A Complexity-Effective Local Delta Prefetcher

Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros

***Abstract*—Data prefetching is crucial for performance in modern processors by effectively masking long-latency memory accesses. Over the past decades, numerous data prefetching mechanisms have been proposed, which have continuously reduced the access latency to the memory hierarchy. Several state-of-the-art prefetchers, namely Instruction Pointer Classifier Prefetcher (IPCP) and Berti, target the first-level data cache, and thus, they are able to completely hide the miss latency for timely prefetched cache lines.**

Berti exploits timely local deltas to achieve high accuracy and performance. This paper extends Berti with a larger evaluation and with extra optimizations on top of the previous conference paper. The result is a complexity-effective version of Berti that outperforms it for a large amount of workloads and simplifies its control logic. The key for those advancements is a simple mechanism for learning timely deltas without the need to track the fetch latency of each cache miss. Our experiments conducted with a wide range of workloads (CVP traces by Qualcomm, SPEC CPU2017, and GAP) show performance improvements by 4.0% over a mainstream stride prefetcher, and by a non-negligible 1.4% over the previously published version of Berti requiring similar storage.

***Keywords*-Data prefetching; hardware prefetching; first-level cache; stride; local deltas; accuracy; timeliness.**

I. Introduction

Data prefetching plays an important role in improving the performance of modern processors. The prefetching mechanisms aim to predict the data blocks that the processor will request during program execution and to preload them in cache. In this way, subsequent data requests may result in fast cache hits instead of costly misses.

Hardware data prefetching mechanisms can be located close to the first-level data cache (L1D), the second-level cache (L2), or the last-level cache (LLC). Most of the recently proposed storage-efficient prefetchers target L2 [9], [11], [13], [40], [40]. Exceptions are the Multi-Lookahead Offset Prefetching (MLOP) [36], the Instruction Pointer Classifier Prefetching (IPCP) [32], and Berti [31], which are L1D prefetchers. An L1D prefetcher has a higher performance potential than an L2 prefetcher for several reasons. First, an L1D prefetcher sees all accessed virtual addresses, whereas an L2 prefetcher only sees accesses for physical addresses that are missing in L1D. Second, prefetching into L1D hides the latency better than prefetching into L2. Third, an L1D prefetcher has easier access to contextual information, such as the instruction pointer (IP) [28] in Intel's IP-Stride prefetcher [16].

Agustín Navarro-Torres and Alberto Ros are with the Computer Engineering Department, University of Murcia, 30100 Murcia, Spain. E-mails: agustin.navarro@um.es, aros@ditec.um.es

Biswabanda Panda is with the Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, India, E-mail: biswa@cse.iitb.ac.in.

Jesús Alastruey-Benedé, Pablo Ibáñez and Víctor Viñals-Yúfera are with the Computer and System Engineering Department, Universidad de Zaragoza, Zaragoza, Spain. E-mails: jalastru@unizar.es, imarin@unizar.es, victor@unizar.es.

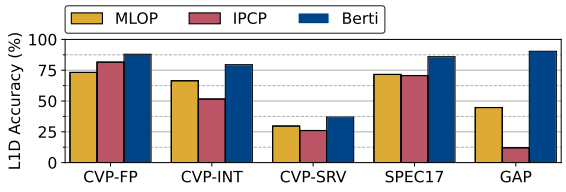

Fig. 1. Accuracy of L1D state-of-the-art prefetchers (MLOP, IPCP, and Berti), across different memory-intensive traces taken from CVP, SPEC CPU2017, and GAP.

However, the design of an L1D prefetcher is a challenging task as the space constraints inherent to its placement impose certain requirements that impact performance: (i) high prefetch accuracy to avoid L1D pollution, eviction of useful cache lines, and generation of memory traffic overhead, (ii) degree of prefetch restricted by the limited size of structures such as the prefetch queue (PQ) and the miss status holding registers (MSHR), and (iii) limitation of the logic complexity and the storage capacity needed to implement the prefetcher mechanism.

A. Berti, an accurate local delta prefetcher

State-of-the-art data prefetchers push the limit of single-thread performance, with average performance boosts from 3% to 5% [9], [13], [32]. However, some prefetchers load a large amount of useless blocks, resulting in suboptimal performance [26]. Fig. 1 shows the average prefetching accuracy in various benchmark suites for three state-of-the-art L1D prefetchers: MLOP [36], IPCP [32], and Berti [31]. For example, in the GAP benchmark suite, the accuracy of MLOP and IPCP is 44.7% and 12.1%, respectively. This means that 55.3% and 87.9% of the cache blocks prefetched by MLOP and IPCP enter L1D but are not accessed by memory instructions.

On the other hand, Berti, a per-IP *best request time* delta L1D prefetcher that makes a strong case for timeliness and accuracy, brings only 9.6% useless blocks for GAP (90.4% accuracy). Berti learns the deltas that result in timely prefetch requests, and issues prefetch requests only for the deltas predicted to provide high coverage, which translates to overall high prefetch accuracy. Located in the L1D and seeing all virtual addresses generated by the processor, Berti orchestrates the prefetch requests for the memory hierarchy.

Timely local deltas. Berti defines local delta as the difference in cache line addresses between *any two* demand accesses issued by the same memory access instruction (*same IP*). On the other hand, stride refers to the difference between

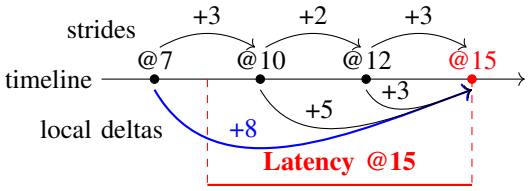

Fig. 2. Stride, local delta, and timely local delta. The values on the timeline (@7, @10, @12 and @15) represent cache line addresses referenced by the same instruction.

two *consecutive* addresses by the same IP. Fig.2 illustrates an example of stride, local delta, and timely local delta. An address sequence issued by a single load instruction (same IP) is shown: 7, 10, 12, 15. The corresponding stride sequence is: *+3, +2, +3*. However, access to address 15 is at delta +3, +5, and +8 with respect to the previous three accesses. If the goal of a prefetcher is to *cover* address 15, the prefetcher could initiate prefetching with deltas +3, +5, or +8 upon seeing demand accesses to addresses 12, 10, and 7, respectively. However, considering the time to fetch the cache line with address 15, prefetching with deltas +3 and +5 will not fully mitigate the L1D miss latency, since they will be late prefetch requests. In contrast, if a prefetcher issues a request for address 15 with delta +8 from demand access to address 7, the cache line of address 15 could be prefetched early enough. For more benefits about the use of delta versus stride, we refer the reader to previous work [30], [31].

Berti finds the *timely* local deltas, and computes its respective coverage. It prefetches using deltas that used to show high coverage, which translates to overall high prefetch accuracy.

B. Pushing the limits of Berti

This paper extends the Berti prefetcher [31] with two objectives in mind. The first objective is to asses the benefits of Berti for a larger number of workloads and fine-tune it, thus setting a higher baseline for timely local delta prefetchers. To this end, we performed an extensive characterization of L1D prefetchers using also data center workloads that were provided by *Qualcomm* for the first Championship Value Prediction (CVP-1) [4] (see Section IV-C for further details). The second objective is to take Berti a step forward towards a more feasible hardware implementation by simplifying its logic. The end result is a simpler design that brings performance benefits over Berti.

Our first contribution is to extend the analysis of the state-of-the-art prefetchers presented in Berti's paper with a wider range of traces (Section V). Fig. 3 shows the speedup results of MLOP [36], IPCP [32], and Berti [31] prefetchers relative to IP-Stride prefetching [16] in the proposed extended evaluation, i.e., the CVP traces together with SPEC CPU2017 and GAP. While Berti is the current leading prefetcher in SPEC CPU2017 and GAP workloads, its performance falls behind competitors like MLOP by 2.4% in CVP-FP (floating point) traces. It also shows slightly worse or similar performance to MLOP and IPCP in CVP-INT (integer) or IP-Stride in CVP-SRV (server), respectively. We found that this suboptimal

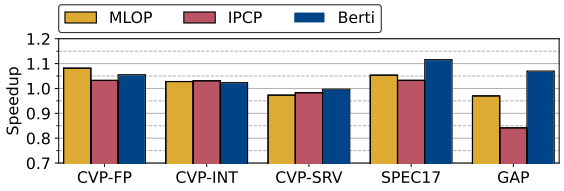

Fig. 3. Speedup of L1D state-of-the-art prefetchers: MLOP, IPCP, and Berti relative to IP-stride prefetching, across different memory-intensive traces taken from CVP, SPEC CPU2017, and GAP.

performance is due to the lack of learning patterns in Berti compared to its competitors, due to the limited number of loads that Berti is able to track. Although the accuracy of Berti is still high, its coverage drops.

Our second and most important contribution is reducing Berti's complexity without losing performance. We found that to learn deltas accurately, it is not necessary to count cycle-accurate latencies of all misses and prefetch requests; instead, it is sufficient to know the identity of the memory level that serves the cache miss and its average access latency (Section III-A). Thanks to this finding, our design simplifies the prefetcher logic and reduces its storage requirements.

Our third contribution relates to performance. We optimized the prefetching structures used by Berti. We found that the hash function used to reduce Instruction Pointer (IP) aliasing is critical for performance in applications with very large code footprints, as is the case for the server and compute-intensive CVP traces. Hence, we explore set-associative structures with different hash functions for indexing them and different replacement policies (Section III-B).

Our evaluation shows, that for Qualcomm data center, SPEC CPU2017 and GAP workloads, our prefetcher consistently outperforms a baseline IP-stride prefetcher (4.0% average speedup) and improves Berti by a non-negligible 1.4% average speedup, further pushing the limits of hardware prefetchers.

II. Recent Works and Motivation

A. Recent advances in data prefetching

Data prefetching plays an important role in designing high-performance processors. Recent developments in this field come mainly from the last two Data Prefetching Championships, DPC-2 [2] and DPC-3 [5], co-located with ISCA 2015 and ISCA 2019, respectively.

Berti. Berti is the state-of-the-art hardware prefetcher. It learns timely local deltas to ensure that prefetch requests are performed on timely basis, with an accuracy higher than 90%. A detailed description can be found in Section II-B.

Best offset prefetching (BOP). The winner of DPC-2 is a degree one L2 prefetcher that finds a global offset that provides the maximum likelihood of future use in the L2 cache [30]. An offset of k means that a cache line is k cache lines away from the current demand address. BOP takes timeliness into account while selecting the best offset per application phase. Multi-lookahead offset prefetching (MLOP) [36] is an extension of BOP that is motivated by Jain's Ph.D. thesis [21]. MLOP considers multiple lookaheads for each offset and selects the offset and lookahead that cover a specific cache miss. Both

BOP and MLOP treat the demand addresses in isolation, and for each demand access, trigger prefetch requests based on the prefetch offset[1]. In general, MLOP provides better prefetch coverage than BOP.

Variable Length Delta Prefetching (VLDP). This spatial data prefetcher uses multiple histories of deltas between successive cache lines observed within an operating system (OS) page to predict future memory accesses in other OS pages [38]. One of the key features of VLDP is that it uses multiple prediction tables and makes predictions based on different lengths of history in terms of deltas.

Signature path prefetching (SPP). This state-of-the-art delta prefetcher predicts irregular strides in L2 [27]. SPP works by relying on the signatures (hashes of consecutive strides) observed within an OS page to index into a table that predicts future deltas. SPP uses a lookahead mechanism that recursively finds out deltas to prefetch until a delta falls below a given *confidence*. Perceptron prefetch filtering (PPF) is a filter that further improves the effectiveness of SPP by deciding whether to prefetch into L2 or not [13]. In general, SPP combined with PPF (SPP-PPF) provides better prefetch coverage than VLDP.

Bingo. This L2 prefetcher makes a case for associating spatial access patterns to both short (such as IP) and long events (IP+offset and memory region) and selecting the best pattern for prefetching [9]. A key point of Bingo is the use of only one hardware table for both short and long events. This table enables multiple predictions from a single entry, providing better coverage than single-event prefetching. In general, Bingo outperforms VLDP and SPP-PPF for SPEC CPU2017 traces. However, it requires significantly more storage than VLDP and SPP-PPF.

Instruction pointer classifier prefetching (IPCP). The winner of DPC-3 is a state-of-the-art L1D data prefetcher that is composite in nature [32]. IPCP 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.

B. Berti: Overview

Berti is a state-of-the-art prefetcher capable of improving performance over IPCP, the DPC-3 winner, by 3.5%. The key concepts of Berti are timeliness and high-coverage deltas. Next, we describe how Berti performs training and prediction and list the hardware structures required for its implementation.

Training the prefetcher. Berti monitors the patterns of each memory access instruction (loads and stores), learns the timely local deltas, and computes the delta's coverage as the probability of being a useful prefetch request. The training consists of four actions:

1) *Building the history.* Berti keeps a record of up to the 16 latest memory misses or prefetch hits for a given IP. This record is stored in a *history table* that is written to as demand misses and demand hits due to previous prefetch requests occur.
2) *Measuring fetch latency.* In order to learn the timely deltas of a memory access, it is required to know its fetch latency. That is, the amount of time it takes for each memory request to fill the L1D cache. The fetch latency is calculated as the difference between two timestamps, the L1D fill timestamp minus the timestamp of the L1D demand access or prefetch that triggered the cache line request to the higher cache levels. A new field of 16 bits is required in all entries of the prefetch queue (PQ) and the miss status holding registers (MSHR) to keep such timestamps.
3) *Learning timely deltas.* Once the request fills the L1D and its fetch latency is calculated, Berti searches the history table for the previous demand miss or hit because of a prefetch request of the same IP that could trigger a timely prefetch request. The deltas are computed as the difference between the actual address and the stored address.
4) *Computing timely delta coverage.* In each search of the history, Berti can retrieve a set of deltas that repeats frequently and covers a high fraction of misses, while other deltas rarely repeat. These frequently seen deltas, with high coverage, are candidates for triggering prefetch requests, while the other deltas are discarded. To calculate its coverage, Berti divides the number of times a delta is seen by the number of searches in the history. It is important to note that the local coverage of a delta (as computed in Berti) is directly related to its accuracy.

Issuing prefetch requests. Upon each IP access, Berti retrieves the previously learned deltas and their coverage to orchestrate prefetch requests. Deltas with coverage exceeding a high-coverage watermark are targeted for L1D prefetch. If the delta coverage surpasses a middle-coverage watermark, the prefetch is directed to fill L2. Otherwise, the delta is discarded. Berti also considers the L1D MSHR occupancy to select the target level. If the occupancy is high, the prefetch will go to L2 even if the delta has high coverage.

Hardware implementation. The hardware implementation of Berti requires extending three existing data structures: L1D, PQ, and MSHR, which account for 47% of Berti's storage overhead and are used for measuring fetch latency and learning timely delta steps. Additionally, two new structures are added: the *history table*, a cache-like structure that stores the history of misses, and the *delta table*, a fully associative table indexed by IP that keeps track of the most frequent deltas.

C. Complexity and effectiveness of Berti

Logical complexity: Berti achieves a significant performance improvement with low-storage overhead. However, the hardware implementation of Berti still requires a high number of logical components (adders, comparators, multiplexers, and highly-associative structures) to calculate and store the fetch latency. These components make the implementation of Berti

[1]For BOP and MLOP, we use the term *global delta* instead of *offset* for the rest of the paper.

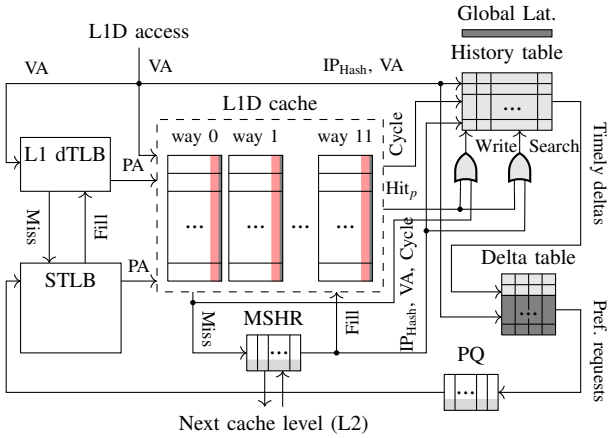

Fig. 4. Berti design overview. Hardware extensions for Berti are shown in light gray, and Berto extensions in dark gray. Elements in red were present in Berti but removed in Berto.

in real hardware although feasible, complex and expensive in terms of logical components and energy consumption.

Size of the structures: As shown in Section I, Berti is the best performing prefetcher in SPEC CPU2017 and GAP workloads. However, it has a suboptimal performance in the CVP-FP and CVP-INT traces. This is because Berti's fully associative structures are unable to track all the different IP addresses associated with memory demand requests. However, increasing its size is not optimal in terms of energy consumption and logical components.

III. Complexity-effective local delta prefetching

This section proposes a set of improvements over the baseline Berti prefetcher, focusing on reducing the logical complexity of Berti while improving performance for datacenter workloads (CVP traces). We refer to our resulting prefetcher mechanism as Berto (**Be**st-**r**equest-**t**ime delta **o**ptimized). Fig. 4 depicts the hardware architecture of Berto and highlights the original elements borrowed from Berti (shown in light gray), the extensions introduced in Berto (shown in dark gray), and the elements removed from Berti (shown in red) and Table I details the storage requirements. First, we explain the simplification of latency computation. Next, we fine-tune the prefetching structures to accommodate more IPs and reduce aliasing.

A. Simplifying latency computation

A large portion of Berti's storage overhead and hardware logical complexity is measuring and storing the fetch latency of the L1D misses, needed to learn the timely deltas. Berti uses a 12-bit field per L1D entry to store the fetch latency of any prefetched cache line.

Berto replaces that expensive storage overhead with i) a single bit per L1D entry indicating whether the block comes from the L2/LLC or DRAM level, and ii) a global 64-bit field (`Global Lat.`) that measures the average main memory latency.

The average latency is calculated for each fill from DRAM as a moving average: $Avg_{n+1} = \alpha \cdot Lat_n + (1-\alpha) \cdot Avg_n$, where

TABLE I
Storage overhead of Berto.

Structure		Storage
History table	8-set, 16-way (128-entry) cache, FIFO replacement policy. Each set: 4 bits (replacement policy). Each entry: 7-bit IP tag, 24-bit address, 16-bit timestamp	0.74 KB
Delta table	64 entries (16 sets, 4 ways), 4-bit NRU replacement policy. Each entry: 10-bit IP tag, 4-bit counter, and an array of 16 Deltas (13-bit delta, 4-bit coverage, 2-bit status)	2.49 KB
PQ + MSHR	16+16 entries, 16-bit timestamp per entry	0.06 KB
L1D	768 cache lines, 1-bit latency per line	0.09 KB
Total		**3.38 KB**

Avg_n indicates the average in the `n` step, α is a value between 0 and 1 and Lat_n is the latest measured DRAM latency (fill cycle - request cycle). Once the memory level that serves the cache line is known, Berto will look into the history table to identify the previous memory requests of the same IP that were able to trigger a timely prefetch request.

- If the block came from L2 or LLC, all entries with matching IPs in the history table are selected as candidates. The reason is that for short-latency misses there is no clear advantage in skipping accesses from the history table. This reduces the amount of computation required.
- If the block came from DRAM, the history entries that arrived later than the current cycle minus the average DRAM latency are discarded. For example, assuming that in cycle 1000 a block arrives from main memory, which shows an average latency of 100 cycles, all matching entries in the history table with a timestamp greater than cycle 900 will not be considered to compute deltas, as they do not guarantee timeliness. This is similar to that in Berti, but using the average latency to memory offers extra performance benefits and saves considerable storage requirements. In particular, this approach allows us to reduce the additional hardware overhead of L1D by reducing the latency field from 12 bits to a single bit (91.6% less storage overhead).

We employ a value of α of 1/8, such that we do not require performing multiplications and the update of the average latency can be performed by shifting and adding registers. This value obtained slightly better performance than 1/4.

B. Tracking effectively more IPs

Berto also modifies the hash function that indexes all structures (history and delta tables) and extends the size of the delta table to track more IPs, required by large-instruction-footprint server workloads.

1) The hash function: To index history and delta tables Berti employs the following hash function: $\texttt{IP}_{\texttt{Hash}}\texttt{=(IP>>1)}\oplus\texttt{(IP>>4)}$, giving satisfactory results on x86 traces. However, when considering CVP (Aarch64) traces, an aliasing problem appears, i.e. several IPs with different behaviors are mixed in the same entry, leading to incorrect learning. To reduce aliasing Berto uses the hash function used in the Entangling instruction prefetcher [35]:

IP_{Hash}=IP⊕(IP>>1)⊕(IP>>5). This hash function is slightly more complex, but has proven to make better use of space and improves performance.

2) Set-associative larger delta table: Berti uses a 16-way fully associative table with a FIFO replacement for storing deltas. However, server workloads require a larger number of IPs to be tracked. Although further increasing the number of ways is feasible, it would require more complex logic (replacement, more comparators, wider multiplexers), leading to longer latency and higher energy consumption. Berto instead implements the delta table with a set associative design (16 sets, 4 ways) and with an NRU replacement algorithm [42]. We noticed that NRU is the best of a set of simple replacement algorithms (see Section V). This new organization offers three key advantages: (1) the total number of entries increases from 16 to 64, (2) hardware complexity decreases (e.g., the number of 13-bit delta comparators is reduced from 128 to 32 per read port), and (3) the miss rate of the delta table becomes lower, holding more IPs, and resulting in improved performance.

IV. Experimental Methodology

A. Experimental setup

We use a modified version of ChampSim [6] a trace-driven simulator used for the 2nd and 3rd Data Prefetching Championships (DPC-2 [2] and DPC-3 [5], master branch, commit: `b44625f`). Recent prefetching proposals [9], [13], [31], [32], [36] are also coded and evaluated in ChampSim. Caches are non-inclusive, although Berto can work similarly with exclusion policies just by bypassing the allocation of memory blocks at the LLC. Table II summarizes our system configuration, mimicking an Intel Alderlake microarchitecture.

B. Energy model

We also report the dynamic energy consumption of the memory hierarchy. We obtain the energy consumption of reads and writes to tag and data arrays at each cache level and DRAM with the CACTI-P [29] model and the Micron DRAM power calculator [3]. Then, we compute the total energy consumption by accounting for the number of accesses of each type across the memory hierarchy. We use 7 nm process technology for our energy calculations.

C. Workloads

We evaluated Berto using a large number of traces from SPEC CPU2017 [41], single-threaded GAP [10] and, CVP [18] benchmark suites. Unless otherwise indicated, we limit our study to memory-intensive traces (MemInt), i.e., those traces that showed at least one miss per kilo-instruction (MPKI) in LLC in our modeled baseline without any hardware prefetcher.

SPEC CPU2017 is one of the most used benchmark sets in academia and industry for measuring CPU performance. We select the 43 MemInt traces out of 95.

GAP is composed of 20 MemInt traces that represent graph workloads.

CVP traces are traces generated by *Qualcomm* for the first Championship Value Predictor (CVP). These AArch64 ARM traces are of special interest to the research community because they cover a wide range of workload types:

1) **CVP-FP and CVP-INT**, are composed of 137 (71 MemInt) and 982 (374 MemInt) float-point and integer compute-intensive traces, respectively. The memory patterns of these traces are similar to those of SPEC CPU2017 traces.
2) **CVP-CRYPTO**. 105 traces representative of cryptography workloads. None of these traces exhibited memory-intensive behavior.
3) **CVP-SRV**. 786 (182 MemInt) traces representative of data center/server-side workloads featuring a large number of different IPs with varying behaviors and cache lines with long-distance reuse.

CVP traces have two characteristics that can hinder the classification of memory instructions, necessary for access pattern detection: they have a larger code footprint and they come from a RISC ISA, with lower entropy in the instruction addresses. The number of IPs contributing to the data footprint is significantly higher in CVP-MemInt traces compared to GAP and SPEC CPU2017 MemInt traces. We measure the number of IPs responsible for 90% of the L1D misses. On average, 143 and 1K IPs account for 90% of the misses in GAP and SPEC, respectively. In contrast, this number increases substantially to 2K, 6K, and 16K in CVP-FP, CVP-Int, and CVP-SRV traces, respectively. This aligns with the observation that server traces generally exhibit a larger footprint [8]. The ARM instructions of the CVP traces have a fixed size, which can decrease the entropy of the addresses of the memory instructions with respect to those of the x86 SPEC and GAP traces. A prefetcher that can track 16 IPs is sufficient to detect the patterns in GAP and in most SPEC CPU2017. However, for CVP-FP traces, we have found it necessary to increase the number of tracked IPs to 64 and use a suitable hash function to reduce aliasing.

For SPEC CPU2017 and GAP traces we warm-up the caches for 50M [37] and collect statistics for the next 200M sim-point instructions. Since CVP traces are smaller, to avoid finishing the trace and starting it again, we warm-up for 25M and simulate the next 50M instructions.

We report performance in terms of IPC improvement (speedup) with respect to an L1D with an IP-stride prefetcher. We use the geometric mean to average the speedups obtained by the traces.

D. Evaluated Prefetching Techniques

We tested the effectiveness of Berto with several high-performance L1D and L1D+L2 prefetchers. As Berto is an L1D prefetcher, we first compare its performance with prefetchers designed for L1D (no prefetching at the L2), and then with multi-level prefetching combinations. The L1D prefetchers are i) Berti[2], the state-of-the-art L1D prefetcher [31], ii) MLOP (3rd place in DPC-3 [36]), an extension

[2]We use the version available for the latest ChampSim Master branch (https://github.com/ChampSim/ChampSim/pull/486)

TABLE II
SIMULATION PARAMETERS OF THE BASELINE SYSTEM.

Core	Out-of-order, TAGE-64KiB [24], 4 GHz with 6-issue width, 12-retire width, 512-entry ROB
TLBs	L1 iTLB: 256 entries, 8-way, 1 cycle L1 dTLB: 96 entries, 6-way, 1 cycle STLB: 2048 entries, 16-way, 7 cycles
MMU Caches	2-entry PSCL5, 4-entry PSCL4, 8-entry PSCL3, 32-entry PSCL2, searched in parallel, one cycle.
L1I	32 KB, 8-way, 3 cycles
L1D	48 KB, 12-way, 4 cycles, with a 24-entry, fully associative IP-stride prefetcher [14]
L2	1.25 MB 20-way associative, 9 cycles, LRU, non-inclusive
LLC	3 MB/core, 12-way, 19 cycles, SRRIP [23], non-inclusive
MSHRs	16/16/48 in L1I/L1D/L2, 64/core at the LLC
DRAM con-troller	One channel/4-cores, 6400 MTPS [15], FR-FCFS, 64-entry RQ and WQ, reads prioritized over writes, write watermark: 7/8th
DRAM chip	4 KB row-buffer per bank, open page, burst length 16, t_{RP}: 12.5 ns, t_{RCD}: 12.5 ns, t_{CAS}: 12.5 ns

TABLE III
CONFIGURATIONS OF EVALUATED PREFETCHERS.

SPP-PPF [13]	256-entry ST, 512-entry 4-way PT, 8-entry GHR, Perceptron weights: 4096×4, 2048×2, 1024×2, and 128×1 entries, 1024-entry prefetch table, 1024-entry reject table
Bingo [9]	2 KB region, 64/128/4K-entry FT/AT/PHT
MLOP [36]	128-entry AMT, 500-update, degree 16
IPCP [32]	128-entry IP table, 8-entry RST table, and 128-entry CSPT table
Berti [31]	8-set 16-way history table, 16-entries 16-deltas delta table
Berto	8-set 16-way history table, 16-set 4-way 16-deltas delta table

of the BOP (DPC-2 winner), and iii) IPCP (DPC-3 winner, published at ISCA 2020 [32]). For multi-level prefetching, we evaluate two state-of-the-art L2 prefetchers along with MLOP and Berti at the L1D: Bingo [9], and SPP-PPF [27]. We also compare with a multi-level IPCP that uses IPCP both at the L1D and L2. The evaluated prefetchers have been briefly described in Section II-A. For all prefetchers, we use a highly tuned implementation as provided by the authors and tune them again for the parameters mentioned in Table II. Table III shows the configurations used for all evaluated prefetchers.

V. EVALUATION

A. Impact of our contributions

Fig. 5 shows the speedup (Y axis, higher is better) of the Berti versions described in Section III. Next, we analyze Berto's sensitivity to some design choices and how these modifications perform on top of the original Berti proposal.

Simplifying latency computation. Replacing the individual measure of latency for each memory miss access (red bar, Computing Latency) with a general DRAM latency counter and searching the whole history table for misses resolved in L2/LLC, does not impact average performance. Still, it reduces memory storage and the number of performed operations.

Hash function. We now compare the performance of Berti's hash function against Berto's hash function (yellow bar, Hash Function). The new hash function gives similar performance in Spec, CVP-FP, and CVP-INT benchmark suites. On CVP-SRV workloads, Berto's hash function improves performance

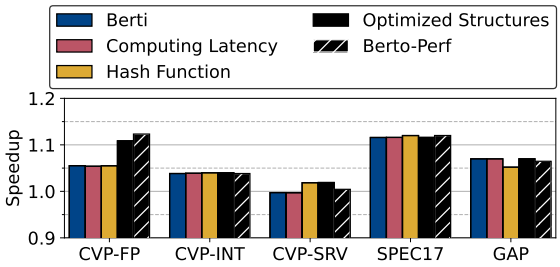

Fig. 5. Speedup over IP-stride of the different Berto modifications averaged across memory-intensive CVP (FP, INT, and SRV), SPEC CPU2017 (FP and INT) and, GAP traces.

TABLE IV
STORAGE OVERHEAD AND PERFORMANCE OF DIFFERENT REPLACEMENT ALGORITHMS WITH 32 AND 64 ENTRIES.

	Storage (Bits)	Speedup 32 entries	Speedup 64 entries
NRU	16	4.5%	4.8%
LRU	160	4.4%	4.7%
SRRIP	32	4.5%	4.8%
FIFO	16	4.1%	4.4%

by 1.4% over Berti's. This is due to distributing better the large number of loads in CVP-SRV workloads.

Tracking more IPs. Replacing the 16-entry fully associative delta table of Berti with a 16-set, 4-way cache-style delta table (black bar, Optimized structures), improves performance by 3.3% over Berti (blue bar) with a more logical and hardware-friendly implementation. Benefits are seen in the CVP suite, due to the large code footprint of its benchmarks.

Sensitivity analysis of the delta table. Increasing the delta table size beyond 64 entries provides no measurable benefit in any of the benchmark suites (not shown in the figure). However, decreasing its size results in a performance drop for the CVP-FP benchmark suite (0.97% decrease from 64 to 32 entries). CVP-INT benchmarks maintain the same performance up to a delta table size of 32 entries, at which point performance starts to decline. For the remaining benchmarks (CVP-SRV, SPEC CPU2017, and GAP), the delta table size can be reduced to 16 entries without any noticeable performance loss.

Replacement algorithm in delta table Our delta table utilizes a Not Recently Used (NRU) replacement policy. We evaluated other common cache replacement policies, including Least Recently Used (LRU), First-In-First-Out (FIFO), and SRRIP. We chose NRU because of its high performance and low implementation overhead in terms of both complexity and storage requirements, as shown in Table IV.

Berto High-Performance We also tested our performance improvements (hash function and optimized delta structure) in Berti without including the modification tailored to reduced complexity related to the latency computation (black bar with white slash). This version obtains performance improvements of 6.4% and 0.4% in the CVP-FP and SPEC CPU17 benchmark suites, respectively, compared to Berti. We can also observe that our latency simplification can be beneficial for some benchmark suites, but detrimental for others.

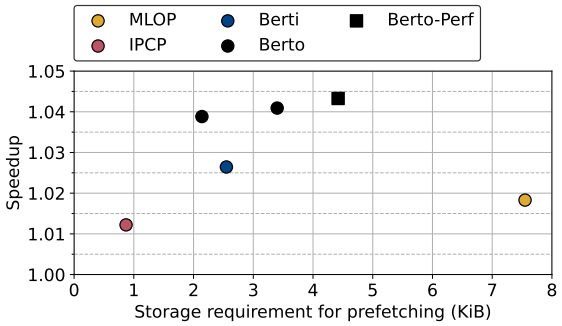

Fig. 6. Speedup vs. storage requirements (log scale). Speedup is normalized to L1D IP-stride and averaged across memory-intensive CVP (FP, INT, and SRV), SPEC CPU2017, and GAP traces.

B. Speedup vs. storage requirements

Fig. 6 summarizes the speedup (Y axis, higher is better) of the evaluated prefetchers with respect to IP-stride for CVP (FP, INT, and SRV), SPEC CPU2017, and GAP traces, along with their storage requirements (X axis, left is better). Berto complexity-effective (black circle) has two different configurations: a low storage overhead configuration with a 32-entry delta table and a standard configuration with a 64-entry delta table. Berto-Perf, i.e., without the simplified latency computation, is shown as a black square.

Berto achieves the highest speedup using only 3.38 KiB of storage. On average, Berto improves performance by 4.0% over IP-Stride and by 1.4% over Berti. In addition, Berto-Perf improves performance with respect to complexity-effective Berto by 0.3% in exchange for a 2.06x increase in storage overhead and more logical complexity. The smallest configuration of Berto (32-entries) achieves better performance than Berti (0.6%) while using 16.1% less storage.

C. Performance of Berto as an L1D Prefetcher

Fig. 7 shows the speedup with respect to IP-Stride achieved by the L1D prefetchers (Y axis) for CVP (FP, INT, and SRV), SPEC CPU2017, and GAP traces (X axis). Berto in the L1D improved or almost equal Berti's performance in all benchmarks suites. Excluding Berto, Berti achieves the best performance in the CVP-SRV, SPEC CPU2017, and GAP benchmark suites. However, on CVP-FP and CVP-INT, its performance decreases by 1.7% (compare to MLOP) and 0.7% (compared to IPCP), respectively. Among the other non-Berti prefetchers, Berto outperforms all other prefetchers across all suites, improving MLOP by 2.7% in CVP-FP, IPCP by 0.7% in CVP-INT, and IP-Stride by 7.8% in GAP.

All L1D prefetchers achieve good speedups for compute intensive benchmark suites (CVP-FP, CVP-INT, and SPEC CPU2017). However, the speedup differences become more significant for other workload types, such as CVP-SRV (server workloads) and GAP (graph workloads). Notably, Berto and Berti are the only prefetchers that do not suffer from performance degradation in CVP-SRV workloads, while MLOP and IPCP have a performance loss of 2.7% and 1.7%, respectively. This trend is also observed in GAP benchmarks, where Berto and Berti outperform IP-Stride by 7.8% and 7.0%, while MLOP and IPCP experience performance degradations of 3.0% and 15.8%, respectively.

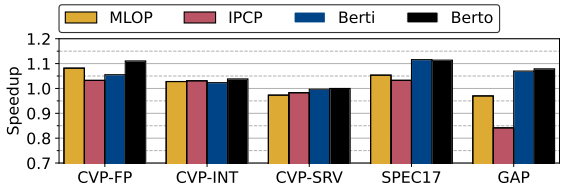

Fig. 7. Speedup of L1D prefetchers compared to a system with L1D IP-Stride.

In the following paragraphs, we analyze the behavior of the L1D prefetchers on a suite-by-suite basis. Fig. 8, 9, 10, 11 and, 12 shows the individual speedup (Y axis) for the 25 traces (20 for GAP workloads) that show the largest performance gaps between the best and worst prefetchers, the geometric mean of the memory intensive traces (GM), and the geometric mean of all traces (GM_ALL) (X axis).

1) CVP-FP: Fig. 8 shows traces with a difference between the worst and the best prefetcher of more than 17.3%. Berto improves performance over IP-Stride in 53 of 71 traces (74.6%) with a maximum of 1.81 (in `compute_fp_101`) and only shows a performance degradation of more than 5% in 6 of 71 traces. MLOP, the second-best prefetcher in the suite, improves performance over IP-Stride in 54 of 71 (76.0%) of the traces (maximum of 1.57 `compute_fp_58`). However, it only loses more than 5% of performance in 3 traces. Berto outperforms Berti in 50 out of 71 traces. We identified that the improved performance of Berto over Berti is due to its ability to track a large number of IPs in its set-associative delta table.

2) CVP-INT: Fig. 9 shows traces with a difference between worst and best prefetcher of more than 14.2%. Berto achieves similar or better results than the other L1D prefetchers in 313 of 374 traces (83.7%). It only shows a performance degradation of more than 5% in 3 traces. Compared to Berti, Berto achieves better performance in 319 of 374 (85.2%) of the traces and only experiences a performance degradation of more than 5% in four traces.

3) CVP-SRV: Fig. 10 shows traces with a difference between the worst and the best prefetcher of more than 8.1%. Unlike the CVP-FP and CVP-INT suites, in CVP-SRV only Berti and Berto do not suffer performance degradation compared to our baseline configuration. CVP-SRV traces exhibit cache block reuse with high distance, e.g, trace `srv105` issue a demand load with address `0xffff0000089dbd1c` every 600 cycles, these patterns are difficult to detect and predict by prefetchers like MLOP, IPCP, Berti, and Berto. However, the confidence mechanism of Berti and Berto prevents triggering incorrect prefetch requests when a pattern is detected with low confidence, unlike MLOP and IPCP. Berto achieves slightly better performance than Berti (1.3% improvement) due to the improved hash function, which reduces aliasing. For example, in trace `srv105`, Berti groups two different IPs into the same entry in the history and delta tables, leading to a learning process that triggers incorrect prefetch requests.

4) SPEC CPU2017: Fig. 11 shows traces with a difference between worst and best prefetcher of more than 8.2%. Berto achieves similar or significantly better results than Berti on all traces except for `602.gcc-1805B`, `603.bwa-(1740B, 2609B)`, `649.fot-(7084B, 10881B, 8225B)`, and

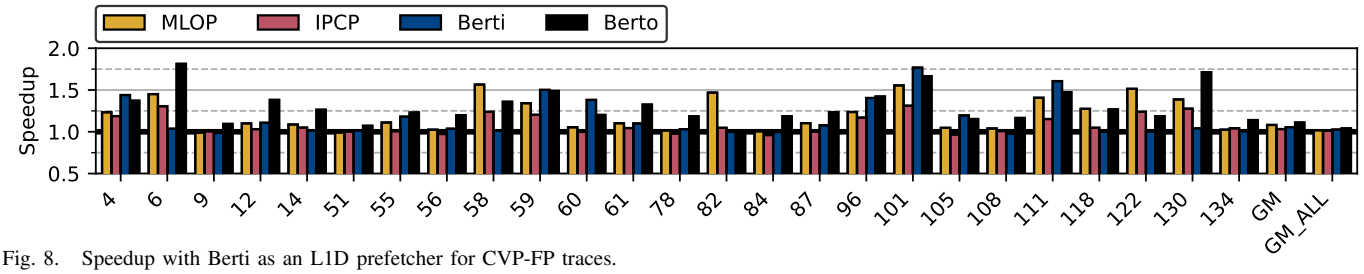

Fig. 8. Speedup with Berti as an L1D prefetcher for CVP-FP traces.

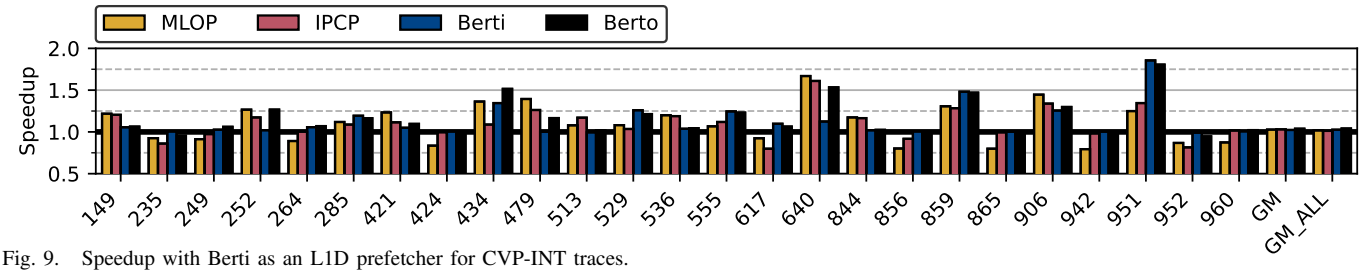

Fig. 9. Speedup with Berti as an L1D prefetcher for CVP-INT traces.

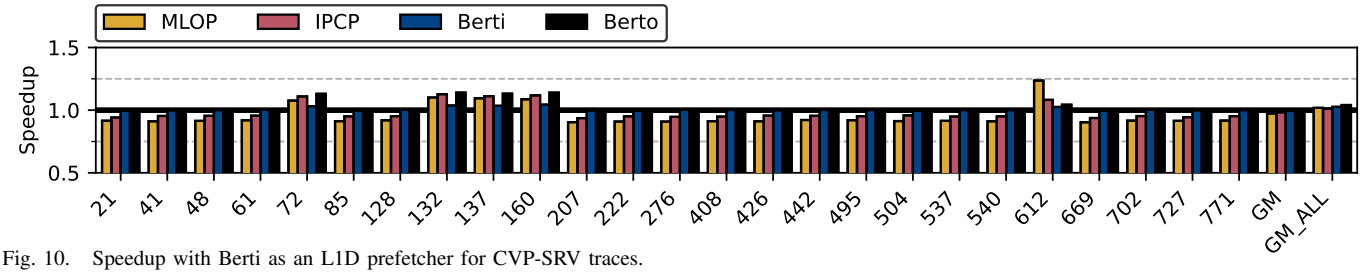

Fig. 10. Speedup with Berti as an L1D prefetcher for CVP-SRV traces.

654.rom-1007B. In some traces, like 602.gcc-1805B, performance degradation can be attributed to not accurately accounting for memory instruction latency. This penalizes the learning process because the deltas identified in the history can be inaccurate, leading to fewer prefetch requests.

On the other hand, for traces such as 649.fot, the new learning process in Berto triggers more prefetching requests. For example, for 649.fot-8225B Berti triggers 7.32 prefetch requests per kiloinstruction vs. 8.30 for Berto; in theory this increase is positive, since it reduces the miss rate in L1D (47.3 MPKI in Berti vs. 40.1 MPKI in Berto), but at the same time it saturates the memory hierarchy more, and in the end performance suffers.

Berti and Berto do not achieve any performance improvement over our baseline configuration in a single benchmark, cactuBSSN, where it can be seen that most memory instructions follow stride patterns. However, there are hundreds of these instructions that execute in an interleaved fashion. Therefore, to track the local behavior of such instructions, Berti would need very large history and delta tables. In contrast, prefetchers that detect patterns in the global address stream do not have this problem, as is the case with MLOP or the IPCP GS class.

5) GAP: Fig. 12 shows all GAP traces. Berto achieves better or similar performance than Berti in all traces except pr-10, pr-5, sssp-3, and sssp-5. On average, Berto improves the performance of Berti by 0.8%.

Excluding Berti, Berto is the best prefetcher for all benchmarks but three (bfs-8, bfs-10 and bfs-14). It consistently achieves similar or better results than IP-stride for all traces except for bfs-10 and pr-10 where it suffer performance degradation of 1.4% and 1.3% respectively, while MLOP and IPCP perform worse than IP-stride in 17 and 28 benchmarks, respectively. In some cases, the IPCP slowdown is very significant, for example, 32.1% in pr-5.

We have selected the application bc-5 to analyze in detail the behavior of prefetchers in GAP. All memory instructions of bc-5 show rather chaotic access patterns, except for one which is very regular. IP-stride, Berti and Berto, by separately tracing the IPs, detect the regular IP pattern and prefetch correctly for it. They do not prefetch for the other IPs. MLOP fails because of the use of a global delta. Accesses issued by IPs with irregular patterns prevent the discovery of a global delta and, therefore, the prefetcher issues very few requests and is not able to prefetch for the regular IP. IPCP detects the delta pattern for the regular IP through its CPLX component, and prefetches correctly for it. However, the GS component generates many useless prefetch requests that drastically decrease the IPCP accuracy and cause a performance loss.

Accuracy. Fig. 13 shows the accuracy (Y axis, higher is better) of the L1D prefetchers (X axis). Berto achieves an accuracy over 75.0% in all benchmark suites (CVP-FP, CVP-INT, SPEC CPU2017, and GAP) except for CVP-SRV, where all prefetcher accuracies are lower than 50.0%. Berto, with a less complex hardware mechanism to measure latency, only loses on average 3.4% accuracy against Berti in CVP-FP, CVP-

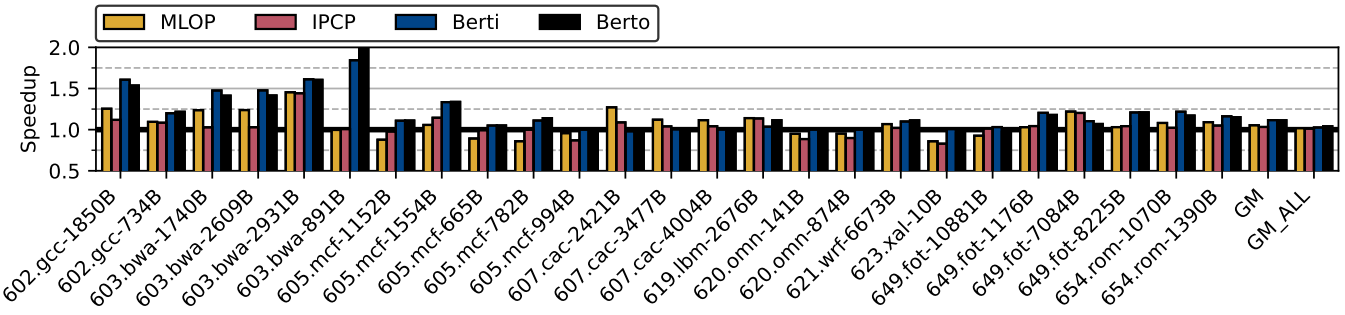

Fig. 11. Speedup with Berti as an L1D prefetcher for SPEC CPU2017 traces.

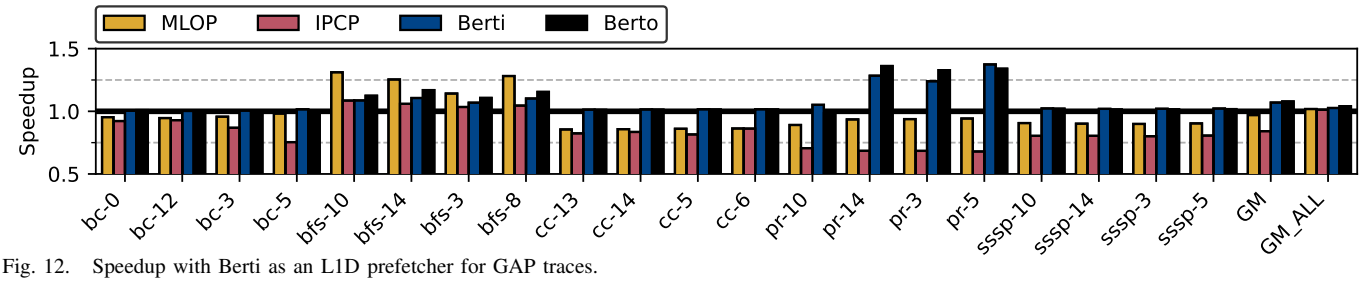

Fig. 12. Speedup with Berti as an L1D prefetcher for GAP traces.

INT, SPEC CPU2017, and GAP, with a maximum loss of 6.6% observed in SPEC CPU2017. The new IP hash function effectively reduces aliasing in CVP-SRV, improving accuracy from 37.0% for Berti to 45.2% for Berto.

MLOP and IPCP shows lower accuracy than Berti and, Berto in all suites. Only on CVP-FP and SPEC CPU2017 their accuracy is similar (73.2% and 71.6% for MLOP and 81.5% and 70.7% for IPCP, respectively) to our proposal. The effectiveness of IPCP is driven by the performance of several tiny prefetchers: a global stream prefetcher (GS class), a constant stride prefetcher (CS class), and a complex stride prefetcher (CPLX class) that work in tandem. For regular access patterns, the CS prefetcher provides high accuracy. However, for complex access patterns, the effectiveness of the CPLX prefetcher is low, with an accuracy of 52.7% and 9.8% for SPEC CPU2017 [41] and GAP [10], respectively.

MLOP, like Berti and Berto, is based on the detection of the best timely deltas. However, it achieves much lower accuracy. The improvement of Berti and Berto over MLOP is mainly due to two factors: i) MLOP uses global deltas for the whole application, while Berti detects different deltas for each IP. Benchmarks like `mcf` generate different delta patterns for each IP, thus using a global delta results in suboptimal performance. ii) Berti uses a strict policy to decide which deltas to use to issue prefetch requests to L1D, while MLOP generates prefetch requests for the best delta with each lookahead regardless of its confidence.

Timeliness. The slashed part of each bar in Fig. 13 represents the prefetch requests whose retrieved data arrive late to L1D. Almost all prefetch requests generated by Berto and Berti are timely. MLOP and IPCP have more than 10% of late prefetchers for CVP-FP, SPEC CPU2017, and GAP benchmarks (MLOP only). Berto achieves similar timeliness to Berti, and despite its simpler mechanism, it is able to discard entries from the history that would produce late deltas. IPCP

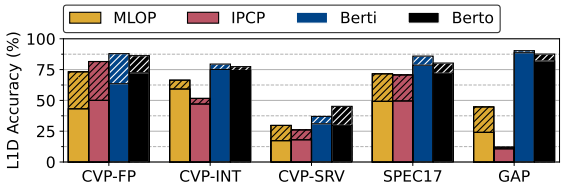

Fig. 13. Prefetch accuracy at the L1D. Percentages of useful requests are broken down into timely (non-slashed) and late (slashed) prefetch requests.

does not use any mechanism to adapt the prefetch requests timing to the miss latency, while MLOP, Berti and Berto do. However, Berti and Berto achieves better timeliness than MLOP due to specific and timely deltas for each IP.

Coverage. Fig. 14 shows demand misses per kilo instructions (MPKI) at the L1D, L2, and LLC (Y axis) with and without L1D prefetching for every benchmark suite (X axis). Berto achieves a lower MPKI at all cache levels for all suites except GAP. Specifically, the percentage reductions in MPKI for Berto relative to Berti at L1D, L2 and LLC are as follows: CVP-FP (25.6%, 26.7%, 23.7%), CVP-INT (13.0%, 5.1%, 4.8%), CVP-SRV (4.0%, 15.4%, 23.5%) and, SPEC CPU2017 (2.8%, 1.8%, 1.7%). In contrast, Berti improves the coverage in GAP, reducing MPKI by the following percentages: 23.8%, 2.0%, and 4.7% in L1D, L2, and LLC, respectively. This is because counting cycles enables Berti to achieve a more accurate learning process for IP. The similar performance is explained by the reduced pressure applied to the memory hierarchy due to fewer prefetch requests.

D. Energy efficiency

Fig. 15 shows the average dynamic energy consumption in the memory hierarchy (L1D, L2, LLC, and DRAM) normalized to no prefetching. Berto consumes more energy than Berti for all benchmarks except GAP, with a maximum increase in CVP-FP due to the extra prefetch requests it triggers. This

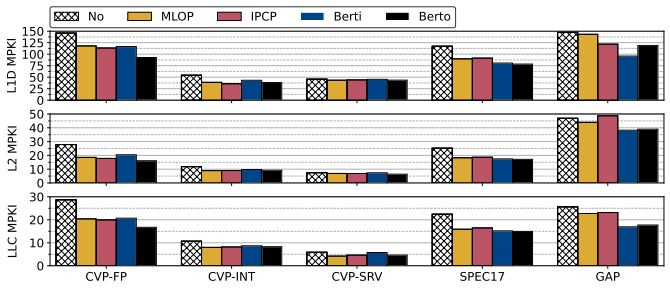

Fig. 14. Prefetch coverage in terms of average L1D, L2, and LLC demand MPKI for all L1D prefetchers.

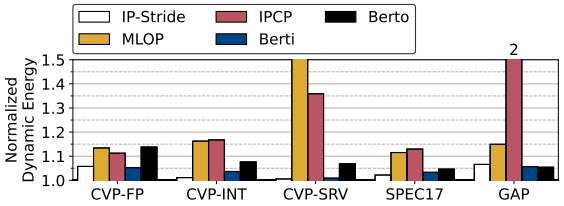

Fig. 15. Dynamic energy consumption in the memory hierarchy normalized to no-prefetching.

additional energy consumption aligns with the performance improvement of Berto over Berti and it comes from the extra L1D accesses. When considering other L1D prefetchers, the average energy consumption is lower than MLOP and IPCP across all benchmarks except CVP-FP (specifically, 0.3% and 2.3% higher than MLOP and IPCP, respectively).

E. Multi-level prefetching performance

Fig. 16 shows the speedup achieved (Y axis) with the multi-level prefetching combinations compared to a system with IP-Stride.

We combine the Bingo and SPP-PPF L2 prefetchers with MLOP, Berti, and Berto L1D prefetchers. For IPCP, we use its configuration as a two-level prefetcher. These multi-level prefetching combinations offer a significant performance boost for CVP-FP and SPEC CPU2017 traces. The best combinations of L1D+L2 prefetchers use Bingo on L2. Bingo adds 19.0%, 17.0% and 12.4% performance increases to systems with only MLOP, Berti and Berto in L1D, respectively for CVP-FP. However, Berto+Bingo has a storage requirement that is 65.3 times higher. For the other suites, Berto alone achieves the same or slightly lower performance compared to any other L1D+L2 prefetcher combination. IPCP at both L1D and L2 (IPCP-IPCP), with a hardware budget comparable to Berto, achieves a significantly lower speedup than Berto alone in L1D, especially in GAP (26.5%).

Coverage. Fig. 17 shows demand MPKI at the L1D, L2 and LLC (Y axis) for the multilevel prefetching combinations. MPKI decreases in both L2 and LLC when adding prefetchers to the L2 cache for most of the L1 prefetchers and benchmark suites. We only observed increases in MPKI for CVP-SRV (in L2 when adding SPP PPF) and for GAP (when using IPCP in L2 and when adding a prefetcher in L2 using Berto or Berti in L1). Adding a prefetcher at the L2 cache level can effectively reduce MPKI for both Berti and Berto prefetchers, with a maximum improvement of 28.5% and 23.9% for L2

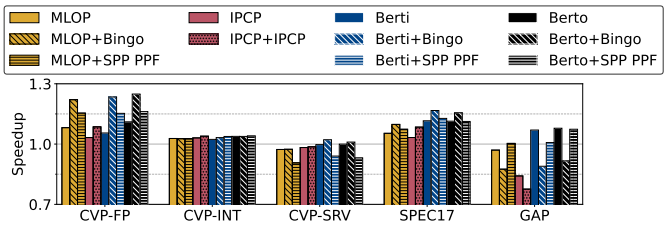

Fig. 16. Speedup with multi-level prefetching normalized to L1D IP-stride.

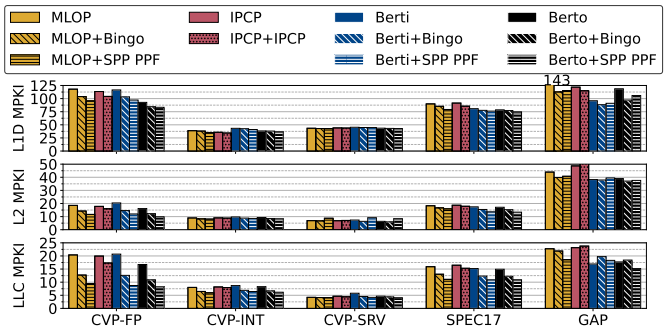

Fig. 17. Prefetch coverage in terms of average L1D, L2 and LLC demand MPKI with multi-level prefetching.

MPKI in the CVP-FP suite using Berti and Berto with Bingo, respectively.

With respect to the L1D MPKI, it decreases significantly when any prefetcher is added to L2 for CVP-FP, SPEC CPU2017 and GAP. A prefetch request issued by the L2 prefetcher brings a block into the L2 cache. When the L1D prefetcher issues a request for the same block, it is found in L2 instead of the LLC or main memory. This decreases the number of late prefetch requests. For example, with MLOP in CVP-FP, the percentage of late prefetches decreases from 30.0% to 17.0% when using PPF in L2. In the case of Berti and Berto, the reduced latency seen by the prefetchers enables the identification of more deltas in the history, which translates to higher coverage. GAP is the only benchmark suite where a lower L1D MPKI does not translate into a higher speedup. This behavior is explained by the increased traffic introduced by the use of L2 prefetchers, which increases the L1D miss latency by 39.9% when using Berti with Bingo compared to standalone Berti.

VI. Related Work

In Section V we presented a quantitative comparison of Berto with recent hardware prefetching techniques [9], [13], [27], [30]–[32], [36]. In this Section we compare other relevant prefetching techniques qualitatively.

Temporal prefetchers. Temporal prefetchers track the temporal order of cache-line accesses (and not the deltas) [20], [22], [25], [39], [43]. Temporal prefetchers usually demand hundreds of KBs of storage, which requires the storage of prefetch metadata in the off-chip memory. Some of the recent works on temporal prefetching are in the pursuit of improving the storage overhead without affecting the prefetch coverage [44], [45].

Spatial prefetchers. Compared to temporal prefetchers, spatial prefetchers are lightweight in terms of storage overhead

and usually learn memory access patterns within a small spatial region of a few KB. Conventional prefetchers such as stride [16] and stream [19], [40]are already being used on commercial processors. Timely Stride prefetching improves the timeliness of conventional stride prefetchers [46]. However, it does not provide better prefetch coverage compared to state-of-the-art L1D and L2 prefetching techniques. Spatial prefetchers like Spatial Memory Streaming (SMS) [40] (similar to Bingo) usually learn single repeating deltas or bit patterns within a spatial region, where a set bit denotes a cache line that should be prefetched. All these techniques do not consider prefetch timeliness.

Kill the program counter (KPC) proposes a holistic cache replacement and prefetching framework [28]. However, the prefetching technique is similar to SPP, with similar performance improvements as SPP. Multi-level adaptive prefetching based on performance gradient tracking [34] (3rd place in DPC-1 [1]) is one of the first proposals that achieves a correlation between IPs and delta sequences. DSPatch [12] tunes a hardware prefetcher based on the available DRAM bandwidth and selects memory access patterns based on prefetch accuracy (if the available DRAM bandwidth is low) and prefetch coverage (if the available DRAM bandwidth is high). Overall, SPP-PPF performs marginally better than SPP+DSPatch.

Prefetch filters and throttling mechanisms. Similar to PPF [13] and DSPatch [12], there are proposals that control the aggressiveness of prefetchers by controlling its prefetch degree and distance, or decides whether to prefetch into the L2 or to the LLC [7], [17], [33] These techniques incur additional storage and perform well for conventional prefetchers with low prefetch accuracy. However, with Berto, the accuracy is significantly higher than prior prefetching techniques, and the implicit confidence mechanism acts like a prefetch throttler.

VII. Conclusions

We propose Berto, an L1D prefetcher with less hardware complexity than Berti. We demonstrate that Berto can learn various memory access patterns while maintaining or increasing the high accuracy and coverage of Berti. We quantify the effectiveness of Berto on the CVP-FP, CVP-INT, CVP-SRV, SPEC CPU2017, and GAP workloads. On average, Berto outperforms state-of-the-art L1D and L2 prefetchers in CVP and SPEC CPU2017 and only lag behind Berti on GAP. In summary, Berto achieves similar or better prefetch accuracy, timely prefetching, and good coverage while offering a performance improvement over Berti with simplified logic.

Acknowledgements

This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Berti-Chip, GA No 101158023, ECHO, GA No.819134), by the MCIN/AEI/10.13039/501100011033/ and the "ERDF A way of making Europe", EU (grants PID2022-136315OB-I00, PID2022-136454NB-C22, RTI2018-098156-B-C53), by the MCIN/AEI/10.13039/501100011033/ the European Union NextGenerationEU/PRTR (grant TED2021-130233B-C33), and by Government of Aragón (T58_23R research group). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] "The 1st data prefetching championship (dpc-1)," Feb. 2009. [Online]. Available: https://jilp.org/dpc/
[2] "The 2nd data prefetching championship (dpc-2)," Jun. 2015. [Online]. Available: https://comparch-conf.gatech.edu/dpc2/
[3] "Micron dram power calculator," Dec. 2015. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf
[4] "The First Championship Value Prediction," https://www.microarch.org/cvp1/cvp1/index.htm, Jun. 2018.
[5] "The 3rd data prefetching championship (dpc-3)," Jun. 2019. [Online]. Available: https://dpc3.compas.cs.stonybrook.edu/
[6] "ChampSim simulator," May 2020. [Online]. Available: http://github.com/ChampSim/ChampSim
[7] J. Albericio, R. Gran, P. Ibáñez, V. Viñals, and J. M. Llabería, "Abs: A low-cost adaptive controller for prefetching in a banked shared last-level cache," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 19:1–19:20, Jan. 2012.
[8] G. Ayers, N. P. Nagendra, D. I. August, H. K. Cho, S. Kanev, C. Kozyrakis, T. Krishnamurthy, H. Litz, T. J. Moseley, and P. Ranganathan, "Asmdb: Understanding and mitigating front-end stalls in warehouse-scale computers," in *46th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2019, pp. 462–473.
[9] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *25th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2019, pp. 399–411.
[10] S. Beamer, K. Asanović, and D. A. Patterson, "The GAP benchmark suite," *CoRR*, vol. abs/1508.03619, Aug. 2015.
[11] R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in *54th Int'l Symp. on Microarchitecture (MICRO)*, Oct. 2021, pp. 1121–1137.
[12] R. Bera, A. V. Nori, , O. Mutlu, and S. Subramoney, "Dspatch: Dual spatial pattern prefetcher," in *52nd Int'l Symp. on Microarchitecture (MICRO)*, Oct. 2019, pp. 531–544.
[13] E. Bhatia, G. Chacon, S. H. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *46th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2019, pp. 1–13.
[14] Y. Chen, L. Pei, and T. E. Carlson, "Leaking control flow information via the hardware prefetcher," *CoRR*, vol. abs/2109.00474, Sep. 2021.
[15] DDR, "DDR standards." [Online]. Available: https://en.wikipedia.org/wiki/Double_data_rate
[16] J. Doweck, "Inside intel core microarchitecture and smart memory access," in *Intel whitepaper*, 2006.
[17] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated control of multiple prefetchers in multi-core systems," in *42nd Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009, pp. 316–326.
[18] J. Feliu, A. Perais, D. A. Jiménez, and A. Ros, "Rebasing microarchitectural research with industry traces," in *Int'l Symp. on Workload Characterization (IISWC)*. IEEE, 2023, pp. 100–114.
[19] B. Grayson, J. Rupley, G. D. Zuraski, E. Quinnell, D. A. Jiménez, T. Nakra, P. Kitchin, R. Hensley, E. Brekelbaum, V. Sinha, and A. Ghiya, "Evolution of the samsung exynos cpu microarchitecture," in *47th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2020, pp. 40–51.
[20] Z. Hu, M. Martonosi, and S. Kaxiras, "Tcp: Tag correlating prefetchers," in *9th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2003, pp. 317–326.
[21] A. Jain, "Exploiting long-term behavior for improved memory system performance," Ph.D. dissertation, The University of Texas at Austin, May 2016.
[22] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *46th Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2013, pp. 247–259.
[23] A. Jaleel, K. B. Theobald, S. C. S. Jr., and J. S. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *37th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2010, pp. 60–71.

[24] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *7th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Jan. 2001, pp. 197–206.

[25] D. Joseph and D. Grunwald, "Prefetching using markov predictors," in *24th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 1997, pp. 252–263.

[26] N. S. Kalani and B. Panda, "Instruction criticality based energy-efficient hardware data prefetching," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 146–149, 2021.

[27] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *49th Int'l Symp. on Microarchitecture (MICRO)*, Oct. 2016, pp. 60:1–60:12.

[28] J. Kim, E. Teran, P. V. Gratz, D. A. Jiménez, S. H. Pugsley, and C. Wilkerson, "Kill the program counter: Reconstructing program behavior in the processor cache hierarchy," in *22nd Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, Apr. 2017, pp. 737–749.

[29] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *2011 Int'l Conf. on Computer-Aided Design (ICCAD)*, Nov. 2011, pp. 694–701.

[30] P. Michaud, "Best-offset hardware prefetching," in *22nd Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Mar. 2016, pp. 469–480.

[31] A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibáñez, V. Viñals-Yúfera, and A. Ros, "Berti: an accurate local-delta data prefetcher," in *55th Int'l Symp. on Microarchitecture (MICRO)*, Oct. 2022, pp. 975–991.

[32] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *47th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2020, pp. 118–131.

[33] B. Panda, "SPAC: A synergistic prefetcher aggressiveness controller for multi-core systems," *IEEE Transactions on Computers (TC)*, vol. 65, no. 12, pp. 3740–3753, Dec. 2016.

[34] L. M. Ramos, J. L. Briz, P. E. Ibáñez, and V. Viñals, "Multi-level adaptive prefetching based on performance gradient tracking," *The Journal of Instruction-Level Parallelism*, vol. 13, pp. 1–14, Jan. 2011.

[35] A. Ros and A. Jimborean, "A cost-effective entangling prefetcher for instructions," in *47th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2021, pp. 99–111.

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

[37] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *10th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, Oct. 2002, pp. 45–57.

[38] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *48th Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2015, pp. 141–152.

[39] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," in *36th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2009, pp. 69–80.

[40] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *33rd Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2006, pp. 252–263.

[41] Standard Performance Evaluation Corporation, "SPEC CPU2017," 2017. [Online]. Available: http://www.spec.org/cpu2017

[42] I. Sun Microsystems, "Ultrasparc t2 supplement to the ultrasparc architecture 2007," Draft D1.4.3, 2007.

[43] D. A. Varkey, B. Panda, and M. Mutyam, "RCTP: Region correlated temporal prefetcher," in *35th Int'l Conf. on Computer Design (ICCD)*, Nov. 2017, pp. 73–80.

[44] H. Wu, K. Nathella, J. Pusdesris, D. Sunwoo, A. Jain, and C. Lin, "Temporal prefetching without the off-chip metadata," in *52nd Int'l Symp. on Microarchitecture (MICRO)*, Oct. 2019, pp. 996–1008.

[45] H. Wu, K. Nathella, D. Sunwoo, A. Jain, and C. Lin, "Efficient metadata management for irregular data prefetching," in *46th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2019, pp. 449–461.

[46] H. Zhu, Y. Chen, and X.-H. Sun, "Timing local streams: improving timeliness in data prefetching," in *24th Int'l Conf. on Supercomputing (ICS)*, Jun. 2010, pp. 169–178.

Agustín Navarro Torres is a postdoctoral researcher at the Computer Architecture and Parallel Systems Group (CAPS) in the University of Murcia. Prior to this, he received his Ph.D. in computer sciences from the Universidad de Zaragoza (2023). His research interests include processor microarchitecture, data hardware prefetching, secure hardware prefetching, and real hardware characterization.

Biswabandan Panda is a member of faculty at IIT Bombay, India. Biswa's well-known contributions are the state-of-the-art high-performing cache compressors and multi-level hardware data prefetchers. Biswa is one of the recipients of the Qualcomm India Faculty Award 2022, Google India Research Award 2022, Prof. Krithi Ramamritam Award for creative research 2023, and Qualcomm Faculty Award 2024.

Jesús Alastruey-Benedé is a Telecommunication Engineer and holds a PhD in Computer Science from the Universidad de Zaragoza (UNIZAR, 1997 and 2009). He is an associate professor in the Computer Science and Systems Engineering Department, UNIZAR. He is a member of the Computer Architecture research group at UNIZAR (gaZ), within the framework of the Aragón Institute for Engineering Research (I3A). His research interests include processor microarchitecture, memory hierarchy, and high-performance computing applications.

Pablo Ibáñez received the MS degree in computer science from the Universitat Politécnica de Catalunya, Spain, in 1989, and the PhD degree in computer science from the Universidad de Zaragoza, Spain, in 1998. He is an associate professor with the Computer Science and Systems Engineering Department, Universidad de Zaragoza. His research interests include processor microarchitecture, memory hierarchy, parallel computer architecture, and high performance computing applications.

Víctor Viñals-Yúfera is a Telecommunications Engineer and holds a PhD in Computer Science from the Universitat Politècnica de Catalunya (UPC, 1982 and 1987). He is currently Professor of Computer Architecture and Technology in the Department of Computer Science and Systems Engineering at the Universidad de Zaragoza. He leads the Computer Architecture research group at the Universidad de Zaragoza (gaZ) since its inception in 1998. His research interests include processor design, performance-oriented and real-time cache memory hierarchy, including network-on-chip, high-performance programming for parallel architectures and energy-saving techniques for multiprocessor chips. He is a member of the IEEE and the European HiPEAC network. He is also President, since June 2024, of the Spanish Society of Computer Architecture (SARTECO). He has supervised 10 theses and has been principal investigator in 6 consecutive National Plan projects in Spain. He has published more than 100 contributions in prestigious journals and conferences.

Alberto Ros is full professor in the Computer Engineering Department at the University of Murcia, Spain. He received Ph.D. in computer science from the University of Murcia in 2009. He received an European Research Council Consolidator Grant in 2018 to improve the performance of multi-core architectures. Working on cache coherence, memory consistency, and processor microarchitecture, he has co-authored more than 100 peer-reviewed articles. He has been inducted into the ISCA and MICRO Hall of Fame. He is IEEE Senior member.