T-SKID: Predicting When to Prefetch Separately from Address Prediction

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

***Abstract*—Prefetching is an important technique for reducing the number of cache misses and improving processor performance, and thus various prefetchers have been proposed. Many prefetchers are focused on issuing prefetches sufficiently earlier than demand accesses to hide miss latency. In contrast, we propose a T-SKID prefetcher, which focuses on delaying prefetching. If a prefetcher issues prefetches for demand accesses too early, the prefetched line will be evicted before it is referenced. We found that existing prefetchers often issue such too-early prefetches, and this observation offers new opportunities to improve performance. To tackle this issue, T-SKID performs timing prediction independently of address prediction. In addition to issuing prefetches sufficiently early as existing prefetchers do, T-SKID can delay the issue of prefetches until an appropriate time if necessary. We evaluated T-SKID by simulations using SPEC CPU 2017. The result shows that T-SKID achieves a 5.6% performance improvement for multi-core environment, compared to Instruction Pointer Classifier based Prefetching, which is a state-of-the-art prefetcher.**

I. Introduction

Data prefetching is a key technique for hiding memory access latency, which is a typical execution bottleneck for many applications. Various prefetching methods have been studied, ranging from those that predict simple memory access patterns [4], [10], [11], [15], such as stream and stride accesses, to those that predict complex memory access patterns [5], [12], [14], [20], [21] using a series of address deltas or combining access history and PC.

For prefetchers, it is essential not only to predict addresses but also to issue prefetches early enough. Even if the address prediction is correct, issuing a prefetch too late for a demand access does not effectively hide the latency and loses the opportunity for performance improvement. Many prefetchers focus on issuing prefetches early enough for demand access. Such prefetchers typically predict accesses as far in the future as possible, within sufficiently high confidence in the address prediction [11], [12], [15], [18], [20]. Best-Offset Prefetcher (BOP) [13] embeds timing prediction in the address prediction, and it determines a prefetch address considering whether the prefetch is in time for a demand access.

Whereas existing prefetchers focus on issuing prefetches early enough, we focus on *delaying* prefetches. If a prefetcher issues a prefetch too early for a demand access, the prefetched line is replaced before it is referenced. In such a case, appropriately delaying the issue of the prefetch can improve the performance.

A particular but frequent example of the above case is re-referencing the same line after a long enough time has passed that the line is replaced from the cache. We call this kind of re-reference a *zero-stride* pattern. For example, if we set the stride to zero in a stride prefetcher (that is, if it generates the same address as the first access), the address of the second access is correctly predicted. However, prefetching the same address simultaneously as the first access obviously is meaningless. Instead, by delaying prefetching until the line has been replaced and then re-referenced, it can enable effective prefetching and improve performance.

We found that existing prediction mechanisms often issue too-early prefetches, and this observation offers new opportunities to improve performance by delaying prefetches. Based on this observation, we propose T-SKID, a prefetcher that can delay prefetching in addition to prefetching enough early as in the existing prefetchers. T-SKID predicts the appropriate timing for prefetching independently of the address prediction. It predicts the timing using the temporal correlation of the PCs of load instructions. On a cache miss, it does not immediately issue a prefetch using a predicted address, but it delays issuing the prefetch until the predicted time when the prefetch should be issued. This allows T-SKID to issue effective prefetches in cases that are difficult to predict with existing prefetchers, (i.e., when the insertion is too early and the line is replaced before the demand access including the zero-stride patterns).

The contributions of this paper are as follows.

- We found that the existing prefetchers often issue too-early prefetches, and we show that *delaying* prefetches offers new opportunities to improve performance.
- We propose T-SKID, which predicts prefetch addresses and timing separately. We introduce a novel correlation for data prefetchers, which is based on the repeatability of the order and timing of PCs. We propose a timing prediction mechanism that leverages this correlation.
- In addition to issuing prefetches sufficiently early as the existing prefetchers do, T-SKID can delay the prefetch issue until an appropriate time if necessary. T-SKID can also issue effective prefetches for the zero-stride pattern, which is a re-reference pattern that cannot be handled by the existing prefetchers.
- We evaluated T-SKID by detailed simulation with the same configuration used in the 3rd Data Prefetching Championship (DPC3) [1]. The evaluation results show that T-SKID achieves a performance improvement of 46.0% and 26.2% for single-core and multi-core, respectively, compared to a processor without prefetching. Compared to In-

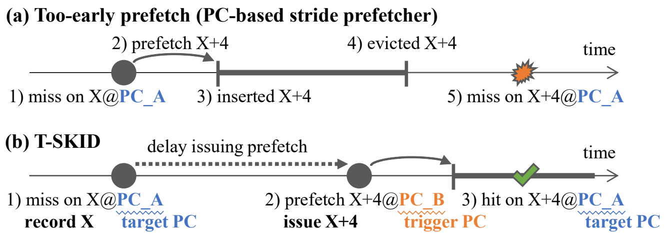

Fig. 1: Too-early prefetch and T-SKID

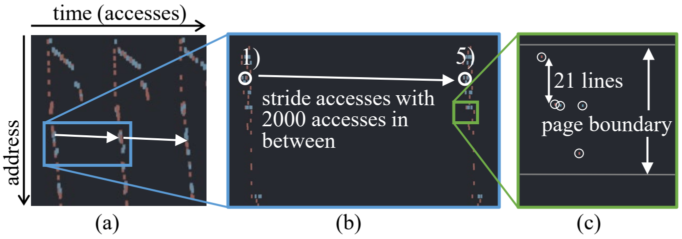

Fig. 2: Memory access pattern of 607.cactuBSSN_s-2421B. (a) Visualized access pattern. The white arrows represent stride accesses. There are thousands of stride accesses in the figure. (b) Zoomed image. The left and right access series are stride accesses. (c) More zoomed image. These accesses are very sparse.

struction Pointer Classifier based Prefetching (IPCP) [14], which is state-of-the-art and the DPC3's champion, T-SKID achieves a performance improvement of 1.5% and 5.6% for single-core and multi-core, respectively.

II. Motivation

Even if an address prediction is correct, a line inserted by issuing a prefetch too early for a demand access will be evicted before it is referenced, resulting in a miss. An example of such a miss is shown in Figure 1(a), in which a PC-based stride prefetcher is used as a prefetcher. In the figure, (1) a miss occurs at address X, (2) so the stride prefetcher issues a prefetch for address X+4 based on the stride (four) learned in advance. (3) After the prefetched line is inserted, (4) the line is evicted because too many instructions have been executed before it is referenced. (5) As a result, access to address X+4 results in a demand miss, even though it was prefetched. We refer to such misses as misses prefetched too-early.

We show a practical example of such an access pattern in Figure 2, which visualizes memory accesses in 607.cactuBSSN_s-2421B of SPEC CPU 2017 [3]. In this figure, the vertical axis represents address space, the horizontal axis represents access time, and each plotted point represents a memory access. In Figure 2, time moves from the left to the right, and the arrows in (a) represent stride accesses. In the zoomed image (b), 1) and 5) represent stride accesses that are separated in time, corresponding to 1) and 5) in Figure 1(a). These 1) and 5) are the same PC-based stride access sequence, and a PC-based stride prefetcher can easily predict their addresses. However, between 1) and 5), approximately 2000 lines are accessed, which is much larger than the size of the L1 cache (512 lines). Consequently, the prefetched line will be evicted until 5). Note that spatial correlation-based prefetchers (i.e., spatial memory streaming (SMS) [21] and Bingo [5]) cannot even predict the addresses of the lines accessed in the figure. This is because the number of addresses accessed in a page is very small, as shown in Figure 2(c), and the access pattern shown vertically in the figure is irregular.

We found that the existing prefetchers often issue too-early prefetches and that delaying prefetches offers new opportunities to improve performance significantly. Using traces generated from SPEC CPU 2017 and distributed in DPC3, we investigate how many existing prefetchers generate misses prefetched too-early in the L1 cache. In this investigation, we used multi lookahead offset prefetching (MLOP) [18] and IPCP [14], which are the state-of-the-art prefetchers and awarded in DPC3. As a result, out of the 46 traces, 7 (MLOP) or 12 (IPCP) traces had more than one misses prefetched too-early per 1K instructions, and 3 (MLOP) or 4 (IPCP) traces had more than five. In particular, the maximum number of misses prefetched too-early per 1K instructions was 20.7 with MLOP and 16.0 with IPCP. These results show that appropriately delaying prefetches offers new opportunities to improve performance.

III. T-SKID

A. Overview

As explained in Section II, the key to improving performance is to delay issuing prefetches appropriately. We propose T-SKID to realize such prefetch timing adjustment.

In addition to address prediction, T-SKID predicts prefetch timing and performs both the prediction independently. For the address prediction, T-SKID uses a PC-based stride predictor. For the timing prediction, T-SKID exploits the repeatability of the order and timing of PCs.

Figure 1(b) shows an overview of T-SKID mechanism. In this figure, addresses X and X+4 are accessed in the same way as in Figure 1(a). In Figure 1(a), the miss occurred due to the prefetch being issued too early, as mentioned earlier. In contrast, T-SKID delays issuing the prefetch until the access of the load by PC_B in (2) in Figure 1(b). As a result, the prefetched line is not evicted, and the access of PC_A in (3) results in a cache hit.

Unlike existing PC-based prefetchers, the PC of a load instruction (e.g., PC_B) that determines when to issue the prefetch can be different from a PC (e.g., PC_A) used for address prediction. The PC used for the address prediction (e.g., PC_A) is called a *target PC*. The PC of a load instruction that determines the timing of issuing (e.g., PC_B) is called a *trigger PC*. T-SKID learns trigger PCs that achieve appropriate issue timing for corresponding target PCs. This timing learning allows T-SKID to prefetch with appropriate timing.

In the following, we describe the implementation that realizes the behavior of T-SKID. First, we describe the structure of T-SKID, and then we explain the details of learning and prediction.

B. Structure

T-SKID has the following four components, as shown in Figure 3. The following two tables are used for the address and timing prediction, respectively:

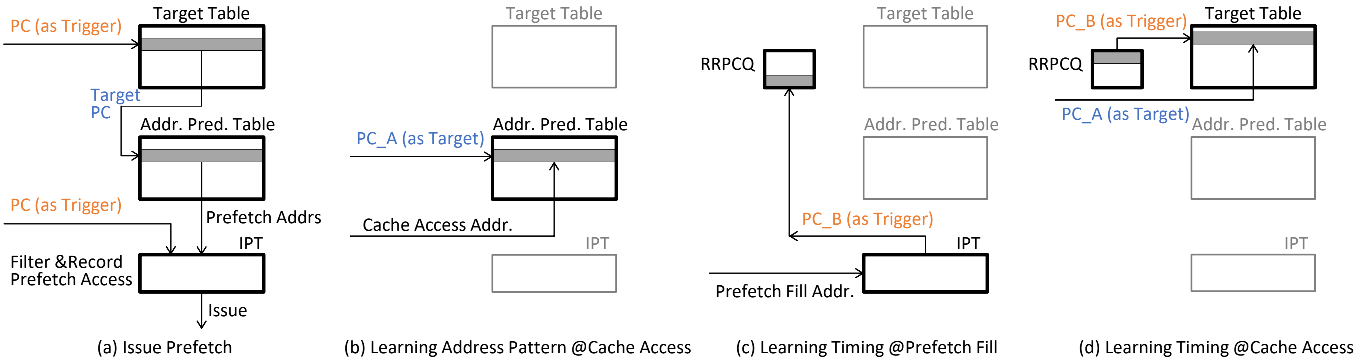

Fig. 3: Issuing Prefetch and Learning Timing/Address Patterns in T-SKID

1. Address prediction table: This table records information (e.g., `stride`, `last_addr`, and `degree`) needed for the PC-based stride prediction.
2. Target table: This table records target PCs associated with trigger PCs.

The next two components are for monitoring cache fill and learning in the target table.
3. Inflight prefetch table (IPT): This table tracks issued prefetches that have not yet been filled into the cache. Each entry of this table has a PC that triggered a prefetch and its prefetch address.
4. Recent request PC queue (RRPCQ): This queue records PCs that triggered a prefetch recently filled into the cache.

C. *Issuing prefetches*

An overview of issuing prefetch is shown in Figure 3(a). T-SKID issues prefetch in the following steps.

1) For each memory access, the PC of the access is treated as a trigger PC, and the target table is searched with the trigger PC for timing prediction. If the trigger PC hits in the target table, a target PC is obtained and then it goes to step 2 to predict addresses.
2) The address prediction table is searched with the obtained target PC. If the target PC hits in the address prediction table, T-SKID predicts addresses and issue prefetches using the obtained information. Specifically, the prefetch addresses are calculated by $\mathrm{last\_addr} + \mathrm{stride} \times \{1, 2, \ldots, \mathrm{degree}\}$.
3) For each issued prefetch, the trigger PC and the prefetch address are tied together and recorded in the IPT. This information is used for timing learning, which will be explained later.

D. *Address learning*

T-SKID performs prediction and learning based on existing PC-based stride prefetchers [10], [14]. As mentioned above, a `last_addr` and `stride` for each target PC are recorded in the address prediction table. As shown in Figure 3(b), for each memory access of an instruction with PC_A, the address prediction table is searched with PC_A, and the `last_addr` and `stride` in the corresponding entry are updated. Note that a prefetch degree is updated independently by the timing learning as described below. In addition, to handle stream access patterns by multiple PCs, the number of recent accesses within a certain range from `last_addr` is counted, and if the number exceeds a threshold, `stride` is set to the line size.

Unlike the existing PC-based stride prefetchers, T-SKID learns any stride values as a valid stride, even if it is zero. Although it does not make sense for existing prefetchers to issue a prefetch when a stride is zero, T-SKID delays issuing the prefetch and thus can issue effective prefetches in the zero-stride pattern, which is a long-term re-reference described in Section I.

E. *Timing learning*

T-SKID learns the latency required for prefetching and issues the prefetch at the appropriate time. This latency is specifically the time between issuing a prefetch and its completion. T-SKID integrates this latency into the form of the relationship between the trigger PC and the target PC, rather than a specific number of cycles. Specifically, T-SKID learns that relationship so that a prefetch triggered by the trigger PC is in time for an access by the target PC.

We describe the behavior of this timing learning below.

1) As mentioned above, when a prefetch is issued, a trigger PC and a prefetch address are tied together and recorded in the IPT.
2) Recording trigger PCs: When the prefetched line is inserted into the cache, T-SKID searches the IPT with the address of the line to obtain the corresponding trigger PC, as shown in Figure 3(c). The obtained trigger PC (PC_B) is recorded in the RRPCQ. This process makes the RRPCQ contain PCs that triggered prefetches recently filled into the cache. These recorded PCs have the following property: If these PCs triggered a prefetch, the insertion of the prefetched line into the cache would have already been completed. Thus, the PCs recorded in the RRPCQ are good candidates for trigger PCs.
3) Linking a target PC to trigger PCs: As shown in Figure 3(d), when a memory access is performed by the instruction with PC_A, all the trigger PCs in the RRPCQ are read. For example, two trigger PCs are obtained in our evaluation because the number of entries in the RRPCQ is two. Using the obtained trigger PCs as indices, PC_A is written to each corresponding entry in the target table. Through this process, a target PC is linked to trigger PCs.

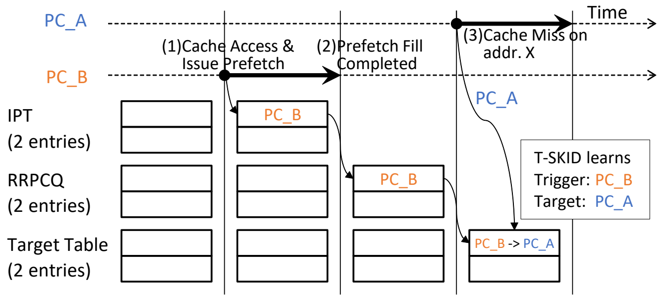

Fig. 4: Timeline of Learning Timing

In the following, we show an example of the above behavior using a timeline shown in Figure 4. The numbers (1)–(3) below correspond to the steps in (1)–(3) above, respectively.

1) When a load instruction with PC_B issues a prefetch, an IPT entry is allocated and PC_B is written to there.
2) When the prefetch fill triggered by PC_B is completed, PC_B is read from the corresponding entry in the IPT and is written in the RRPCQ.
3) When a load instruction with PC_A causes a miss on address X, PC_B is obtained from RRPCQ and PC_A is written in the target-table entry corresponding to PC_B.

As a result of this procedure, when the target table is referenced by using PC_B as a trigger PC, PC_A is obtained as a target PC. The above mechanism realizes the behavior shown in Figure 1 (b), where the prefetch for the target PC, PC_A, is appropriately delayed until an access with PC_B.

As a result of the above learning, in some cases, the trigger PC and the target PC may be the same PC. This means that accesses by the same PC form a short-interval stride-access pattern. In such a case, the above mechanism immediately issues prefetches, like the existing stride prefetchers.

To detect whether a prefetch needs to be issued even earlier, T-SKID learns the degree dynamically by using IPT. Each entry in the IPT monitors cache accesses by load instructions with a PC recorded in the entry. It counts the number of lines accessed by such instruction from issuing corresponding prefetch until completion of the prefetched-line fill. For example, if a line size is 64B and addresses 64, 128, and 192 are accessed before a prefetch by PC_A is filled, the counter is set to 3. When the fill is completed, the value of the counter is read and stored as degree in the address prediction table. This allows issuing distant prefetches when the interval between accesses is short.

IV. Evaluation

A. Methodology

We evaluated T-SKID using ChampSim [2]. ChampSim is a trace-based out-of-order CPU simulator that can simulate detailed memory systems. Table I shows the parameters used in our evaluation. Two configurations were used: single-core and multi-core. In the simulation, a 4 KiB page was used. In ChampSim, virtual addresses are randomly mapped to physical addresses. These configurations are the same as those in DPC3 [1].

TABLE I: Simulation Parameters

Core parameters	1 or 4 cores, 5.0 GHz, 192-entry ROB, 3 ALUs, 2 Loads, 1 Store
Private L1D cache	32 KiB, 8-way, 4 cycle, 2 line/cycle, 8 MSHRs, LRU
Private L2 cache	256 KiB, 8-way, 8 cycle, 1 line/cycle, 16 MSHRs, LRU
Shared L3 cache (LLC)	2 MiB/core, 16-way, 20 cycle, 1 line/cycle, 32 MSHRs/core, LRU
DRAM	4 GiB 1-channel (single core) or 8 GiB 2-channels (multi-core) 1/24 line/cycle/channel, 48 shared WQs, 48 shared RQs

1) Workload: We used SPEC CPU 2017 [3] traces as a benchmark according to the configuration of DPC3. To analyze the performance improvement by prefetching, we used 46 simpoint [19] traces with LLC MPKI of 1.0 or higher without prefetchers. All results were obtained from a simulation of 200M instructions after a warm-up of 50M instructions.

In addition to single-core simulations, we did multi-core simulations to measure how much interactions between cores affect performance when LLC and main memory are shared by multiple cores. For the multi-core simulations, we made 46 mixed-trace workloads. Each mixed-trace workload consists of four different traces chosen randomly from the traces that were used in the single-core simulations. To avoid bias in the trace selection, all the traces appear the same number of times (i.e., four times) out of 46 mixed-trace workloads.

In the multi-core simulations, we assigned one different trace to each core. First, a warm-up was performed until the last core finished executing 50M instructions. Then, simulation was performed until the last core finished executing 200M instructions. For each trace, only the first 200M-instructions simulation was considered for measuring IPC, and speedup was calculated by comparison to no-prefetch baseline. The speedup of a prefetcher is calculated as the geometric mean of the speedup of each core.

2) Evaluated prefetchers: To evaluate the performance of T-SKID, we used the following state-of-the-art prefetchers: IPCP [14], SPP with PPF [6], MLOP [18], and Bingo [5]. In DPC3, IPCP and SPP with PPF achieved the first and second scores in single-core performance, and MLOP and Bingo achieved the first and second scores in multi-core performance. In the simulation, we used the source code of these prefetchers uploaded on the DPC3 website.

We configured T-SKID that consumes 24.75 KiB storage. It consumes 12.12 KiB for a 1024-entry target table (8-way set associative), 9.50 KiB for a 1024-entry address prediction table (8-way set associative), and 3.13 KiB for other metadata including a 16-entry IPT and a 2-entry RRPCQ. T-SKID was connected only to the L1D cache, and no other prefetchers were connected to the other caches.

B. Result

1) Performance improvement: Figure 5 shows the single-core and multi-core performance improvement of the evaluated models over the no-prefetching baseline. In this evaluation, we also evaluated the following prefetchers that participated in DPC3: Sangam [7], Berti [17], and Pangloss [16]. T-SKID showed the highest performance among other existing prefetchers both in the single-core and multi-core configurations.

Some prefetchers achieved performance improvements close to that of T-SKID only in the single-core or multi-core configuration, but not in both. In the multi-core configuration, T-SKID achieved a 5.6% speedup over IPCP, which had the

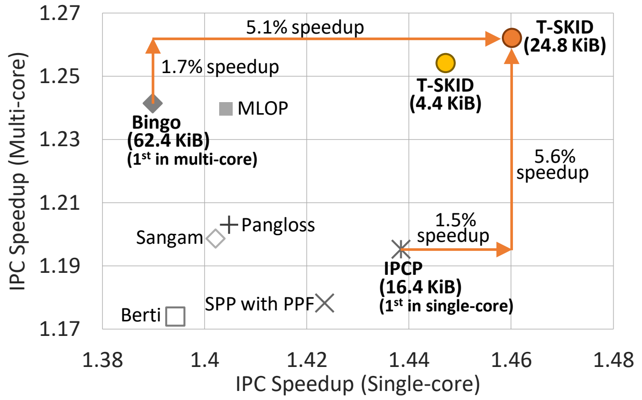

Fig. 5: IPC speedup of T-SKID in single/multi configurations

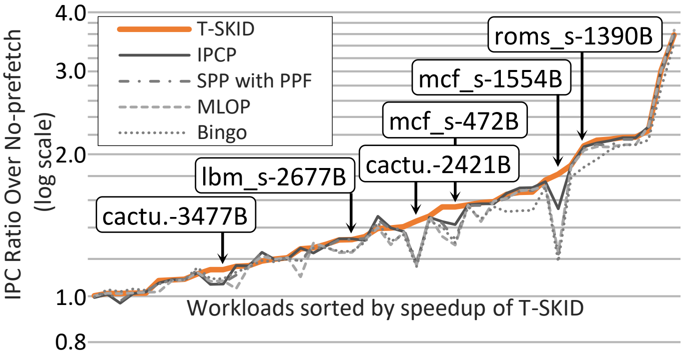

Fig. 6: IPC speedup in single-core configuration

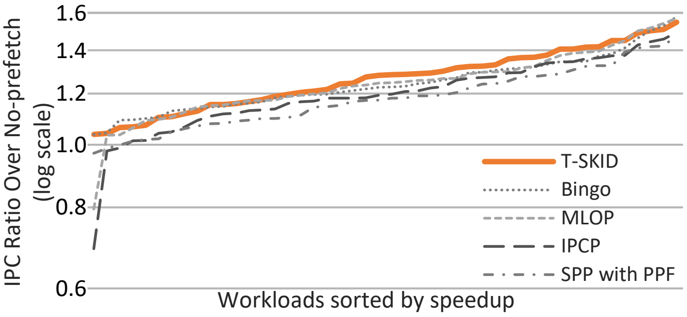

Fig. 7: IPC speedup in multi-core configuration

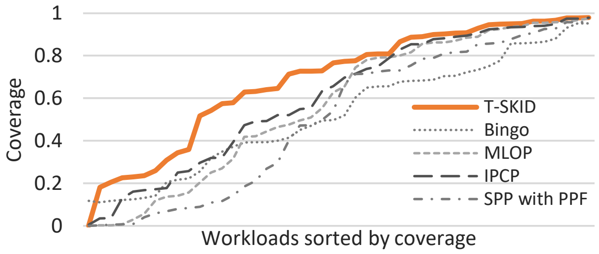

Fig. 8: Coverage in single-core configuration

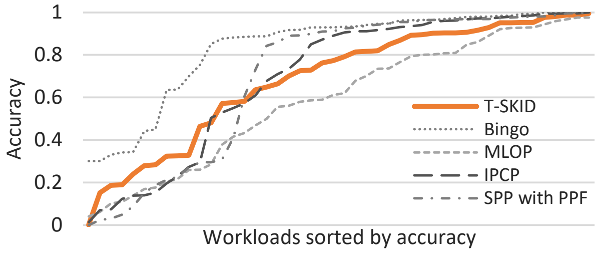

Fig. 9: Accuracy in single-core configuration

highest single-core performance among existing prefetchers. Even if the storage budget of T-SKID is 4.4 KiB, T-SKID still achieved a 4.9% speedup over IPCP. In single-core configuration, T-SKID achieved a 5.1% speedup over Bingo, which had the highest multi-core performance among existing prefetchers. These results show that T-SKID is adaptable to a variety of environments.

2) *Single-core performance analysis:* Figure 6 shows the IPC speedup of the evaluated prefetchers over a no-prefetching baseline in the single-core configuration. The workloads have been sorted in increasing order of T-SKID's speedup. T-SKID showed the highest performance improvement or almost the highest performance improvement in almost all traces. In particular, in 607.cactuBSSN_s and 605.mcf_s, T-SKID achieved a significant performance improvement over existing prefetchers because it can delay issuing prefetchers until the appropriate timing. In 607.cactuBSSN_s-2421B, T-SKID achieved a 44.4% improvement, which is a 23.4% speedup over IPCP.

3) *Multi-core performance analysis:* Figure 7 shows the speedup of the evaluated prefetchers over the no-prefetching baseline. The workloads have been sorted in increasing order of the speedup. The left end of Figure 7 shows that the performance of IPCP and MLOP was significantly lower than the no-prefetching baseline. The performance of SPP with PPF was also lower than that of the no-prefetching baseline. On the other hand, T-SKID improved the performance by 3.8% for this workload. Additionally, T-SKID greatly improved the performance compared to the other prefetchers near the center of the figure. These results show that T-SKID's timing learning is robust and can adapt to many environments. When multiple programs are running at the same time, the interaction between them can significantly change the prefetch latency from that of running a single program. T-SKID's timing learning method covers a wide range of such cases.

4) *Prefetching coverage and accuracy:* Figures 8 and 9 show the coverage and accuracy of the evaluated prefetchers, respectively. The workloads have been sorted in increasing order of the accuracy or the coverage. This result shows that T-SKID greatly improved the coverage without much reduction in the accuracy. The high coverage of T-SKID is largely due to the timing learning. The left sides of Figures 8 and 9 show that T-SKID simultaneously improved coverage and accuracy for memory-access patterns that are difficult to prefetch with existing prefetchers.

5) *Storage sensitivity:* Figure 10 shows the change of single-core IPC speedup varying the storage budgets for the evaluated prefetchers. T-SKID showed the highest performance improvement in every range of capacities shown in the figure.

T-SKID and existing PC-based prefetchers, including IPCP, both perform PC-based stride prediction; thus they can learn many stride patterns as storage capacity is increased so that many PCs can be learned. However, as shown in Figure 10, the performance improvement of IPCP is quickly saturated when the storage capacity is increased. This is because the number of

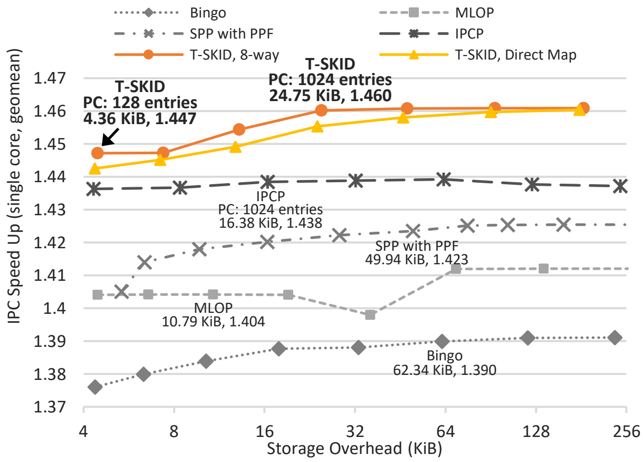

Fig. 10: Storage Sensitivity. The storage consumption of each model used in the other evaluations is also shown.

PCs that cause the normal stride access pattern is not large. On the contrary, T-SKID can handle the zero-stride access patterns in addition to the normal stride access patterns. T-SKID can further improve the performance by using budgets to learn a new class of access patterns, the zero-stride patterns.

We also evaluated T-SKID with a simple direct-mapped table configuration as well as the 8-way set-associative configuration used in the previous evaluations. T-SKID with the direct-mapped tables still showed the highest performance improvement in every range of capacities, even though it has a slight reduction in performance improvement due to conflict.

V. Related Work

Various prefetching methods have been proposed. They are focused on issuing prefetches early enough and cannot delay issuing as follows. (1) To determine how early to issue prefetches, stream-based prefetchers typically have two parameters, degree and distance. The degree represents how many consecutive lines are prefetched at a time, while the distance represents how far ahead of the current line is prefetched. These parameters are statically determined in IPCP [14] and several works [11], [15], [23]. Some other works dynamically change these parameters to avoid issuing useless prefetches [8], [9], [22]. (2) VLDP [20], SPP [12], and an IP complex stride prefetcher included in IPCP focus on delta between accessed addresses. By recursively predicting delta sequences, they prefetch several accesses ahead and thus improve the possibility of issuing prefetches in time for demand accesses. (3) BOP [13] learns the single best offset to prevent issuing prefetches too late. MLOP [18] determines multiple offsets considering the order of accesses, and it attempts to issue more valuable prefetches than BOP.

VI. Conclusion

We found that existing prefetchers often issue too-early prefetches, and this observation provides new opportunities to improve performance. To tackle the issue of the too-early prefetch, we proposed T-SKID, which predicts prefetch addresses and timing separately and delays issuing prefetch accesses until appropriate times. We evaluated T-SKID through simulations with SPEC CPU 2017. As a result, T-SKID achieved a 46.0% performance improvement for single-core and 26.2% performance improvement for multi-core compared to a processor without prefetching.

Acknowledgements

This work was partially supported by JSPS KAKENHI Grant Numbers JP19H04077, JP20H04153, JP20J22752, JP21J11687.

References

[1] "The 3rd data prefetching championship," https://dpc3.compas.cs.stonybrook.edu/.

[2] "ChampSim," https://github.com/ChampSim/ChampSim/.

[3] "Standard performance evaluation corporation CPU2017 benchmark suite," http://www.spec.org/cpu2017/.

[4] J. Baer and T. Chen, "An effective on-chip preloading scheme to reduce data access penalty," in *ACM/IEEE Int. Conf. on Supercomputing (SC)*, 1991, pp. 176–186.

[5] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Accurately and maximally prefetching spatial data access patterns with Bingo," in *The 3rd Data Prefetching Championship*, 2019.

[6] E. Bhatia, G. Chacon, E. Teran, D. A. Jiménez, and P. Gratz, "Enhancing signature path prefetching with perceptron prefetch filtering," in *The 3rd Data Prefetching Championship*, 2019.

[7] M. Chaudhuri and N. Deshmukh, "Sangam: A multi-component core cache prefetcher," in *The 3rd Data Prefetching Championship*, 2019.

[8] E. Ebrahimi, O. Mutlu, and Y. N. Patt, "Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems," in *IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, 2009, pp. 7–17.

[9] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated control of multiple prefetchers in multi-core systems," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2009, pp. 316–326.

[10] J. W. C. Fu, J. H. Patel, and B. L. Janssens, "Stride directed prefetching in scalar processors," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 1992, pp. 102–110.

[11] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, 1990, pp. 364–373.

[12] J. Kim *et al.*, "Path confidence based lookahead prefetching," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[13] P. Michaud, "Best-offset hardware prefetching," in *IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, 2016, pp. 469–480.

[14] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier based hardware prefetching," in *The 3rd Data Prefetching Championship*, 2019.

[15] S. Palacharla and R. E. Kessler, "Evaluating stream buffers as a secondary cache replacement," in *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, 1994, pp. 24–33.

[16] P. Papaphilippou, P. H. J. Kelly, and W. Luk, "Pangloss: a novel Markov chain prefetcher," in *The 3rd Data Prefetching Championship*, 2019.

[17] A. Ros, "Berti: A per-page best-request-time delta prefetcher," in *The 3rd Data Prefetching Championship*, 2019.

[18] M. Shakerinava, M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Multi-lookahead offset prefetching," in *The 3rd Data Prefetching Championship*, 2019.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 45–57, 2002.

[20] M. Shevgoor *et al.*, "Efficiently prefetching complex address patterns," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2015, pp. 141–152.

[21] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, 2006, pp. 252–263.

[22] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, 2007, pp. 63–74.

[23] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–25, 2002.