improving the performance of modern processors. In addition, we propose a method for estimating the approximate performance of a design two orders of magnitude faster than a full simulation each time the designers modify their design.**

***Index Terms*—instruction fetch, modeling techniques**

I. Introduction

Various techniques, such as cache replacement algorithms and prefetching, have been studied to prevent instruction cache misses from becoming a bottleneck in the processor front-end. In such studies, reducing the number of instruction cache misses has often been focused on evaluations. Although the metrics used by the studies vary, such as the number of cache misses per 1000 instructions [1]–[3], as well as cache hit rate [4], cache miss rate [5], cache miss reduction rate [6], and coverage [7], they all focus on whether the number of cache misses is reduced.

However, owing to the increasing complexity of modern processors, the correlation between reducing the number of instruction cache misses and reducing the number of executed cycles has increasingly decreased. Indeed, in classical processors, an instruction cache miss can immediately lead to a stall, which can be a performance-critical issue. By contrast, this is not always the case with modern processors. In the front-end of a modern processor, an instruction cache access is decoupled from branch prediction [8]–[10]. In such processors, subsequent instructions continue to be fetched after an instruction cache miss, and subsequent instruction cache misses are handled in overlap with the previous instruction cache miss until the next branch misprediction. Therefore, in modern processors, a reduction in the number of instruction cache misses does not always lead to a reduction in the number of executed cycles.

Hence, designers of instruction caches are required to verify the performance by simulating the entire processor pipeline, not just the behavior of the instruction cache. Although simulations of the entire processor pipeline indicate the performance of the designed processor, they provide the designer no guidance for increasing the performance. Hence, the designers repeat the simulation and explore a design space based on their heuristics. In addition, simulating the entire processor pipeline wastes time by simulating components unrelated to the instruction cache.

In this paper, instead of reducing the number of instruction cache misses, we propose a new guideline for improving the performance of modern processors. We also propose a method for estimating the processor performance without simulating the entire processor pipeline when the instruction cache design is modified. Using this method, once the designers have obtained the baseline performance by simulating the entire processor pipeline, they can estimate the processor performance without repeating simulations of the entire processor pipeline.

Our contributions are as follows.

- We show that reducing the number of instruction cache misses is insufficient for designing instruction caches in modern processors.
- We propose a new guideline for achieving a performance improvement instead of reducing the number of instruction cache misses.
- Based on the guideline, we propose a method for estimating the performance after modifying the instruction cache design. Using our proposed method, once designers obtain the baseline performance by simulating the entire processor pipeline, they can estimate the change in performance by simulating only the behavior of the instruction cache. In our preliminary evaluation, the simulation time for the instruction cache alone was $\sim$ 3s on average, and that for the entire processor pipeline was $\sim$650s on average.
- Our proposed method estimates the performance when applying an instruction prefetcher with an average error of 1.1% and a maximum error of 4.1%, whereas the estimate using the number of instruction cache misses achieved an average error of 9.0% and a maximum error of 43.1%.

II. Front-end of Modern Processors

In classical processors, because branch prediction and instruction cache accesses operate in synchronization, instruction

This work was partially supported by JST PRESTO JPMJPR20M1 and JSPS KAKENHI JP20H04153, JP20J22752, and JP21J11687.

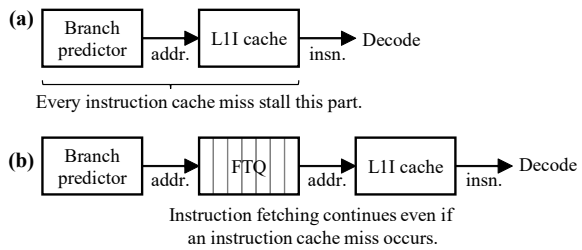


Fig. 1. Front-end behavior of (a) classical and (b) modern processors.

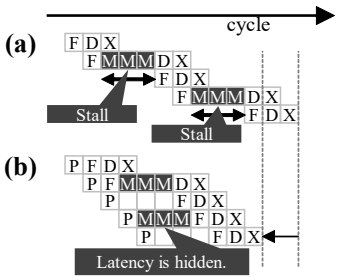


Fig. 2. Pipeline diagrams of (a) a classical processor and (b) a modern processor. P, F, D, X, and M represent instruction fetching according to the addresses in the FTQ (prefetching), demand fetching, instruction decode, execution, and processing of an instruction cache miss, respectively.

cache misses immediately stall the pipeline. Fig. 1a shows its structure and behavior. In such processors, each instruction cache miss delays the instruction fetching, as shown in Fig. 2a.

By contrast, modern processors have a front-end where instruction cache accesses are decoupled from a branch prediction [5], [8]–[11]. Fig. 1b shows its structure and behavior. The branch predictor predicts instruction addresses, which are then inserted into a queue called fetch target queue (FTQ). According to the addresses in the FTQ, instruction fetching from the instruction cache proceeds sequentially.

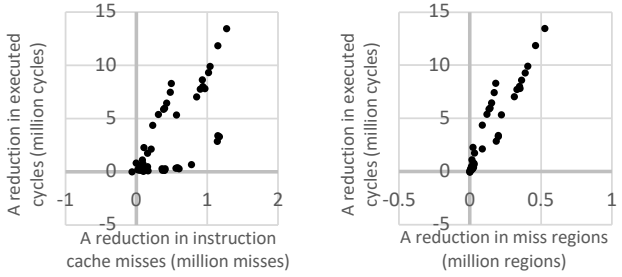
In this decoupled front-end, an instruction cache miss stalls neither branch prediction nor instruction cache accesses, whereas it stalls both in classical processors. Even if an instruction cache miss occurs, the branch predictor continues to predict until the FTQ is full. In addition, addresses following the missed address are sequentially retrieved from the FTQ, and instruction cache accesses continue using those addresses.

If a miss occurs in the subsequent cache access, the latency of the first miss hides that of the subsequent miss, as shown in Fig. 2b, thus mitigating the increase in processing time. Considering that modern processors have a decoupled front-end, researchers at Arm have argued that the study of instruction prefetching should be based on a decoupled front-end [12].

III. Design Guideline for Improved Performance

A. Gap between Reduction in Instruction Cache Misses and Reduction in Executed Cycles

Instruction cache designers have focused on reducing instruction cache misses as a guideline for improving the performance. However, we found that the guideline for reducing the number of instruction cache misses does not necessarily



(a) A reduction in cache misses. (b) A reduction in miss regions.

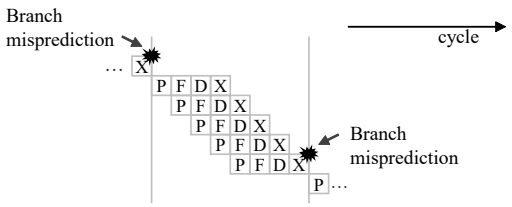
Fig. 3. Relationship between each metric and a reduction in executed cycles.

achieve a performance improvement in modern processors, where instruction cache accesses are decoupled from branch prediction. Fig. 3a shows the relationship between a reduction in instruction cache misses and a reduction in executed cycles when adding return-address-stack directed instruction prefetching (RDIP) [1] in a decoupled front-end. The simulation environment and workload are described in Section V-A. Some points are distributed around the horizontal axis in the figure, indicating that reducing the number of instruction cache misses does not reduce the number of executed cycles for some workloads. Hence, the strategy of reducing the number of instruction cache misses is insufficient to achieve a performance improvement in modern processors with a decoupled front-end.

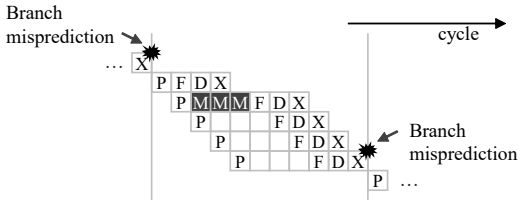
B. Metric Showing a High Correlation with a Reduction in Executed Cycles

1) Hit and Miss Regions: To clarify the relationship between instruction cache misses and performance in processors with a decoupled front-end, we focused on the relationship between branch mispredictions and instruction cache misses. Herein, "branch misprediction" includes a branch target buffer (BTB) miss. As mentioned above, once an instruction cache miss occurs in a decoupled front-end, the latency of the subsequent instruction cache misses is hidden. This hiding of the latency is effective until the next branch misprediction. When a branch misprediction is detected during the decoding or execution stage, the pipeline, including the FTQ, is flushed. Then fetching instructions on the correct path starts in the next cycle. Because the FTQ is empty at this time, the interval between the P-stage and F-stage, which has widened after the first cache miss shown in Fig. 2b, becomes zero. Thus, the latency of the next cache miss increases the number of executed cycles.

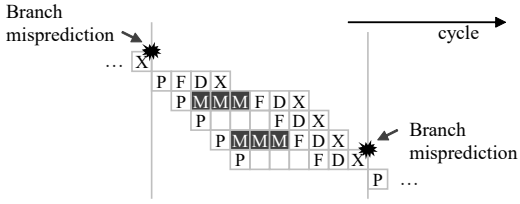
Based on this observation, we consider one branch misprediction to the next branch misprediction as a *region*. We call a region not containing instruction cache misses, as shown in Fig. 4a, a *hit region*, and a region containing one or more instruction cache misses, as shown in Figs. 4b and 4c, a *miss region*. As Figs. 4a and 4b show, if there is one instruction cache miss in a region, the length of the processing time increases based on the latency of lower cache access in comparison to when no instruction cache misses occur. As



(a) A region containing no instruction cache miss. This is a hit region.



(b) A region containing an instruction cache miss. This is a miss region.



(c) A region containing two instruction cache misses. This is a miss region.

Fig. 4. Concept of regions. P, F, D, X, and M represent instruction fetching according to the addresses in the FTQ (prefetching), demand fetching, instruction decode, execution, and processing of an instruction cache miss, respectively.

shown in Figs. 4b and 4c, if there are two or more instruction cache misses in a region, the processing time is the same as one instruction cache miss.

2) Reductions in Miss Regions and Executed Cycles: Based on the above insights, we found that to improve the performance of processors with a decoupled front-end, it is crucial to reduce the number of miss regions rather than aiming to reduce the number of instruction cache misses. To demonstrate this, we depict the relationship between a reduction in miss regions and a reduction in executed cycles when adding RDIP [1] in a decoupled front-end, as shown in Fig. 3b. The simulation environment and workload are described in Section V-A. Unlike in Fig. 3a, the points in Fig. 3b do not concentrate on the horizontal axis. This result means that the number of executed cycles can certainly be reduced by reducing the number of miss regions.

IV. Performance Estimation

We propose a method for estimating the reduction in executed cycles from a reduction in miss regions. This method allows designers to estimate the performance of a processor using a simple simulator that reproduces only the instruction cache. In the following discussion, we introduce some symbols. Let MR and HR be the set of miss regions and hit regions of the baseline, respectively. In addition, let MR' and HR' be the set of miss regions and hit regions after modifying the instruction cache design, respectively. The number of regions in a set is expressed using vertical bars, such as in $|MR|$.

To estimate the reduction in executed cycles from a reduction in miss regions, we define a *penalty* for a region. This is the maximum number of cycles required to fetch instructions in the region. Let P_i denote the baseline penalty for region i, and P'_i denote the penalty after modifying the instruction cache design. Suppose a workload is instruction-cache-intensive, where modifying the instruction cache design improves the performance. In this case,

$$\text{(a reduction in executed cycles)} \simeq \sum_i P_i - \sum_i P'i \quad (1)$$

under the condition that the number of branch mispredictions does not change before or after modifying the instruction cache design. Some instruction prefetch techniques reduce the number of branch mispredictions [13]–[15]. When applying such techniques, we need to add a penalty for branch mispredictions to (1). This study considers the case in which the number of branch mispredictions does not change before or after modifying the instruction cache design. In Section V-B, we demonstrate that (1) holds for the workload we used for the evaluation.

Transforming the right-hand side of (1), we obtain the following:

$$\begin{aligned}&\sum_i P_i - \sum_i P'i \\ &= \left(\frac{\sum_{i\in MR} P_i}{|MR|}|MR| + \frac{\sum_{i\in HR} P_i}{|HR|}|HR|\right) \\ &\quad - \left(\frac{\sum_{i\in MR'} P'_i}{\left|MR'\right|}\left|MR'\right| + \frac{\sum_{i\in HR'} P'_i}{\left|HR'\right|}\left|HR'\right|\right). \quad (2)\end{aligned}$$

We assume that the mean penalties for miss and hit regions are almost unchanged before and after modifying the instruction cache design, respectively, that is,

$$\frac{\sum_{i\in MR} P_i}{|MR|} \simeq \frac{\sum_{i\in MR'} P'_i}{\left|MR'\right|}, \frac{\sum_{i\in HR} P_i}{|HR|} \simeq \frac{\sum_{i\in HR'} P'_i}{\left|HR'\right|}. \quad (3)$$

In Section V-B, we describe in detail whether (3) holds.

Because this study considers the case in which the number of branch mispredictions does not change before or after modifying the instruction cache design, the total number of regions does not change before or after the modification, that is,

$$|MR| + |HR| = \left|MR'\right| + \left|HR'\right|. \quad (4)$$

From (1), (2), (3), and (4),

$$\begin{aligned}&\text{(a reduction in executed cycles)} \\ &\simeq \left(\frac{\sum_{i\in MR} P_i}{|MR|} - \frac{\sum_{i\in HR} P_i}{|HR|}\right)\left(|MR| - \left|MR'\right|\right), \quad (5)\end{aligned}$$

where $\frac{\sum_{i\in MR} P_i}{|MR|}$, $\frac{\sum_{i\in HR} P_i}{|HR|}$, and $|MR|$ can be computed simultaneously with the baseline performance by simulating the entire processor pipeline. Therefore, $|MR'|$ is the only additional value needed to estimate a reduction in executed cycles achieved by modifying the instruction cache design.

TABLE I
PROCESSOR CONFIGURATION

Module	Parameter
Branch predictor	GEHL perceptron, 64 KiB (232-bit history)
Target predictor	128-entry L1 BTB, 32-entry RAS, 1 cycle
	4096-entry L2 BTB, 2 additional cycles
FTQ	instruction granularity, 144 entries
Front-end width	fetch width: 6, decode width: 6
L1I cache	32 KiB, 8-way, 4 cycles[a], FDP[b]
L1D cache	48 KiB, 12-way, 4 cycles[a], Next-line
L2 cache	512 KiB, 8-way, 15 cycles[a], SPP
L3 cache	2 MiB, 16-way, 45 cycles[a]
Main memory	116 cycles[a]

[a]These values are typical (minimum) load-to-use latencies.
[b]Realized through the instruction-granularity 144-entry FTQ.

Based on the above discussion, the following performance estimation flow is feasible.

1) By simulating the entire processor pipeline, $\frac{\sum_{i \in MR} P_i}{|MR|}$, $\frac{\sum_{i \in HR} P_i}{|HR|}$, $|MR|$, and the baseline performance are obtained.
2) Using a simple simulator that replicates only the behavior of the instruction cache and receives a sequence of cache accesses and branch mispredictions, $|MR'|$ is obtained.
3) A reduction in executed cycles is estimated according to (5).

In this flow, we need to simulate the entire processor pipeline only once. In our preliminary evaluation, the simulation time for the instruction cache alone was $\sim$3s on average, whereas that for the entire processor pipeline described in Section V-A was $\sim$650s on average. Hence, the above flow significantly reduces the time required to explore the design space compared to repeating the simulations of the entire processor pipeline.

V. EVALUATION

A. Methodology

We used ChampSim [16] to simulate a processor with the configuration listed in Table I. We determined the value of the parameters based on Intel Sunny Cove [17]. We made the size of the FTQ larger than the default size of ChampSim, which is 12, to simulate a front-end where instruction cache access is decoupled from branch prediction. To simulate realistic branch target prediction, we implemented the BTB and the return address stack in ChampSim.

As the workloads, we used 50 traces distributed in the 1st Instruction Prefetching Championship [18]. We used the first 50M instructions of these traces for warming up and the following 50M instructions for the evaluation. Some of the traces contain fewer than 100M instructions, and for those, we warmed up with the first 50M instructions and then ran simulations for the evaluation until reaching the end of the traces.

B. Results

We added RDIP [1] to the processor as an example of a modification in the instruction cache design. We checked that

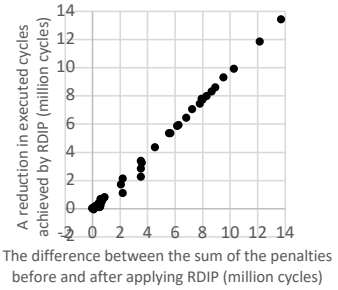
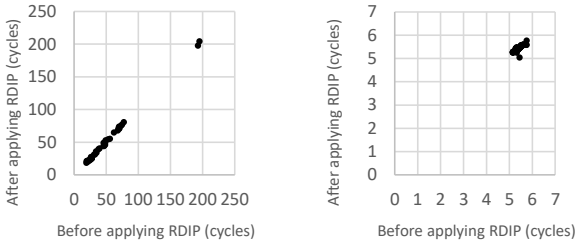


Fig. 5. Scatter plot comparing the difference between the sum of the penalties before and after applying RDIP ($\sum_i P_i - \sum_i P'_i$) and a reduction in executed cycles achieved by RDIP.



(a) Miss regions. (b) Hit regions.

Fig. 6. Scatter plot comparing the mean penalties before and after applying RDIP.

(1) and (3) held, i.e., that the workloads were instruction cache intensive and that the mean penalty for miss/hit regions barely changed before and after applying RDIP.

Fig. 5 shows the difference between the sum of the penalties before and after applying RDIP ($\sum_i P_i - \sum_i P'_i$) on the horizontal axis and a reduction in the number of executed cycles by applying RDIP on the vertical axis. Each point represents a workload, which mostly lie on a straight line with a slope of 1 passing through the origin. This result means that these workloads satisfy (1); namely, they are instruction-cache-intensive.

Fig. 6a shows the mean penalty for miss regions before applying RDIP ($\frac{\sum_{i \in MR} P_i}{|MR|}$) on the horizontal axis and the mean penalty after applying RDIP ($\frac{\sum_{i \in MR'} P'_i}{|MR'|}$) on the vertical axis. Similarly, Fig. 6b shows the mean penalties for hit regions ($\frac{\sum_{i \in R} P_i}{|HR|}$ and $\frac{\sum_{i \in HR'} P'_i}{|HR'|}$). Each point represents a workload, most of which lie on a straight line with a slope of 1 passing through the origin, satisfying (3).

Fig. 7 shows the absolute value of the error of the estimated cycle per instruction (CPI). The *proposal* represents the CPI when applying RDIP, as calculated using our proposed method, in which a reduction in executed cycles is estimated by (5). Although, as mentioned above, $|MR'|$ in (5) can be obtained by simulating only the behavior of the instruction cache, we calculated it using ChampSim, the reason for which being that the purpose of this evaluation is not to verify that simulating only the behavior of the instruction cache reduces the design time, but to verify the validity of (5). We also estimated a reduction in the executed cycles from a

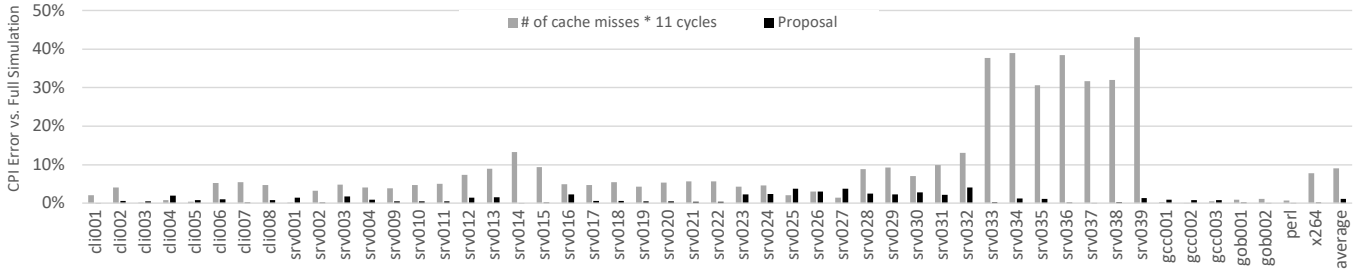


Fig. 7. Performance prediction accuracy when applying RDIP.

TABLE II
ERROR RATE OF CPI ESTIMATED BY THE TWO METHODS

	# of cache misses * 11 cycles		Proposal	
	Average	Maximum	Average	Maximum
RDIP	9.0%	43.1%	1.1%	4.1%
FNL+MMA	33.5%	166.7%	2.3%	7.3%
D-JOLT	43.6%	234.8%	5.1%	11.7%

reduction in L1I cache misses multiplied by 11 cycles, which is the difference between the L2 cache load-to-use latency (15 cycles) and the L1I cache load-to-use latency (4 cycles). We then calculated CPI when applying RDIP, which is depicted as *# of cache misses* $\times$ *11 cycles* in Fig. 7. The proposed method estimated CPI more accurately than the estimate using the reduction in L1I cache misses.

To show that our proposed method is also effective for state-of-the-art prefetchers, we conducted the same evaluation as RDIP on FNL+MMA [6] and D-JOLT [3]. Table II presents the results of this study. Although the error rate of our proposed method was larger than that when applying RDIP, the performance estimate using our proposed method is more accurate than the performance estimate using a reduction in L1I cache misses.

VI. RELATED WORK

Matsuo et al. used the notion that after an instruction cache miss occurs, the latency of the instruction cache miss can be hidden until the next branch misprediction in their method of dynamically changing the length of the instruction supply pipeline [19].

There are several performance-modeling techniques with a dependence graph [20], [21]. Although they help designers determine which components should be improved, our proposed method helps designers estimate the performance obtained by making specific modifications to the components, such as changing the cache replacement algorithm or adding prefetchers.

VII. CONCLUSION

We introduced the idea of a *miss region* as an appropriate metric for designing an instruction cache. We also proposed a method for estimating the performance when modifying the instruction cache design based on the number of miss regions. Our proposed method represents the performance of a processor applying an existing instruction prefetcher with an average error of 1.1% and a maximum error of 4.1%.

ACKNOWLEDGMENT

The authors are grateful to Reoma Matsuo, for his study, which inspired us to conduct this research.

REFERENCES

[1] A. Kolli, A. Saidi, and T. F. Wenisch, "RDIP: Return-address-stack directed instruction prefetching," in *MICRO*, 2013, pp. 260–271.
[2] S. Mirbagher Ajorpaz, E. Garza, S. Jindal, and D. A. Jiménez, "Exploring predictive replacement policies for instruction cache and branch target buffer," in *ISCA*, 2018, pp. 519–532.
[3] T. Nakamura *et al.*, "D-JOLT: Distant jolt prefetcher," in *The 1st Instruction Prefetching Championship*, 2020.
[4] A. Ros and A. Jimborean, "The entangling instruction prefetcher," in *The 1st Instruction Prefetching Championship*, 2020.
[5] G. Reinman, B. Calder, and T. Austin, "Fetch directed instruction prefetching," in *MICRO*, 1999, pp. 16–27.
[6] A. Seznec, "The FNL+MMA instruction cache prefetcher," in *The 1st Instruction Prefetching Championship*, 2020.
[7] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive instruction fetch," in *MICRO*, 2011, pp. 152–162.
[8] J. Rupley, B. Burgess, B. Grayson, and G. D. Zuraski, "Samsung M3 processor," *IEEE Micro*, vol. 39, no. 2, pp. 37–44, Feb. 2019.
[9] D. Suggs, M. Subramony, and D. Bouvier, "The AMD "Zen 2" processor," *IEEE Micro*, vol. 40, no. 2, pp. 45–52, Feb. 2020.
[10] A. Pellegrini *et al.*, "The Arm neoverse N1 platform: Building blocks for the next-gen cloud-to-edge infrastructure SoC," *IEEE Micro*, vol. 40, no. 2, pp. 53–62, Feb. 2020.
[11] A. Perais, R. Sheikh, L. Yen, M. McIlvaine, and R. D. Clancy, "Elastic instruction fetching," in *HPCA*, 2019, pp. 478–490.
[12] Y. Ishii, J. Lee, K. Nathella, and D. Sunwoo, "Rebasing instruction prefetching: An industry perspective," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 147–150, Oct. 2020.
[13] R. Kumar, C. Huang, B. Grot, and V. Nagarajan, "Boomerang: A metadata-free architecture for control flow delivery," in *HPCA*, 2017, pp. 493–504.
[14] R. Kumar, B. Grot, and V. Nagarajan, "Blasting through the front-end bottleneck with shotgun," in *ASPLOS*, 2018, pp. 30–42.
[15] A. Ansari, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Divide and conquer frontend bottleneck," in *ISCA*, 2020, pp. 65–78.
[16] "ChampSim." [Online]. Available: https://github.com/ChampSim/ChampSim
[17] D. Kanter, "Intel's Sunny Cove sits on an icy lake," *Microprocessor Report*, Feb. 2019.
[18] "The 1st instruction prefetching championship." [Online]. Available: https://research.ece.ncsu.edu/ipc/
[19] R. Matsuo, R. Shioya, and H. Ando, "Improving the instruction fetch throughput with dynamically configuring the fetch pipeline," *IEEE Comput. Archit. Lett.*, vol. 18, no. 2, pp. 170–173, Nov. 2019.
[20] B. Fields, R. Bodik, M. Hill, and C. Newburn, "Using interaction costs for microarchitectural bottleneck analysis," in *MICRO*, 2003, pp. 228–239.
[21] J. Lee, H. Jang, and J. Kim, "Rpstacks: Fast and accurate processor design space exploration using representative stall-event stacks," in *MICRO*, 2014, pp. 255–267.