Berti: an Accurate Local-Delta Data Prefetcher

Abstract

Data prefetching is a technique that plays a crucial role in modern high-performance processors by hiding long latency memory accesses. Several state-of-the-art hardware prefetchers exploit the concept of deltas, defined as the difference between the cache line addresses of two demand accesses. Existing delta prefetchers, such as best offset prefetching (BOP) and multi-lookahead prefetching (MLOP), train and predict future accesses based on global deltas. We observed that the use of global deltas results in missed opportunities to anticipate memory accesses. In this paper, we propose Berti, a first-level data cache prefetcher that selects the best local deltas, i.e., those that consider only demand accesses issued by the same instruction. Thanks to a high-confidence mechanism that precisely detects the timely local deltas with high coverage, Berti generates accurate prefetch requests. Then, it orchestrates the prefetch requests to the memory hierarchy, using the selected deltas. Our empirical results using ChampSim and SPEC CPU2017 and GAP workloads show that, with a storage overhead of just 2.55 KB, Berti improves performance by 8.5% compared to a baseline IP-stride and 3.5% compared to IPCP, a state-of-theart prefetcher. Our evaluation also shows that Berti reduces dynamic energy at the memory hierarchy by 33.6% compared to IPCP, thanks to its high prefetch accuracy.

Keywords: hardware prefetching, first-level cache, local deltas, accuracy, timeliness

1 Introduction

Data prefetching techniques play an important role in hiding long-latency memory accesses. Hardware prefetchers learn memory access patterns and fetch data into the cache hierarchy before time so that future memory accesses get cache hits. Data prefetching techniques can be employed either at the private first-level data cache (L1D), second-level cache (L2), or at the shared last-level cache (LLC).

2 Related works

2.1 Spatial prefetchers

Spatial prefetchers predict strides, streams, or complex strides within a spatial region providing competitive coverage. Prefetchers like variable length delta prefetching (VLDP) [45] and signature path prefetching (SPP) are well known delta prefetchers. VLDP stores the history of deltas to predict future deltas. SPP is a state-of-theart delta prefetcher that predicts the non-constant (irregular) strides (commonly known as deltas). SPP works based on the signatures (hash of deltas) seen within a physical OS page to index into a prediction table that predicts future delta. It dynamically controls the prefetch aggressiveness based on the success probability of future deltas. Spatial Memory Streaming (SMS) , is a spatial prefetcher that exploits the relationship between the IP of a memory request, and access pattern within a spatial region based on the IP and the first offset within that region. SMS incurs huge storage overhead, which is larger than the L1-D size. A recent work called Bingo, uses multiple signatures (like IP, IP+Offset, and memory region) and fuses them into a single hardware table. Bingo provides better coverage than SMS. However, Bingo incurs similar overhead as SMS (around 119KB). There are component prefetchers like division of labor (DOL) that target specific program semantics (like pointer chains, loops, etc.) for prefetching by getting the information of interest from the processor core.

2.2 Offset prefetchers

Offset based prefetchers such as Bestoffset Prefetcher (BOP) and Sandbox prefetcher explore multiple offsets. An offset of k means the cache block that is distanced by k cache blocks. These prefetchers choose the offset that provides the maximum likelihood of future use. BOP continues to prefetch with a particular offset till a new offset performs better than the current offset. Multi-Look-ahead Offset Prefetcher (MLOP) is an extension of BOP that considers several lookaheads for each offset, and finds the best offset for each look-ahead. MLOP is motivated by Jain's Ph.D. thesis that proposed an Aggregate Stride Prefetcher (ASP).

2.3 Temporal Prefetchers

Temporal prefetchers like temporal streaming, Irregular Stream Buffer (ISB), and Domino track the temporal order of accesses. Usually, temporal prefetchers demand hundreds of KBs. Recently, Managed ISB (MISB) and Triage have optimized the hardware overhead without compromising coverage.

2.4 Prefetch filters/throttlers

To further improve the effectiveness of hardware prefetchers, prefetch filters like PerceptronPrefetch-Filter (PPF) and Evicted-Prefetch-Filter (EPF) [43] have been proposed. Apart from filters, there are aggressiveness controllers (throttlers) that control the prefetch degree and prefetch distance based on prefetch metrics like accuracy, coverage, LLC pollution, and DRAM bandwidth. Dual Spatial Pattern Prefetcher (DSPatch) is an adjunct spatial prefetcher that works like a throttler. It improves the effectiveness of SPP based on DRAM bandwidth utilization.

2.5 Instruction pointer classifier prefetching (IPCP)

The winner of DPC-3 is a state-of-the-art L1D data prefetcher that is composite in nature [40]. It classifies an IP into three classes: constant stride (CS), complex stride (CPLX), and global stream (GS). IPCP uses three lightweight prefetchers that issue prefetch requests according to the IP class. If it fails to classify an IP into one of the three classes, it uses a next-line prefetcher.

3 Method

Berti is a data prefetcher sited at the L1D, where it can see all the requests generated by the processor and orchestrate the prefetch requests to the memory hierarchy. Berti makes a strong case for prefetch accuracy. For each IP, it selects the deltas2 that are timely and computes their respective local coverage. High accuracy is achieved by only using deltas with high coverage.

3.1 Training the prefetcher

3.1.1 Measuring fetch latency

In order to learn the deltas that are timely it is necessary to measure the time required to fetch data to the L1D, i.e., the L1D miss latency. This measurement is performed for any cache line in L1D, both for demand misses and prefetch requests. Computing latency for prefetch requests is fundamental because, in an ideal scenario, there would not be L1D misses but just L1D hits due to timely prefetch requests. In addition, the latency of prefetch requests may be larger than the latency of demand requests due to prefetch queue (PQ) contention or L1D port contention. Fetch latency can be measured by keeping a timestamp for any L1D miss inserted into the MSHR and any prefetch request inserted into the PQ. On an L1D fill, the latency is simply computed by subtracting the stored timestamp from the current one.

3.1.2 Learning timely and accurate deltas

Once the fetch latency is obtained for each L1D fill, our prefetcher can precisely learn timely deltas, given that the history of accesses and timestamps by the same IP is recorded. By searching in the history of recent accesses and comparing the timestamp of each previous access with the timestamp when a prefetch should be issued to be timely, the accesses that would trigger timely prefetch requests are detected. Deltas are then computed by subtracting the address of each timely request in the history from the current address. Figure 4 depicts how timely deltas are detected. All addresses represented in the timeline are accessed by the same IP. When address 10 is demanded and its fetch latency computed , the history of accesses for that IP is searched, from the point in time, a timely prefetch should have been triggered. In this case, no previous accesses are found. After accessing address 12 and computing its fetch latency, a timely delta corresponding to address 2 is found. That is, address 2 should initiate the prefetch request for 12 in order to be timely. The timely delta +10 is therefore learned. Similarly, when computing the latency for access 15, two deltas, +10 and +13, are detected as timely. Berti triggers the procedure to learn timely deltas for each miss that would have occurred in the baseline, which translates to two scenarios. First, when a demand miss fills the L1D with the requested data. Second, when a cache line brought into L1D by a prefetch request is demanded (i.e., misses that would have occurred without a prefetcher). Berti does not learn deltas on a cache fill caused by a prefetch request since its demand time is not known. Therefore, it is necessary to keep the latency of prefetch requests until the core demands the cache line.

3.1.3 Computing the coverage of deltas

On every search in the history, Berti obtains a set of timely deltas. Deltas that frequently appear in the searches would cover a significant fraction of misses, while deltas that rarely appear would result in low coverage. It is easy to compute the coverage by dividing the number of occurrences of a delta by the number of searches in the history. For example, in Figure 4, after three accesses, the delta +10 has the higher coverage, being in two out of three searches (66.7%). If the same access pattern continues, the delta +10 will reach close to 100% coverage. It is important to note that this local (per IP) coverage translates into accuracy. If a delta covers 100% of cache lines, since each access-delta pair results in only one prefetch request, that delta will bring 100% accuracy.

3.2 Prediction: issuing prefetch requests

Once we know the deltas and their associated coverage, we can orchestrate the prefetch requests across the cache hierarchy. Based on both the coverage of each delta and the L1D MSHR occupancy, we decide which deltas to use and till which cache level to prefetch. We use four watermarks to decide where to issue the prefetch requests. If the coverage of a delta is above a high-coverage watermark and the L1D MSHR occupancy is below the occupancy watermark, then prefetch requests using that delta get filled at all the cache levels till L1D. Otherwise, if the coverage is above a medium-coverage watermark, irrespective of the L1D MSHR occupancy, prefetch requests get filled till L2. Finally, if the coverage is above a low-coverage watermark, requests get filled only in the LLC. To generate a prefetch request, we add the selected delta to the address of the current access and the resulting address is inserted in the PQ. Requests in the PQ are processed in a first-in-first-out (FIFO) order. Since our prefetcher is trained with virtual addresses, the generated prefetch requests are also in the virtual address space. A prefetch request obtains the physical address from the L2 translation look-ahead buffer (STLB). If the translation misses in the STLB, the prefetch request is dropped. If the translation is obtained, the prefetch request checks if the target block is already present in the cache it wants to fill. In case of a miss, the block is prefetched, and the request is inserted into the MSHR.

4 Implementation details

4.1 Comparing with the released source codes

I use the open source code provided by the paper to reproduce the work.

4.2 Experimental environment setup

The artifact is ready to be built and run automatically by executing the run.sh script. The overall flow is as follows:

1. Clone the artifact:

   git clone https://github.com/agusnt/Berti-Artifact

2. Enter the cloned repository:

   cd Berti-Artifact

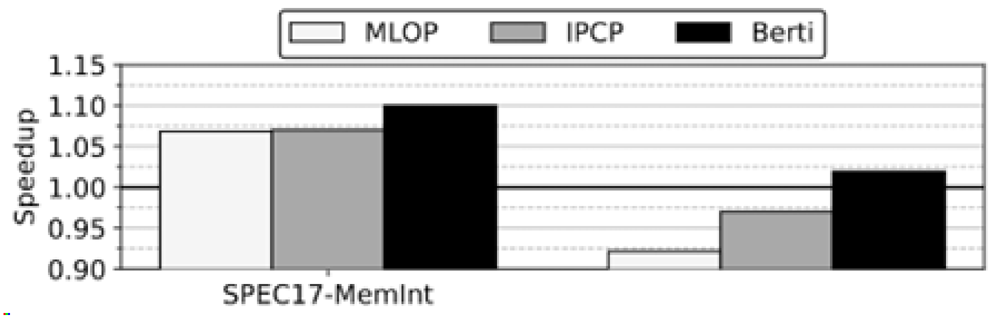

Figure 1

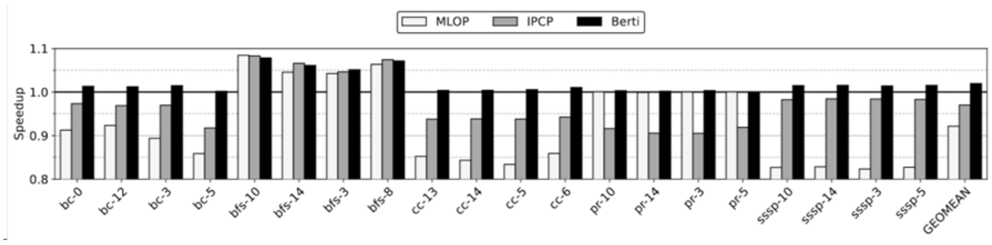

Figure 2

3. Run the script:
   ./run.sh -p 64

5 Results and analysis

The figure 1 - 5 illustrates the prefetching accuracy on L1 Dcache. The request percentage is divided into timely (gray) and delayed (black) prefetching requests.

The darker part in each bar of the figure represents a prefetch request for which the retrieved data arrives late at L1D. Almost all prefetch requests generated by Berti are timely, while MLOP and IPCP generate a large number of delayed requests. IPCP does not use any mechanism to adjust the timing of prefetch requests, so timeliness performs worst. Each IP has specific, timely deltas. So Berti has better timeliness than MLOP.

The figure 6 shows the number of instruction misses per thousand (MPKI) for L1D, L2, and LLC for different combinations of prefetches.

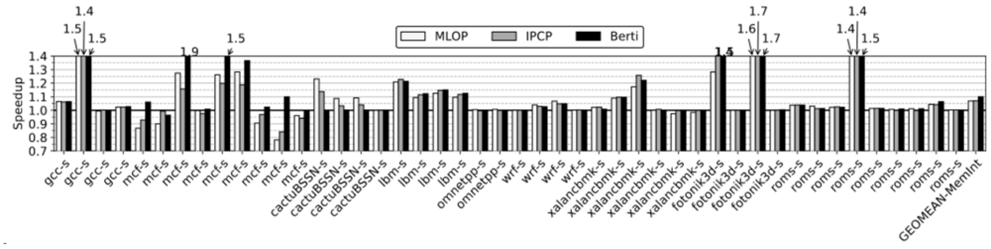

Figure 3

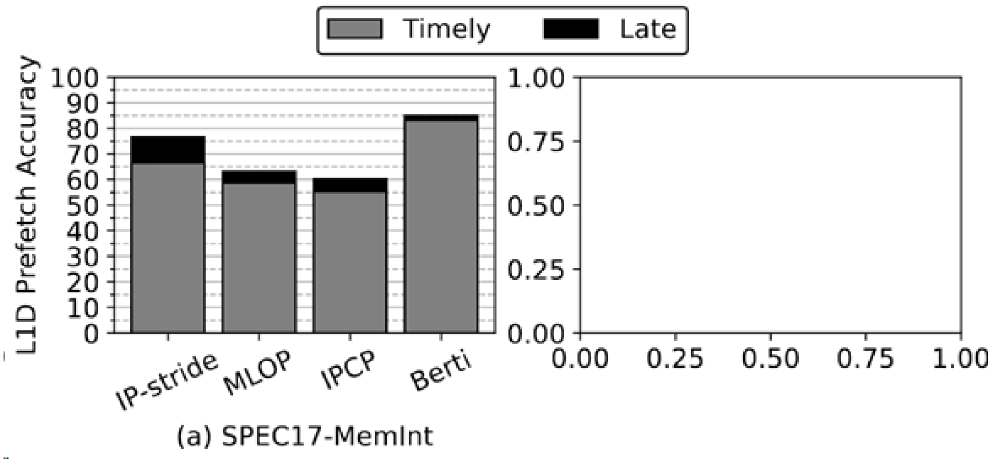

Figure 4

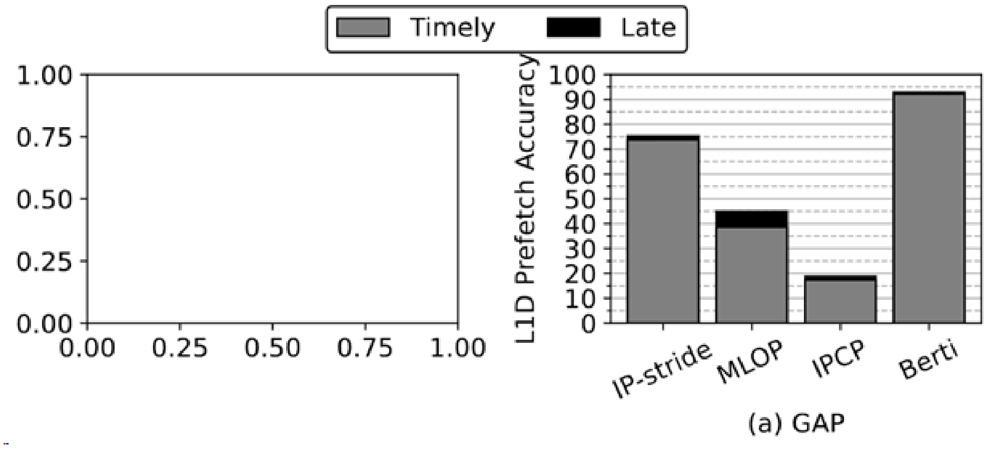

Figure 5

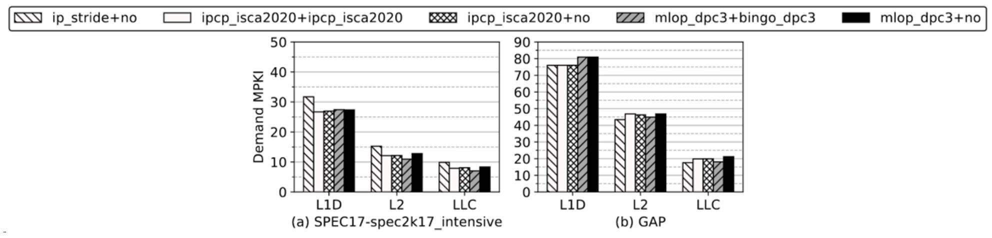

Figure 6

6 Conclusion and future work

We proposed Berti and made a case for an L1D prefetcher based on local, timely deltas. Berti learns the best delta to prefetch, keeping timeliness (in the form of time to prefetch an address) and prefetch accuracy in mind. We showed that Berti could learn varieties of memory access patterns. We quantified the effectiveness of Berti across SPEC CPU2017 and GAP workloads, and showed high prefetch accuracy and timely prefetching into the cache hierarchy. On average, Berti outperforms state-of-the-art L1D and L2 prefetchers. Berti is equally effective even in the constrained DRAM bandwidth scenarios and also for multi-core mixes. Berti consumes the least dynamic energy at the memory hierarchy among all state-of-the-art prefetchers. In summary, Berti provides high prefetch accuracy, timely prefetching, and good coverage with a limited storage overhead of 2.55 KB per core. [1]

References

[1] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros. Berti: an accurate local-delta data prefetcher. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 975–991, 2022.