D-JOLT: Distant Jolt 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

Instruction cache misses are performance bottlenecks in recent applications, and many prefetchers have been proposed to hide the memory access latency. In this study, we focused on a prefetcher that uses the history of function calls and returns. We found that the prefetcher has various characteristics related to the association of history and miss addresses, the length of the history, and the distance to the prefetch target. Based on this observation, we propose D-JOLT, which is a prefetcher consisting of multiple prefetchers with different characteristics. D-JOLT consists of long-range prefetcher, which predicts a distant future with higher coverage, short range prefetcher, and fallback prefetcher, which predicts a near future with higher accuracy. We evaluated D-JOLT with distributed traces according to the rule of IPC1, and the results show an improvement of 28.9% in performance compared to a processor without instruction prefetching.

I. Introduction

In recent years, instruction-working sets have continued to grow in server applications and client applications, and instruction cache misses have become a performance bottleneck. Instruction prefetchers can effectively hide a cache miss latency due to such instruction cache miss. To address this issue, many prefetchers such as methods for learning stream access [3], [4] and methods using branch target buffer [6]–[8] have been recently proposed.

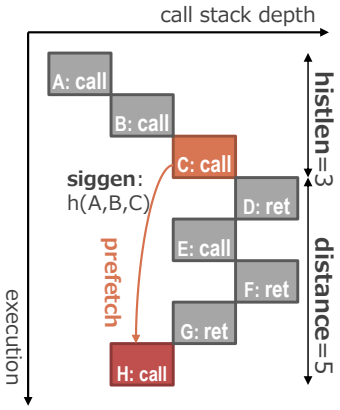


Fig. 1: Each block represents a dynamic instruction sequence with a call/return instruction at the end. In this figure, when fetching the call instruction of C, prefetch addresses are obtained using h(A,B,C) and they are prefetched for a block with H.

In this study, we focus on one of such prefetcher, the return-address-stack directed instruction prefetching (RDIP) [5]. A return address stack (RAS) is used to predict a return address from a function call. The RDIP uses a smaller RAS than that used in a branch predictor. It generates a signature from the addresses recorded in the RAS, and associates cache misses with the signature. The RDIP assumes that the same cache misses will be reproduced, if the signature has the same value as at the time of recording and prefetches those miss addresses.

We investigate characteristics of the RDIP using the following three parameters shown in Fig. 1.

1) **Siggen**: A *siggen* represents an algorithm used to generate a signature. The siggen used in the RDIP (hereinafter, referred to as RasWhole) generates signatures using all the addresses recorded in the RAS for each signature generation. By using the RAS, RasWhole can predict addresses taking account in the distant past with small cost.
2) **Histlen**: A *histlen* is the number of addresses used to generate a signature. In the RDIP, the histlen is a variable, and the maximum histlen is the number of the RAS entries. In general, making a histlen longer improves the accuracy when the miss address varies depending on the distant past execution path, while it decreases the capacity efficiency when the miss address is constant, regardless of the past execution path.

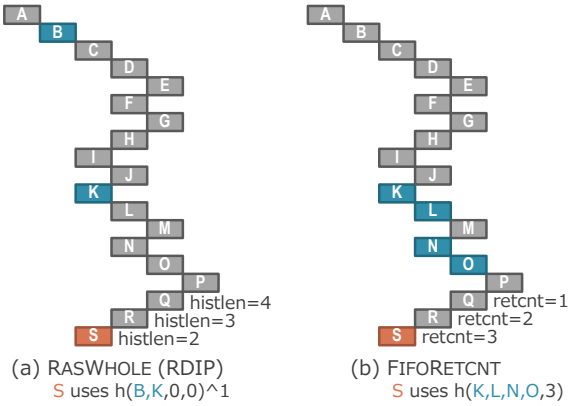


Fig. 2: (a) In this figure, the size of RAS is four. In S, RasWhole generates a signature using only B and K, because N and O are popped from the RAS and the histlen is decreased to two. (b) In this figure, the histlen is four. In Q, R, and S, FifoRetcnt generates different signatures while keeping use of K, L, N, and O.

3) **Distance**: A *distance* indicates how much time has elapsed since the signature associated with the miss address was generated. When the distance is increased, prefetches can be issued earlier than demand accesses, but the prediction accuracy will decrease because the prefetcher needs to predict a distant future. In the RDIP, the distance is two.

For the parameters described above, we observed the following new points.

1) Siggen: a) With the RASWHOLE method, the number of valid entries in the RAS decreases when the return instruction is fetched, as shown in Fig.2 (a). This leads to a reduction in signature variations, which reduces the prediction accuracy. b) With the RASWHOLE method, when the returns are fetched more than once in succession, the information about the function call immediately before is removed from the RAS, and thus, the correlation of the latest history cannot be extracted.
2) Histlen: Increasing the histlen does not lead to an explosive increase in the variation of signatures, while it leads to a super-linear increase in the case of branch direction predictor.
3) Distance: Increasing the distance provides better coverage because the number of signatures that reach a certain prefetch target increases, as shown in Fig. 3. On the other hand, the prediction accuracy decreases with increasing the distance, as described before.

Based on the observation described above, we propose the D-JOLT prefetcher. The D-JOLT consists of a novel siggen, FIFORETCNT, and a hybrid configuration that takes into account the characteristics of histlen and distance.

FIFORETCNT generates signatures using a first-in, first-out (FIFO) that records a called address in every function call, and a counter that counts the number of successive returns, as shown in Fig. 2 (b). By using a FIFO, the correlations of the last function calls can be used. In addition, it is possible to change the signature without discarding the FIFO information in a return.

The D-JOLT takes a hybrid configuration of the following three prefetchers. Each of these three prefetchers intends to prefetch the same target, and one prefetcher covers the target where other prefetcher fails to predict.

1) Long-range prefetcher: This prefetcher has a long histlen and distance. By using long distance, it achieves high coverage and timely prefetch. The reduction in the prediction accuracy is compensated for by the longer histlen.
2) Short-range prefetcher: This prefetcher has a short histlen and distance. Even when the long-range prefetcher fails to predict the addresses, the short-range prefetcher prefetches using recent history.
3) Fallback prefetcher: This is a modest stream prefetcher. When the former two prefetchers fail in address prediction and cache misses occur, the fallback prefetcher prefetches later.

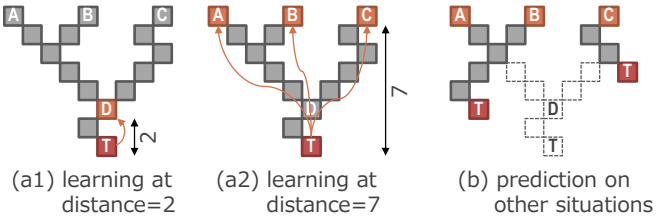


Fig. 3: (a1) When distance=2, missed addresses in T are recorded only associated with a signature at D. (a2) When distance=7, those in T are recorded associated with signatures at A, B, and C. (b) There are other paths from A, B, and C to reach T without going through D (e.g., "`if(..){D();}T();`"). In these cases, distance=2 cannot issue prefetches for T, but distance=7 can issue them, and thus, the coverage is improved.

The D-JOLT is not a TAGE-like predictor. In D-JOLT, the prefetchers with each different histlen runs in parallel, and does not form a cascade connection. Conversely, in TAGE the cascade of prediction tables with different histlens is aimed at capacity efficiency and/or rapid learning. In D-JOLT, the long-range prefetcher increases the coverage and the more accurate prefetcher that predicts the nearest future assists the long-range prefetcher.

We evaluated D-JOLT using the traces provided on the IPC1 web-site [2], and demonstrated that D-JOLT achieved 28.9% IPC speedup compared to no instruction prefetching.

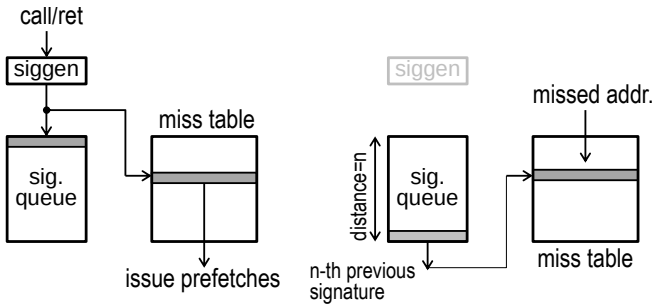
II. DESIGN

A. Structure

The long-range prefetcher and the short-range prefetcher in D-JOLT have the same structure. These prefetchers mainly consist of tables to record cache misses and components to generate signatures.

The tables for recording cache misses consist of the following three kinds of components.

1) Miss table: This table records the association between a signature and the line address of a cache miss. The table is a set-associative structure accessed by the signatures. The addresses of nearby lines are recorded together: the miss bit vectors containing the first address and the bits corresponding to each line are recorded. To improve the capacity efficiency, the upper bits of the line address are recorded in the upper bit table, described below, and the indices are recorded in the miss table. The D-JOLT has two miss table, one each for the long-range prefetcher and the short-range prefetcher.
2) Upper bit table: This table records only the upper bits of the miss line address to be recorded in miss table. The table is a fully-associative structure. The D-JOLT shares one upper bit table with two miss tables.
3) Extra-miss table: When an address of a cache miss is outside the range of a single bit vector corresponding to a signature, the extra-miss table is used to record the address. The table is a set-associative structure accessed by the signatures same as miss table. The D-JOLT shares an extra-miss table with two miss tables.



(a) When a call/return instruction is fetched. (b) When a cache miss occurs.

Fig. 4: Behavior of D-JOLT. The miss table is simplified in this figure and actually consists of the three kinds of tables.

		FIFORETCNT		RASWHOLE	
PC	Ins.	State	Signature	State	Signature
A	Call	X Y Z A 0	h(X,Y,Z,A,0)	X Y Z A	h(X,Y,Z,A)
B	Call	Y Z A B 0	h(Y,Z,A,B,0)	Y Z A B	h(Y,Z,A,B)
-	Return	Y Z A B 1	h(Y,Z,A,B,1)	Y Z A -	h(Y,Z,A,B)^1
C	Call	Z A B C 0	h(Z,A,B,C,0)	Y Z A C	h(Y,Z,A,C)
-	Return	Z A B C 1	h(Z,A,B,C,1)	Y Z A -	h(Y,Z,A,C)^1
-	Return	Z A B C 2	h(Z,A,B,C,2)	Y Z - -	h(Y,Z,A)^1
-	Return	Z A B C 3	h(Z,A,B,C,3)	Y - - -	h(Y,Z)^1
D	Call	A B C D 0	h(A,B,C,D,0)	Y D - -	h(Y,D)

Fig. 5: Behavior of FIFORETCNT and RASWHOLE. In this figure, the histlen is set to four.

The components for generating the signatures consist of the following components.

1) FIFORETCNT: To implement FIFORETCNT, the long-range prefetcher and the short-range prefetcher each have their own FIFO. In addition, they include head pointers and counters that count the number of consecutive returns.
2) Signature queue: The signature queue records past signatures, and it is used to reference signatures in the past for *distance*. Because the long-range prefetcher and the short-range prefetcher have different signatures, they each have a signature queue.

The fallback prefetcher has several tables for detecting the streams used by the stream prefetcher [9].

B. Behavior

The long-range prefetcher and the short-range prefetcher work similarly. Fig. 4 shows the behavior of the learning and issuing prefetches in these prefetchers. In addition to the above prefetchers, the fallback prefetcher works based on the stream prefetcher described in [9]. In the following section, we describe the behavior of the long-range prefetcher and the short-range prefetcher.

1) Issuing prefetch: When a call/return instruction is fetched, D-JOLT calculates new signatures for each prefetcher. The details of the algorithm to generate the signature are discussed later. The obtained signature is pushed into the signature queue. Then, D-JOLT accesses the miss table, upper bit table, and extra-miss table with the updated signature. D-JOLT reconstructs raw prefetch line addresses from the obtained data, and issues prefetches.

2) Learning miss addresses: When a cache miss occurs, D-JOLT records missed line addresses into the tables using a signature obtained from the signature queue. This signature is one that was pushed to the queue *distance* times ago. A missed line address is divided into the upper and lower bits, and the bits are recorded into each table. If a signature hits on the miss table and the missed line address is not within the range of the bit vector in the entry, the missed line address is recorded in the extra-miss table.

3) Signature generation algorithm: Fig. 5 shows the behavior of FIFORETCNT and RASWHOLE. When a call/return instruction is fetched, FIFORETCNT calculates a new signature as follows.

When a call instruction is fetched, the address is pushed into the queue and the return counter is reset to zero. When a return instruction is fetched, the queue is not updated, and the return counter is incremented. After fetching a call/return instruction, it calculates a new signature from the *histlen* addresses obtained from the queue head and the return counter.

C. Implementation in a real design

The configuration of the D-JOLT used for the evaluation in this paper uses a large capacity to achieve high performance in the championship, but about 25 KiB is sufficient to achieve a similar performance in practical use.

III. EVALUATION

A. Methodology

We evaluated D-JOLT using ChampSim [1] according to the IPC1 rules with the traces on the IPC1 web-site [2]. The parameters of D-JOLT used in the evaluation are as follows. In the long-range prefetcher, the histlen is seven and the distance is fifteen. In the short-range prefetcher, the histlen and the distance is four. Table I lists the storage breakdown.

To compare with D-JOLT, we also evaluated next-three-lines prefetcher, RDIP[1] [5], Boomerang[1] [7], and Shotgun[1] [6]. In addition, to investigate the theoretical limit of the performance improvement by the L1I prefetcher, we implemented and evaluated an ideal model, where the memory access latency below the L1I cache in the instruction fetch is always one cycle. We did not use a perfect L1I cache as an ideal model, because the perfect L1I cache does not take into account the impact on the L2 cache, which is shared for instruction data and non-instruction data.

[1]We implemented and tuned these prefetchers with some parameter searching, but their parameters may not be the best.

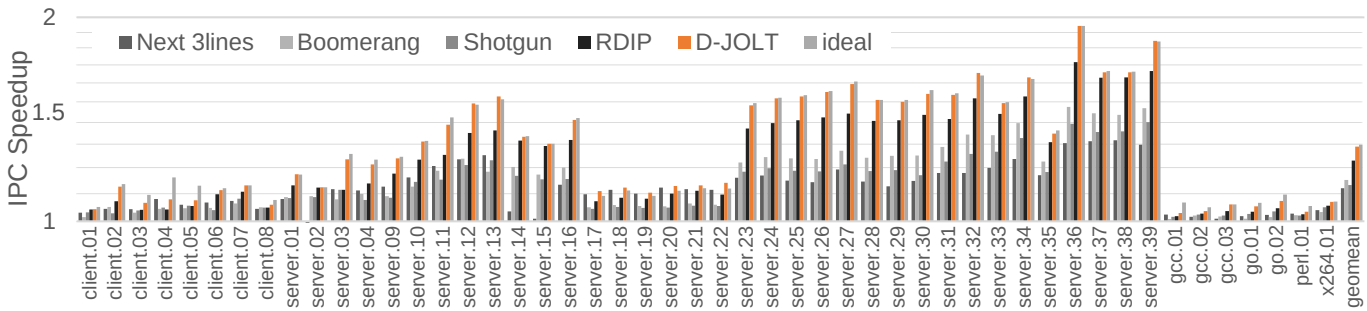


Fig. 6: IPC speedup over no instruction prefetching. Note that the vertical axis is in log scale.

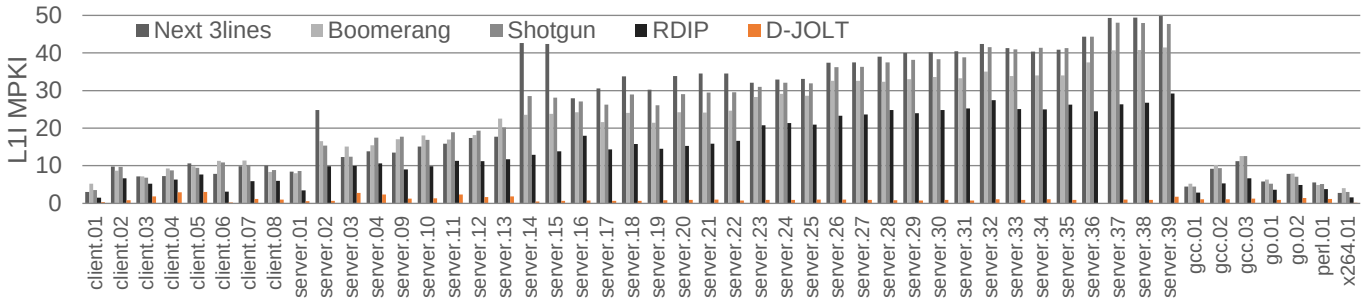


Fig. 7: L1I Miss per kilo instructions (MPKI)

TABLE I: Storage computation

Long-range Prefetcher	Signature Generator	FIFO (7x32+3 bits), return counter (32 bits)		259
	Signature Queue	Signature (23 bits)	15 entries + misc.	349
	Miss Table	Tag (12 bits), LRU (2 bits), miss vectors* (2x31 bits)	2048 Sets, 4-way (8192 entries)	622592
Short-range Prefetcher	Signature Generator	FIFO (4x32+2 bits), return counter (32 bits)		162
	Signature Queue	Signature (23 bits)	4 entries + misc.	94
	Miss Table	Tag (13 bits), LRU (2 bits), miss vectors* (2x31 bits)	1024 Sets, 4-way (4096 entries)	315392
Extra-miss Table		Tag (15 bits), LRU (2 bits), miss vectors* (2x31 bits)	256 Sets, 4-way (1024 entries)	80896
Fallback Prefetcher	Stream Train Table	Valid (1 bit), LRU (4 bits), line address (58 bits), counter (2 bits)	16 entries	1040
	Stream Monitor Table	Valid (1 bit), LRU (4 bits), line address (58 bits)	16 entries	1008
Upper Bit Table		Valid (1 bit), upper bit (40 bits)	31 entries	1271
Storage overhead				**1023062 bits (125 KiB)**

* A miss vector (31 bits) = compressed upper address (5 bits) + lower address (18 bits) + bit vector (8 bits)

TABLE II: Simulation parameters

Front-end Module	Parameter
Fetcher	6-way, 12 entries fetch buffer
Branch direction predictor	64 KiB hashed perceptron, ghist 232 bits
Branch target predictor	Oracle
Decoder	7-way, 3 cycle, non-pipelined

Memory	Capacity	Hit Latency	Prefetcher
L1I Cache	32KiB	4 cycle	Evaluation Target
L1D Cache	48 KiB	5 cycle	Next Line Prefetcher
L2 Cache	512 KiB	18 / 20 cycle	Signature Path Prefetching (5.4 KiB)
L3 Cache	2 MiB	48 / 50 cycle	N/A
Main Memory	4 GiB	>118 cycle	N/A

All simulation results are warmed up with 50 M instructions and simulated for additional 50 M instructions. The simulation parameters are shown in Table II.

B. Performance

Fig. 6 shows the IPC speedup of all the evaluated models and the ideal model over a no-instruction-prefetching baseline. This result shows that D-JOLT achieved performance improvement close to the ideal model in almost all traces. In geomean, D-JOLT achieved a 28.9% improvement, which is 4.8% higher than that of RDIP. In client_04, client_05, and gcc_01, D-JOLT has room for performance improvement. In these traces, D-JOLT generated many kinds of signatures and the storage capacity was not enough to record the signatures.

Fig. 7 shows the L1I cache miss per kilo instructions (MPKI) of all the evaluated prefetchers. The D-JOLT significantly reduced the L1I MPKI to 3.0 or less for the all traces.

IV. Conclusion

We propose D-JOLT, consisting of multiple prefetchers with different characteristics. The long-range prefetcher predicts a distant future with higher coverage, while the short-range prefetcher predicts a near future with higher accuracy. We evaluated D-JOLT with distributed traces according to the IPC1 rule, and the results show a 28.9% improvement in performance compared to a processor without instruction prefetching.

Acknowledgements

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

References

[1] "Champsim," https://github.com/ChampSim/ChampSim/.
[2] "IPC1 traces," https://research.ece.ncsu.edu/ipc/infrastructure/.
[3] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive instruction fetch," in *MICRO*, 2011, pp. 152–162.

[4] M. Ferdman, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Temporal instruction fetch streaming," in *MICRO*, 2008, pp. 1–10.
[5] A. Kolli, A. Saidi, and T. F. Wenisch, "RDIP: return-address-stack directed instruction prefetching," in *MICRO*, 2013, pp. 260–271.
[6] R. Kumar, B. Grot, and V. Nagarajan, "Blasting through the front-end bottleneck with shotgun," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 30–42, 2018.
[7] R. Kumar, C.-C. Huang, B. Grot, and V. Nagarajan, "Boomerang: A metadata-free architecture for control flow delivery," in *HPCA*, 2017, pp. 493–504.
[8] G. Reinman, B. Calder, and T. Austin, "Fetch directed instruction prefetching," in *MICRO*, 1999, pp. 16–27.
[9] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *HPCA*, 2007, pp. 63–74.