Appears in IEEE Computer Architecture Letters*

Harnessing Pairwise-Correlating Data Prefetching with Runahead Metadata

Fatemeh Golshan, Mohammad Bakhshalipour, *Student Member, IEEE,* Mehran Shakerinava, Ali Ansari, Pejman Lotfi-Kamran, *Member, IEEE,* and Hamid Sarbazi-Azad

Abstract—Recent research revisits pairwise-correlating data prefetching due to its extremely low overhead. Pairwise-correlating data prefetching, however, cannot accurately detect where data streams end. As a result, pairwise-correlating data prefetchers either expose low accuracy or they lose timeliness when they are performing multi-degree prefetching. In this work, we propose a novel technique to detect where data streams end and hence, control the multi-degree prefetching in the context of pairwise-correlated prefetchers. The key idea is to have a separate metadata table that operates one step ahead of the main metadata table. This way, the runahead metadata table harnesses the degree of prefetching by allowing/disallowing the main metadata table to issue prefetch requests. We evaluate our proposal in the context of a four-core chip multiprocessor and show that it significantly reduces erroneous prefetches, providing up to 16.1% performance improvement on top of a state-of-the-art pairwise-correlating prefetcher.

Index Terms—Cache Memory, Performance, Data Prefetching, Pairwise-Correlating Data Prefetching, Overprediction.

1 Introduction

Data prefetchers leverage the predictability of memory access patterns and proactively fetch pieces of data that are not in on-chip caches, thereby hiding the long latency of memory accesses and improving execution performance [1], [2], [3].

Traditionally, the ability of data prefetchers in enhancing performance was the single major metric at evaluating data prefetchers. As such, data prefetchers have grown in their performance enhancements, while other factors like imposed overheads have been marginalized. However, with the widespread use of multi- and many-core processors, and accordingly, the movement towards *lean* cores [4], [5], computer architects re-design virtually every component, considering low-overhead as a major design constraint. One of the components that recently has been contemplated for simplification is the data prefetcher.

Recent research [6], [7], [8], [9], [10], [11], [12], [13] advocates of using low-overhead prefetchers, even if they offer slightly lower performance as compared to high-performance but extremely-high–overhead prefetcher designs. The research towards *lean data prefetchers* has culminated in revisiting delta-based pairwise-correlating data prefetching.

In contrast to streaming prefetchers [14], [15], [16], that store the whole observed address streams next to each other in a FIFO buffer, or footprint-based prefetchers [17], [18], [19], [20], that store bit-vectors for all observed accesses over large memory regions, pairwise-correlating prefetchers correlate every *event* with a *single next prediction*. The event can be some relevant information about the memory accesses, like the last observed address, the last observed delta, or even a combination of several pieces of information like the last three observed deltas. The next prediction, also, can be the next expected address, with *address-based pairwise-correlating prefetchers* [21], [22], [23], [24], [25] or the next expected delta, with *delta-based pairwise-correlating prefetchers* [8], [9]. Figure 1 illustrates how prefetcher metadata is organized with different prefetching methods.

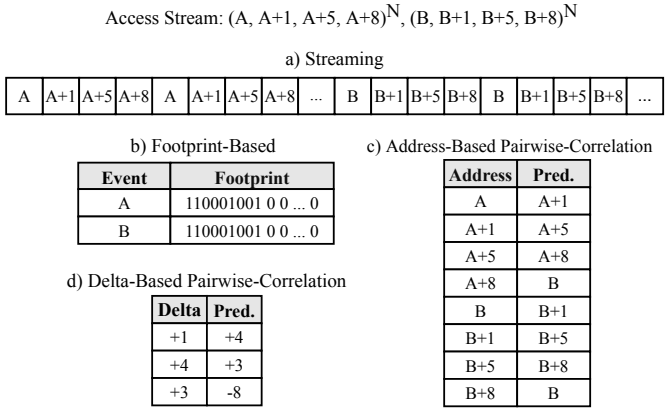

Fig. 1: Comparison of streaming, footprint-based, and pairwise-correlating prefetchers.

Pairwise-correlating prefetching, in fact, is not a new concept in data prefetching literature; it goes way back to the 90s when the simple, preliminary data prefetchers were initially proposed. However, the turnaround from large, high-overhead streaming and footprint-based prefetching [14], [15], [17], [18] to simple delta-based[1] pairwise-correlating prefetching [8], [9] is a new trend motivated by the movement towards multi- and many-core systems, where architectural components from hardware optimizers like prefetchers to cores themselves need to be as simple and low-overhead as possible [4], [5].

- *F. Golshan, M. Shakerinava, and A. Ansari are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. E-mail: {fgolshan, mshakerinava, aansari}@ce.sharif.edu.*
- *M. Bakhshalipour was with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran while doing this work. He is currently affiliated with Carnegie Mellon University (CMU). E-mail: m.bakhshalipour@gmail.com.*
- *P. Lotfi-Kamran is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. E-mail: plotfi@ipm.ir.*
- *H. Sarbazi-Azad is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran and also with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. E-mail: azad@{sharif.edu, ipm.ir}.*

(Corresponding author: Mohammad Bakhshalipour.)

1. All the discussions and techniques we propose in this paper are applicable to all pairwise-correlating prefetchers regardless of being address-based or delta-based. However, in this paper, in line with all recent state-of-the-art work, we narrow down our focus to delta-based pairwise-correlating prefetchers because they are extremely low-overhead and hence, more suitable for modern multi- and many-core systems.

Organizing metadata in a delta-based pairwise-correlated manner leads to significantly less storage requirement for the prefetcher to offer a considerable performance improvement. State-of-the-art streaming prefetchers [14], [16] require megabytes of metadata; state-of-the-art footprint-based prefetchers [18], [19] require tens of or over a hundred kilobytes of metadata; state-of-the-art delta-based pairwise-correlating prefetchers [8], [9] require several hundreds of bytes or a few kilobytes of metadata. The main reason why delta-based pairwise-correlating prefetchers impose less storage overhead is the "desirable fusion" of metadata in these methods. I.e., an observed access pattern that would be otherwise captured in several correlation entries is captured in a single pairwise correlation entry. For example, consider the same sequence of accesses shown in Figure 1, taking that 'A' and 'B' each falls into a separate 4 KB memory page. Such an access pattern is likely to happen when the program loops over two objects of the same class, where each object is allocated in a different page [26]. With a 48-bit physical address space and 64-byte cache blocks (6-bit cache block offset), for such an access pattern, a streaming prefetcher needs to store the all observed addresses, resulting in $N \times 4 \times 42 + N \times 4 \times 42 = 336N$ bits storage requirement (excluding the overhead of indexing the history [14]). With a footprint-based prefetcher, two large bit-vector (each 64-bits) plus the corresponding page tags (each 36-bits) are stored, resulting in 200 bits storage overhead. With a delta-based pairwise-correlating prefetcher, the prefetcher simply stores the correlation between three 16-bit delta pairs, resulting in only 48 bits of storage requirement. Thanks to this extremely-low storage overhead, delta-based pairwise-correlating prefetchers gain increasingly more attention in the literature [8], [9].

While fusing metadata results in less storage overhead, it causes loss of useful information on the other hand. An important item of information that gets lost with pairwise-correlating is *lookahead* information. Unlike streaming prefetchers [14], [15], [16] that prefetch multiple data addresses that follow the correlated address in the FIFO history buffer, or footprint-based prefetchers [17], [18], [19] that prefetch multiple data addresses whose corresponding bit in the bit-vector is set, pairwise-correlating prefetchers have the information of *only one* next expected address or delta per correlation entry. With this lookahead limitation, pairwise-correlating prefetching could face *timeliness* as a major problem, in that, issuing merely a single prefetch request every time may not result in prefetch requests that cover the *whole* latency of cache misses. Therefore, pairwise-correlating prefetchers struggle to issue multiple prefetch requests at once (i.e., *multi-degree* prefetching); nevertheless, the lack of sufficient lookahead information makes this task particularly challenging.

Initial work [27], [28] suggested storing *several* next predictions (e.g., three next addresses that are expected to follow the current address) in order to perform multi-degree prefetching. However, the inefficiency of these methods was shown by later research [15], [29]. The main reason why storing a certain number of next predictions is not efficient is that the length of different correlated address/delta streams is quite *dissimilar*, ranging from a couple to hundreds of thousands [29]. Hence, storing a certain number of predictions, say, N, would either result in storage-inefficiency or limited miss coverage. In detail, if the length of a stream is smaller than N, say M, the $N-M$ entries stored in the metadata storage is useless, resulting in storage-inefficiency; otherwise, if the length of a stream is greater than N, say K, the $K-N$ addresses in the stream would not be captured.

What is typically employed in state-of-the-art pairwise-correlating data prefetchers as the de facto mechanism, including delta-based [9] and address-based [25] ones, and even similar instruction prefetchers [30], is *using the prediction as input to the metadata tables to make more predictions*: whenever a prediction is made, the prefetcher assumes it a correct prediction, and repeatedly indexes the metadata table with the prediction to make more predictions. While this approach has no storage overhead, it offers poor accuracy, as explicitly shown by recent work [8], [16], [19], [30]. The problem is that the prefetcher has no information about *how many times it should repeat this process*. In other words, the mechanism is not able to detect when a data stream ends and cease the prefetching for the stream accordingly. In fact, this emanates again from dissimilar stream lengths: if we repeat this process N times, for streams whose length is smaller than N, say M, we overprefetch $N-M$ addresses, resulting in inaccuracy; for streams longer than N, we may lose timeliness.

Prior approaches that perform multi-degree prefetching in such a way, typically choose the degree of prefetching empirically based on a set of studied workloads. For example, Shevgoor et al. [9] set the degree to four; Bakhshalipour et al. [25] set it to three. These numbers are chosen completely experimentally for a specific configuration and by examining a limited number of workloads, with which, the chosen number provides a reasonable trade-off between accuracy and timeliness. Obviously, limiting the degree to a certain predefined number neither is a solution that scales to various configurations and workloads, nor is optimal (w.r.t accuracy and timeliness) for the examined very configuration/workloads because the length of streams, not only across applications but also across different address streams of the same application, conspicuously varies [29]; that is why prior work [8], [16], [19], [30] observes severe inaccuracy for these prefetching methods.

Recently, Kim et al. [8] proposed a method, named *SPP*, to *harness* multi-degree prefetching in the context of delta-based pairwise-correlating prefetching. SPP predicts prefetching *confidence* and throttles the prefetcher if the prediction is smaller than a constant predefined threshold (chosen empirically). While such a mechanism might be useful for harnessing the prefetcher, it causes the performance of the prefetcher to become increasingly dependent on the accuracy of throttling decisions [10], [19], [20]. That is, when the confidence prediction is not accurate, the prefetcher either loses accuracy due to overprefetching or loses miss coverage (and timeliness) because of inopportunely stopping prefetching. Moreover, enforcing constant predefined thresholds, that are chosen empirically based on evaluating a limited set of configurations and workloads, reduces the flexibility of this at scaling beyond its evaluated scope.

In this work, we propose a novel solution to harness the multi-degree prefetching in the context of pairwise-correlating prefetchers. The key idea is to *have separate metadata information for predicting the next but one expected event (e.g., the delta following the next delta; two deltas away from now)*. This way, in fact, we employ two separate metadata tables: one predicts the next event (DISTANCE1; D1), the other predicts the next but one event (DISTANCE2; D2), which we call *Runahead Metadata Table*. When issuing multi-degree prefetching, the first prefetching is issued using only D1. For issuing the second prefetch, D1 is searched using its previous prediction, similar to multi-degree prefetching of previous methods; meanwhile, D2 is searched using the actual input (not prediction); the prefetch request is issued only if the prediction of both tables match; otherwise, the prefetching is finished. From the third prefetch request (if any) onward, both tables are searched using the corresponding inputs from the previous steps; if their predictions match, the prefetch request is issued and the process continues; otherwise, the prefetching is finished, concluding that the stream has come to an end.

2 THE PROPOSAL

We propose *Runahead MetaData (RMD)*, a general method for harnessing pairwise-correlating prefetching. The main component of *RMD* is an additional D2 table, namely *Runahead Metadata Table*, which runs a step (e.g., one delta) ahead of the conventional

D1 table. More specifically, D2 predicts what address/delta will happen next but one (two steps away from now); in contrast, D1 (the conventional metadata table of pairwise-correlating prefetching) predicts what address/delta will happen next (one step away from now).

The reason for adding D2 is to *harness* the multi-degree prefetching of D1: until when the recursive lookups should resume? As D2 operates one step ahead of D1, what D2 offers is what D1 is expected to offer in the next step. Hence, when D1's second prediction (i.e., prediction using the previous prediction as input) is *not* equal with D2's prediction, we intuitively conclude that the stream has been finished, and do not further issue prefetch requests for the current stream. However, as long as the predictions match, we continue prefetching to provide efficient timeliness, while preserving accuracy[2].

Note that this mechanism is fundamentally different from confidence-based methods like SPP [8]. *RMD*, unlike confidence-based methods, does not assign probability numbers to correlation entries, and hence, does not admit/deny prefetching probabilistically. Instead, it relies on the effectiveness of the underlying prefetcher and ceases prefetching when it determines an end-of-stream.

Using Figure 2, we epitomize how *RMD* works. First off, the entries in tables are interpreted in this way: $\langle A, B \rangle$ in D1 shows that immediately after A, we expect B to happen; $\langle C, J \rangle$ in D2 is intended to mean that two steps away from C, we expect J to happen. Consider that A happens. We index D1 by A. The prediction of D1 is B; we issue the first prefetch request for B. Then, we index D1 by B; meanwhile, we index D2 by A. The predictions of both D1 and D2 are C; their predictions match, and we prefetch C. Then, we index D1 by C and D2 by B. The prediction of D1 is D, and the prediction of D2 is P; their predictions do not match, and we no longer issue prefetch requests.

D1 Table

Delta	Pred.
A	B
B	C
C	D

D2 Table

Delta	Pred.
A	C
B	P
C	J

Fig. 2: An illustration of how the proposed method works.

Note that *RMD* is a general technique and is not limited to any specific pairwise-correlating prefetcher. It can be used in the context of either address-based or delta-based pairwise-correlating prefetchers. Moreover, *RMD* makes no assumption about the underlying metadata tables of a prefetching method, e.g., what is stored in the table, how the table gets updated, the replacement policy of the table, etc. It just adds a new copy of the main metadata table of the prefetcher (D1), and trains and uses it to predict the next but one event. For example, with Variable Length Delta Prefetcher (VLDP) [9], the D1 itself is three tables each indexed by a different length of history. With *RMD* on top of VLDP, the D2 likewise would be three tables.

3 Evaluation

3.1 Methodology

We use ChampSim [31] to simulate a system whose configuration is shown in Table 1. The primary target of *RMD* and similar methods is multi- and many-core systems where the resources such as area and memory bandwidth are scarce and accordingly, hardware optimizers like data prefetchers should be low-overhead and bandwidth-efficient. Having said that, in this paper, we evaluate *RMD* in the context of a system with four computational cores and show that even in such a system, *RMD* is still effective and superior to competing methods. Noteworthy, we expect larger superiorness from *RMD* in the context of systems with more computational cores.

We use SPEC 2006 benchmark [32] for our evaluations. For single-core experiments, we report the results for 15 memory-intensive programs; we create MIX workloads of these programs for our multi-core evaluations. The details of the simulated processor are shown in Table 1. The memory channel is modeled based on borrowed data from commercial DDR4-2133 technology specification [33], which provides peak bandwidth of around 17064 MB/s. We run the simulations for at least 100 M instructions on every core and use the first 20 M as the warm-up and the rest for actual measurements. All data prefetchers are implemented near L1 data cache.

TABLE 1: Evaluation parameters.

Parameter	Value
Processor	4 cores, 8-wide OoO
L1-I/D	32 KB, 8-way set-associative, 1-cycle load-to-use, 64 B lines
L2 Cache	1 MB per core, 16-way set-associative, unified, 11-cycle access latency
Memory	DDR4-2133 MHz, 2 ranks/channel, 8 banks/rank, 2 KB row buffer/bank, tCL-tRCD-tRP-tRAS = 15-15-15-39

We compare the following approaches:

Best-Offset Prefetcher: *BOP* [7] is a state-of-the-art offset prefetcher [2], that tries to find certain offsets, prefetching with which would result in better timeliness. The storage overhead of this method is 1.85 KB.

Variable Length Delta Prefetcher: *VLDP* [9] is a state-of-the-art delta-based pairwise-correlating prefetcher. *VLDP* operates on multiple lengths of history to efficiently learn and prefetch irregular delta patterns. The storage overhead of this method is 998 bytes.

Signature Path Prefetcher: *SPP* [8] is a state-of-the-art method that targets the same goal our proposal does; nonetheless, in a different manner. *SPP* monitors the accuracy of pairwise-correlating prefetcher and adjusts the prefetching degree, considering both accuracy and timeliness of prefetch requests. The storage overhead of this method is 5.37 KB. We use $T_P = 0.5$ and $T_F = 0.8$ as the throttling parameters since this setting provides higher performance improvements as compared to the parameters chosen in the original paper.

Runahead MetaData: *RMD* is our proposal for harnessing the multi-degree prefetching in the context of pairwise-correlating prefetchers. While there are many prefetching methods that can potentially benefit from *RMD*, in this paper, we evaluate our proposal on top of *VLDP* (named *VLDP+RMD*) and *SPP* (named *SPP+RMD*). For *VLDP+RMD*, we make a copy of Delta Prediction Table and train/use it as the runahead metadata. The overall storage overhead is 1.6 KB. For *SPP+RMD*, we remove the baseline throttling mechanism (set the thresholds to zero), then make a copy of Pattern Table as the runahead metadata. The overall storage overhead is 8.38 KB.

3.2 Results

Figure 3 shows miss coverage and overpredictions (i.e., inaccurate prefetches normalized to baseline cache misses [2]) of competing

2. Note that this may result in *livelock* for some particular access patterns. To prevent livelock, we limit the number of outstanding prefetches to a predefined large number (in this paper, 8).

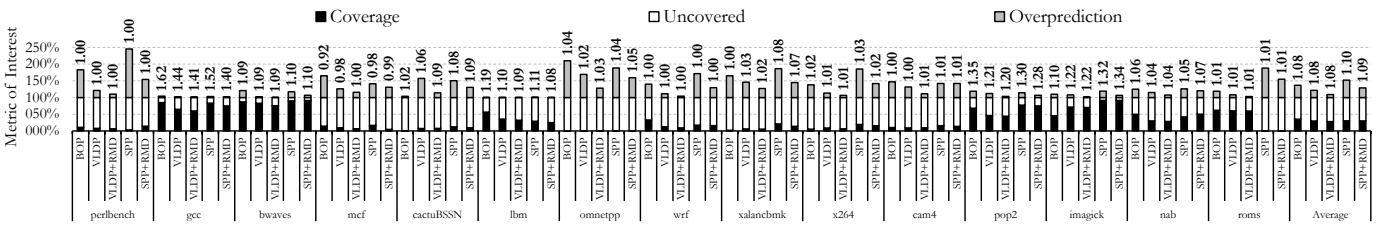

Fig. 3: The results of single-core experiments.

methods. Moreover, numbers on the bars indicate the performance of methods, normalized to a baseline system with no prefetcher.

As Figure 3 shows, *RMD* significantly reduces the overpredictions of both *VLDP* and *SPP*. The reduction for *VLDP* is 1.4*x* on average and up to 3.6*x*; and for *SPP*, it is 77.5% on average and up to 1.9*x*. These large reductions in overpredictions result in substantially less memory bandwidth consumption and cache pollution, which are crucially important for performance and energy-efficiency especially in multi- and many-core substrates. While offering large reductions in mispredictions, *RMD* reduces the miss coverage negligibly; with *RMD*, *VLDP* and *SPP* offer only 1.9% and 0.3% lower miss coverage as compared to without *RMD*. The performance of methods with *RMD* is on par with that of without it. Note that, single-core substrates typically are *not* bandwidth-limited [34], [35], and that is why the reduction in the overprediction does not translate into performance enhancement.

Figure 4 shows the performance of all methods normalized to a baseline with no data prefetcher. The evaluated workloads in this figure are four-core MIX workloads, where MIX_i is created by picking programs $(i, i+1, i+2, i+3)\%15$ (modulo 15) from Figure 3. As the results show, augmenting *VLDP* and *SPP* by *RMD* results in significant performance improvements. The performance improvement of *RMD* on top of *VLDP*/*SPP* is 5.8%/9.6% on average and up to 15%/16.1%.

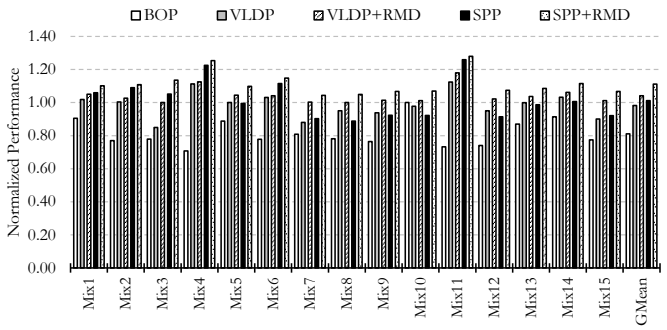

Fig. 4: Performance of competing methods normalized to a baseline with no data prefetcher running multi-core MIX workloads.

4 Conclusion

Pairwise-correlating prefetching, thanks to its extremely small storage requirements, has been recently gaining ever-increasing attention. One major problem of pairwise-correlating prefetching is the lack of sufficient lookahead information, which makes end-of-the-stream detection task particularly challenging, resulting in poor accuracy with prior multi-degree prefetching mechanisms. In this paper, we addressed this issue and proposed a novel technique to harness the prefetching degree in the context of pairwise-correlating prefetching. We evaluated our proposal on top of state-of-the-art methods in the literature and showed that it could largely reduce the overpredictions, resulting in significant performance improvements.

References

[1] B. Falsafi and T. F. Wenisch, "A Primer on Hardware Prefetching," *Synthesis Lectures on Computer Architecture*, 2014.
[2] M. Bakhshalipour *et al.*, "Evaluation of Hardware Data Prefetchers on Server Processors," *ACM CSUR*, 2019.
[3] S. A. Vakil Ghahani *et al.*, "DSM: A Case for Hardware-Assisted Merging of DRAM Rows with Same Content," *POMACS*, 2020.
[4] P. Lotfi-Kamran *et al.*, "Scale-Out Processors," in *ISCA*, 2012.
[5] P. Esmaili-Dokht *et al.*, "Scale-Out Processors & Energy Efficiency," *arXiv preprint arXiv:1808.04864*, 2018.
[6] S. H. Pugsley *et al.*, "Sandbox Prefetching: Safe Run-Time Evaluation of Aggressive Prefetchers," in *HPCA*, 2014.
[7] P. Michaud, "Best-Offset Hardware Prefetching," in *HPCA*, 2016.
[8] J. Kim *et al.*, "Path Confidence Based Lookahead Prefetching," in *MICRO*, 2016.
[9] M. Shevgoor *et al.*, "Efficiently Prefetching Complex Address Patterns," in *MICRO*, 2015.
[10] M. Shakerinava *et al.*, "Multi-Lookahead Offset Prefetching," *The Third Data Prefetching Championship*, 2019.
[11] R. Panda *et al.*, "B-Fetch: Branch Prediction Directed Prefetching for In-Order Processors," *IEEE CAL*, 2011.
[12] A. Ansari *et al.*, "Divide and Conquer Frontend Bottleneck," in *ISCA*, 2020.
[13] A. Ansari *et al.*, "MANA: Microarchitecting an Instruction Prefetcher," *The First Instruction Prefetching Championship*, 2020.
[14] T. F. Wenisch *et al.*, "Temporal Streaming of Shared Memory," in *ISCA*, 2005.
[15] T. F. Wenisch *et al.*, "Practical Off-Chip Meta-Data for Temporal Memory Streaming," in *HPCA*, 2009.
[16] M. Bakhshalipour *et al.*, "Domino Temporal Data Prefetcher," in *HPCA*, 2018.
[17] S. Kumar and C. Wilkerson, "Exploiting Spatial Locality in Data Caches Using Spatial Footprints," in *ISCA*, 1998.
[18] S. Somogyi *et al.*, "Spatial Memory Streaming," in *ISCA*, 2006.
[19] M. Bakhshalipour *et al.*, "Bingo Spatial Data Prefetcher," in *HPCA*, 2019.
[20] M. Bakhshalipour *et al.*, "Accurately and Maximally Prefetching Spatial Data Access Patterns with Bingo," *The Third Data Prefetching Championship*, 2019.
[21] D. Joseph and D. Grunwald, "Prefetching Using Markov Predictors," in *ISCA*, 1997.
[22] Z. Hu *et al.*, "Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior," in *ISCA*, 2002.
[23] Z. Hu *et al.*, "TCP: Tag Correlating Prefetchers," in *HPCA*, 2003.
[24] A.-C. Lai *et al.*, "Dead-Block Prediction & Dead-Block Correlating Prefetchers," in *ISCA*, 2001.
[25] M. Bakhshalipour *et al.*, "An Efficient Temporal Data Prefetcher for L1 Caches," *IEEE CAL*, 2017.
[26] M. Bakhshalipour *et al.*, "Fast Data Delivery for Many-Core Processors," *IEEE TC*, 2018.
[27] Y. Chou, "Low-Cost Epoch-Based Correlation Prefetching for Commercial Applications," in *MICRO*, 2007.
[28] Y. Solihin *et al.*, "Using a User-Level Memory Thread for Correlation Prefetching," in *ISCA*, 2002.
[29] T. F. Wenisch *et al.*, "Temporal Streams in Commercial Server Applications," in *IISWC*, 2008.
[30] L. Spracklen *et al.*, "Effective Instruction Prefetching in Chip Multiprocessors for Modern Commercial Applications," in *HPCA*, 2005.
[31] "ChampSim." https://github.com/ChampSim/.
[32] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Compututer Architecture News*, 2006.
[33] "JEDEC-DDR4." https://www.jedec.org/sites/default/files/docs/JESD79-4.pdf.
[34] M. Bakhshalipour *et al.*, "Reducing Writebacks Through In-Cache Displacement," *TODAES*, 2019.
[35] B. M. Rogers *et al.*, "Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling," in *Proceedings of International Symposium on Computer Architecture*, 2009.