T-SKID: Timing Skid Prefetcher

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

***Abstract*—Prefetching is essential to reduce the number of cache misses and improve processor performance. Many prefetchers have been proposed, including simple but highly effective stream-based prefetchers and prefetchers that predict complex access patterns based on structures such as history buffers and bit vectors. However, many cache misses still occur in many applications. We found that many applications have simple address sequences (e.g. stride accesses) that existing prefetchers often cannot predict due to the long time intervals between their accesses. Moreover, in these address sequences, even if cache lines are correctly prefetched, they are often evicted before demand accesses because of its long interval. In this paper, we propose a timing skid (T-SKID) prefetcher, which independently learns the next addresses and the access timing. We evaluated T-SKID with SPEC CPU 2017 benchmarks according to the rule of DPC3 and the evaluation results show a more than 40% improvement in performance compared to a processor without prefetching.**

I. Introduction

Memory access latency is a major bottleneck in program execution, and cache misses cause significant performance degradation in computer systems. Prefetching is one of the most essential techniques to reduce the number of cache misses and improve processor performance. As a result, many prefetchers have been proposed, including simple stream/stride-based prefetchers ([1], [2]) and prefetchers that predict complex access patterns using multiple delta history ([3], [4]) or using bit vectors recording access patterns ([5], [6], [7]). However, many cache misses still occur in many applications.

To address this issue, we first analyzed cache access patterns that are difficult to predict using existing prefetchers. We focus on L1D cache access patterns because 1) L1D cache hit rates have a significant impact on performance and 2) an L1D prefetcher can make an accurate prediction using all addresses that cannot be obtained in an L2 cache or LLC.

We found that many applications have simple address sequences (e.g. stride accesses) that existing prefetchers often cannot predict due to the long time intervals between their accesses. For example, Figure 1 visualizes memory accesses in `607.cactuBSSN_s-2421B` of SPEC CPU 2017 [8]. In this figure, the vertical axis represents access time, horizontal axis represents address space, and each plotted point represents a memory access.

On the right side of Figure 1 (a), accesses within the same PC represented in the rectangle are stride accesses, which can be easily predicted by simple prefetchers. However, even if a cache line is successfully prefetched by simple stride prediction, thrashing access will evict the prefetched line before a demand access is issued because the capacity of the L1D cache is very small.

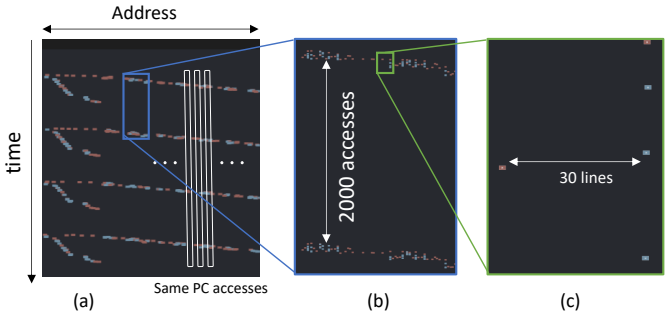


Fig. 1: A part of memory access pattern of `607.cactuBSSN_s-2421B`. The horizontal axis is the physical address and the vertical axis is time.

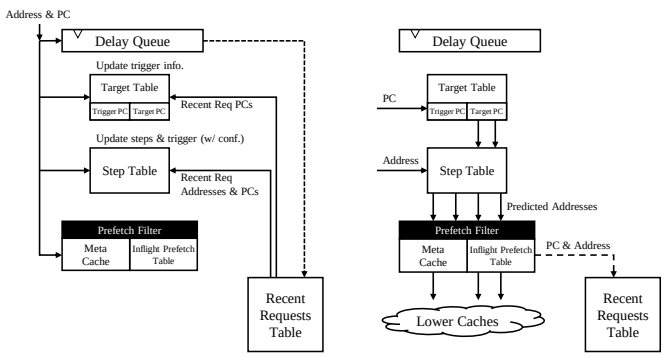
Based on the above observation, issuing prefetches at an appropriate time is important to keep the inserted line from being evicted. In typical existing prefetchers, both address prediction and prefetching access are performed simultaneously on cache misses, and prefetch timeliness is controlled by parameters such as *prefetch distance*, which represents the address distance from a trigger access. Although timing control based on the distance can issue prefetch in time for demand accesses, it cannot delay prefetching for prevention of the eviction of prefetched cache lines.

We propose a timing skid (T-SKID) prefetcher, which independently learns address patterns and appropriate prefetch timing. For learning access patterns, T-SKID is based on a PC, which has strong correlation of memory access patterns even in different address zones. T-SKID is based on a structure called Recent Requests Table (RRT), which is derived from one used in Best Offset Prefetcher (BOP) [9], to learn access timing and appropriate distance.

II. Design

Figure 2 shows the block diagram and the behavior of our proposed T-SKID. T-SKID comprises the following modules:

- The Target Table links *trigger PCs* and *target PCs*. In T-SKID, cache accesses with trigger PCs trigger prefetches whose addresses are predicted using target PCs.
- The Step Table is a table that records *steps* and *last accessed address*for each PC. In this paper, a *step* represents the delta between arbitrary two access addresses issued from the same PC. The difference between a step and a simple stride is that the step contains multiples of a stride.



(a) Learning (b) Prediction

Fig. 2: Block Diagram and Behavior of T-SKID.

- The Delay Queue is a FIFO to delay the insertion of addresses and PCs into RRT. This mechanism simulates a memory access latency.
- RRT is a table that records recent access addresses and their PCs. In RRT, a new pair comprising an address and a PC is inserted on *prefetch fill* or on dequeuing from the Delay Queue.
- The Inflight Prefetch Table is a table that holds prefetch access addresses and their PCs until prefetch fill. On prefetch fill, a fill address is notified from a memory system but its trigger PC and memory address of a trigger access are not notified. The Inflight Prefetch Table is used for tracking such additional information for each prefetch.
- Meta Cache duplicates prefetch bits and cache tags in a cache and simulates cache behavior for filtering unnecessary prefetch accesses.

A. Behavior of T-SKID

1-a) Learning access patterns: We describe how T-SKID learns access patterns. On a cache access, T-SKID looks up the Step Table. If it hits, which means there were accesses with the same PC, T-SKID calculates steps. Specifically, T-SKID determines steps by calculating the difference between the address of the cache access and recently accessed addresses, depending on the following cases:

1) If the RRT does not contain accesses with the same PC of the access: In this case, there is no access with the same PC in the near past. T-SKID looks up the Step Table with the PC and calculates a step, using a last address from the table. Based on a calculated step, T-SKID updates a step in the Step Table. Unlike conventional prefetchers, T-SKID can learn a *zero-step* pattern, which means "step = 0." Zero-step prefetching, that is, prefetching the address that just missed, does not make sense in the conventional prefetchers, but T-SKID can delay prefetch timing and long-term re-referencing access can be prefetched at appropriate timing.
2) If the RRT contains accesses with the same PC of the access: In this case, there are accesses with the same PC in the near past. Steps are calculated using addresses in RRT instead of those in the Step Table. The step in this case corresponds to a combination of *delta* and *distance* in existing prefetchers.

1-b) Learning appropriate prefetch timing: Next, we describe a method to learn appropriate prefetch timing in T-SKID. On a cache access (we call this access α), T-SKID looks up the Step Table as previously described. If it hits or a new entry is allocated, which means any accesses with the PC of α have ever been missed, this PC is regarded as a target PC and is written to Target Table entries corresponding to all PCs in RRT. The PCs in RRT can be used for trigger PCs for α, because these PCs were inserted into RRT on cache fill. That is, when prefetches for α are triggered by these PCs, it is possible to insert the prefetched lines before α occurs.

2) Prediction: Figure 2b shows how T-SKID predicts addresses and issues prefetches. First, on a cache access, target PCs linked to the PC of this access are searched in the Target Table. The obtained target PCs are the PCs of accesses that should be triggered by this access. Second, T-SKID looks up the Step Table using obtained target PCs and obtains steps and the last accessed addresses. Finally, addresses are calculated using the obtained steps and are sent to the Prefetch Filter. If the same address is in neither the Inflight Prefetch Table nor the Meta Cache, the address is prefetched. By using the Prefetch Filter, it is able to avoid to waste Prefetch Queue entries and MSHR entries.

B. Compliance with the contest rules

Not accessing any information other than the permitted: To avoid prefetching addresses existing in the L1D cache and wasting Prefetch Queue entries, it is necessary to know whether a predicted address has been prefetched or exists in the L1D cache but is not allowed to access the L1D cache to know this, according to the rule of DPC3. To follow this rule, we implemented the Inflight Prefetch Table and Meta Cache in T-SKID and included them in the storage budget. These data structures consist of information obtained from the values in the function arguments of the `l1d_cache_fill` function. We do not use the `get_set` and `get_way` functions.

III. Evaluation

1) Storage: T-SKID requires 53.78KB of storage. Table I lists the storage breakdown. We used Signature Path Prefetching (SPP) for the L2 cache in DPC3's Competition. In the simulation, SPP used `spp_dev.l2c_pref` implemented in Champsim [10]. Thus, SPP storage is also shown in Table II. The total storage of T-SKID and SPP is 59.29KB per core, which satisfies the 64KB storage limit for the DPC3.

2) Configuration: We evaluated T-SKID according to the DPC3 1-core configuration with all SPEC CPU 2017 traces with an LLC MPKI of at least 1.0, without any prefetching. All simulation results are warmed up with 50M instructions and simulated for additional 200M instructions. The parameters of caches used in the simulation are shown in Table III.

TABLE I: T-SKID Storage Computation

Structure	Components				Number of Bits		Storage	
Step Table	32 Sets, 16-way	target PC tag			11	= 512 × 11	5632	227328
		last address			48	= 512 × 48	24576	
		step group	4 Entries	step	13	= 512 × 4 × 13	26624	
				confidence	3	= 512 × 4 × 3	6144	
		confidence denominator			5	= 512 × 5	2560	
		trigger PC group	16 Entries	trigger PC	16	= 512 × 16× 16	131072	
				confidence	3	= 512 × 16 × 3	24576	
		confidence denominator			7	= 512 × 7	3584	
		LRU order			4	= 512 × 4	2048	
		valid			1	= 512 × 1	512	
Target Table	32 Sets, 16-way	trigger PC tag			11	= 512 × 11	5632	172032
		16 Entries		target PC	16	= 512 × 16 × 16	131072	
				LRU order	4	= 512 × 16 × 4	32768	
		LRU order			4	= 512 × 4	2048	
		valid			1	= 512 × 1	512	
Recent Requests Table	16 Entries	trigger PC			16	= 16 × 16	256	1088
		trigger address			48	= 16 × 48	768	
		LRU order			4	= 16 × 4	64	
Inflight Prefetch	16 Entries	prefetch line address			42	= 16 × 42	672	1712
		trigger PC			16	= 16 × 16	256	
		trigger address			48	= 16 × 48	768	
		valid			1	= 16 × 1	16	
Meta Cache	64 Sets, 8-way	tag			36	= 512 × 36	18432	18944
		prefetch bit			1	= 512 × 1	512	
Delay Queue	128 Entries	trigger PC			16	= 128 × 16	2048	10255
		trigger address			48	= 128 × 48	6144	
		enqueue time			16	= 128 × 16	2048	
	head				7	= 7	7	
	tail				7	= 7	7	
	full				1	= 1	1	
Total							430271 bits = 52.52 KiB	

TABLE II: SPP Storage Computation

Structure	Components			Number of Bits		Storage	
Signature Table	256 Entries	valid		1	= 256 × 1	256	11008
		tag		16	= 256 × 16	4096	
		signature		6	= 256 × 6	1536	
		last offset		12	= 256 × 12	3072	
		LRU order		8	= 256 × 8	2048	
Pattern Table	512 Entries	4 Entries	delta	7	= 512 × 4 × 7	14336	24576
			confidence	4	= 512 × 4 × 4	8192	
		signature confidence		4	= 512 × 4	2048	
Prefetch Filter	1024 Entries	valid		1	= 1024 × 1	1024	8192
		tag		6	= 1024 × 6	6144	
		useful		1	= 1024 × 1	1024	
Global History Buffer	8 Entries	valid		1	= 8 × 1	8	264
		signature		12	= 8 × 12	96	
		confidence		7	= 8 × 7	56	
		page offset		6	= 8 × 6	48	
		delta		7	= 8 × 7	56	
Accuracy Counter	prefetch_useful			10	= 10	10	20
	prefetch_issued			10	= 10	10	
Total						44060 bits = 5.38 KiB	

3) Performance: Figure 3 shows the IPC speedup of T-SKID (without SPP at the L2 cache in this section) and BOP/SPP, which are the state of the art prefetchers. This result shows T-SKID has significantly improved performance over existing prefetchers, especially in `607.cactuBSSN_s` traces, `605.mcf_s` traces, and `602.gcc_s-2226B`. In `607.cactuBSSN_s` traces, it is important to control prefetch timing, as described in the introduction. This result shows that prefetching the cache line at an appropriate timing for the L1D cache can significantly improve the performance.

TABLE III: cache parameter

	capacity	associativity	hit latency	bandwidth
L1D Cache	32KiB	8-way	4 cycle	2 Line/cycle
L2 Cache	256KiB	8-way	16 cycle	1 Line/cycle
L3 Cache	2MiB/Core	16-way	44 cycle	1 Line/cycle
Main Memory	4 GiB		$>$ 96 cycle	$1/24$ Line/cycle

4) Miss Coverage and Overpredictions: To evaluate the effectiveness of T-SKID, Figure 4 shows the coverage and overprediction. This result shows that T-SKID improves coverage in many traces. This is because T-SKID can predict not only timing control but also close access patterns. In addition, T-SKID improves the accuracy and coverage in `607.cactuBSSN_s` where timing control is important. T-SKID shows a high percentage of useless access in some traces, but this does not cause a serious problem, because T-SKID does not have much useless access to the main memory as shown in Figure 5. Even in a multi-core environment, T-SKID does not disturb other workloads by overprediction.

IV. Conclusion

In this paper, we analyzed access patterns that were difficult to predict with conventional prefetchers, and we revealed that many applications show access patterns with simple address sequences while time intervals between their accesses are long. We proposed T-SKID, which learns address patterns and timing to trigger prefetch accesses independently. T-SKID predicts address steps for each PC, while it learns prefetch trigger timing. We evaluated T-SKID with SPEC CPU 2017 benchmarks according to the rule of DPC3 and the evaluation results show more than 40% performance improvement compared to a processor without prefetching.

V. Acknowledgement

This work was supported by JSPS KAKENHI Grant Number 16H05855 and 19H04077.

References

[1] A. J. Smith, "Sequential Program Prefetching in Memory Hierarchies," *Computer*, vol. 11, no. 12, pp. 7–21, 1978.

[2] J. W. C. Fu, J. H. Patel, and B. L. Janssens, "Stride Directed Prefetching in Scalar Processors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 1992, pp. 102–110.

[3] J. Kim, S. H. Pugsley, P. V. Gratz, A. Reddy, C. Wilkerson, and Z. Chishti, "Path Confidence Based Lookahead Prefetching," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016, pp. 60:1–60:12.

[4] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015, pp. 141–152.

[5] Y. Ishii, M. Inaba, and K. Hiraki, "Access Map Pattern Matching for High Performance Data Cache Prefetch," *Journal of Instruction-Level Parallelism*, vol. 13, pp. 1–24, 2011.

[6] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal Memory Streaming," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2009, pp. 69–80.

[7] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2019, pp. 399–411.

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

[9] P. Michaud, "Best-offset Hardware Prefetching," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2016, pp. 469–480.

[10] "Champsim," https://github.com/ChampSim/ChampSim/

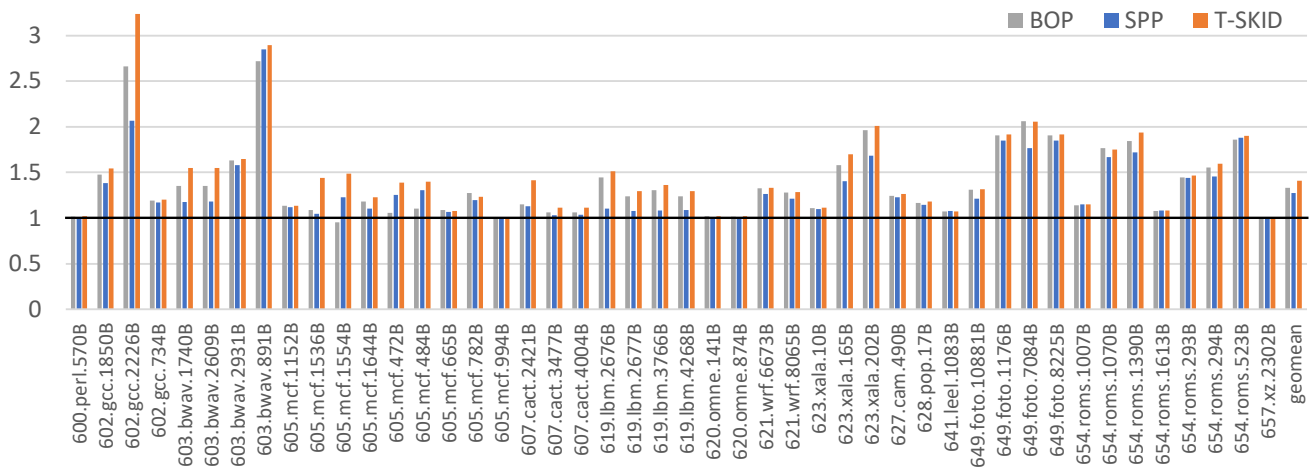


Fig. 3: IPC speedup versus no prefetching.

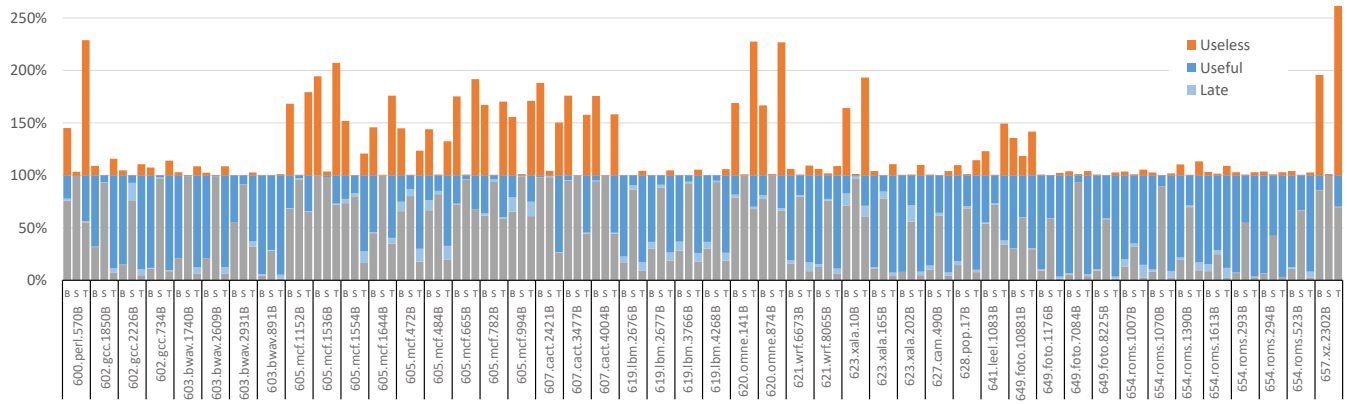


Fig. 4: Coverage and overpredictions of T-SKID as compared to the existing prefetchers. “B” represents Best-Offset Prefetcher. “S” represents SPP. “T” represents T-SKID.

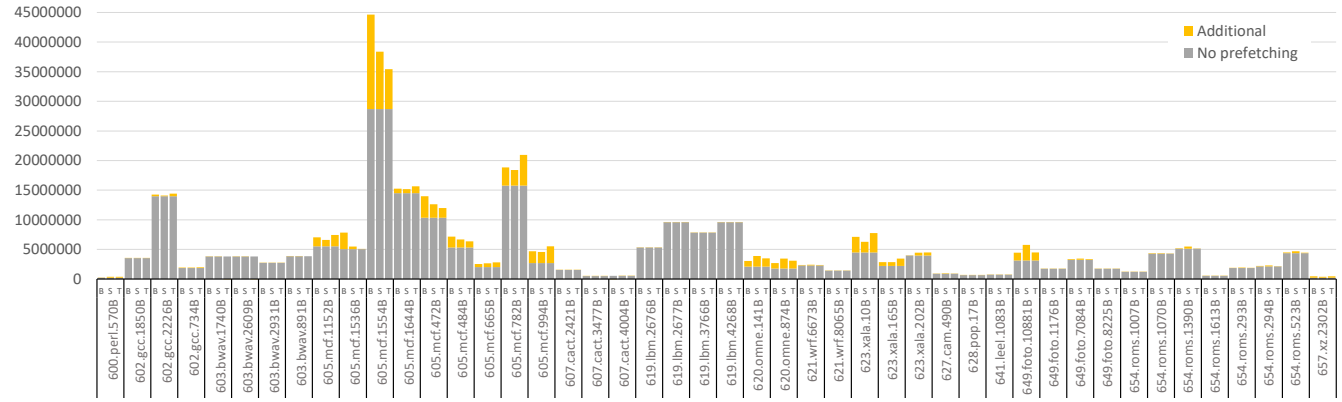


Fig. 5: `LLC_TOTAL_MISS` of T-SKID as compared to the existing prefetchers. “B” represents Best-Offset Prefetcher. “S” represents SPP. “T” represents T-SKID.