Appears in Proceedings of the 25th International Symposium on High-Performance Computer Architecture (HPCA)⋆

Bingo Spatial Data Prefetcher

Mohammad Bakhshalipour[†‡] Mehran Shakerinava[†] Pejman Lotfi-Kamran[‡] Hamid Sarbazi-Azad[†‡]

[†]*Department of Computer Engineering, Sharif University of Technology*
[‡]*School of Computer Science, Institute for Research in Fundamental Sciences (IPM)*

***Abstract*—Applications extensively use data objects with a regular and fixed layout, which leads to the recurrence of access patterns over memory regions. Spatial data prefetching techniques exploit this phenomenon to prefetch future memory references and hide the long latency of DRAM accesses. While state-of-the-art spatial data prefetchers are effective at reducing the number of data misses, we observe that there is still significant room for improvement. To select an access pattern for prefetching, existing spatial prefetchers associate observed access patterns to either a short event with a high probability of recurrence or a long event with a low probability of recurrence. Consequently, the prefetchers either offer low accuracy or lose significant prediction opportunities.**

We identify that associating the observed spatial patterns to just a single event significantly limits the effectiveness of spatial data prefetchers. In this paper, we make a case for associating the observed spatial patterns to both short and long events to achieve high accuracy while not losing prediction opportunities. We propose Bingo spatial data prefetcher in which short and long events are used to select the best access pattern for prefetching. We propose a storage-efficient design for Bingo in such a way that just one history table is needed to maintain the association between the access patterns and the long and short events. Through a detailed evaluation of a set of big-data applications, we show that Bingo improves system performance by 60% over a baseline with no data prefetcher and 11% over the best-performing prior spatial data prefetcher.

***Keywords*-Big-Data Applications, Memory System, Data Prefetching, Spatial Correlation.**

I. Introduction

The long-latency off-chip memory accesses are a well-known performance bottleneck for many big-data applications. Due to the mismatch between the speed of a processor and the off-chip memory, the processor can easily get stalled for hundreds of cycles upon each DRAM access, losing significant performance potentials. Today's highly-speculative and deeply-pipelined out-of-order processors can, at best, tolerate primary data cache misses, and incur considerable performance penalties upon off-chip memory accesses [1], [2], [3], [4], [5], [6], [7].

Traditionally, processor designers have increased the capacity of on-chip caches to improve the hit ratio and reduce the number of off-chip accesses. Such an approach, however, is less applicable to today's processors as it causes the cache hit latency to increase [3], [8], [9], [10]. Moreover, using the silicon real estate to increase the number of cores is more beneficial than enlarging caches [3], [8], [9]. Finally, the continual growth of the datasets of applications (e.g., graph processing and machine learning) have led to data sets of hundreds of gigabytes or even a few terabytes; several orders of magnitude larger than the largest possible cache on the active die.

System architects have used various tools to bridge the processor-memory performance gap. Data prefetching is one of these tools that has demonstrated great potential for mitigating the latency of cache misses [11], [12], [13]. Data prefetching is the act of predicting future memory accesses and fetching those that are not in the cache before the processor explicitly asks for them, to hide the long delay of off-chip accesses. Nowadays, almost every high-performance processor uses data prefetching (e.g., Intel Xeon Phi [14], AMD Opteron [15], and UltraSPARC III [16]), targeting regular and/or irregular memory access patterns.

Spatial data prefetchers predict future memory accesses by relying on spatial address correlation: the similarity of access patterns among multiple pages[1] of memory. That is, if a program has visited locations $\{X, Y, Z\}$ of Page A, it is likely to touch the $\{X, Y, Z\}$ locations of the same or similar pages in the future. Access patterns demonstrate spatial correlation because applications use data objects with a regular and fixed layout, and accesses reoccur when data structures are traversed [17], [18].

Whenever an application requests a page, spatial data prefetchers (e.g., [18], [19], [20], [21]) observe all accesses to the page, and record a *footprint*, indicating which blocks of the page are used by the application. Then they assign the footprint to an *event* and store the $\langle event, footprint \rangle$ pair in a history table, in order to use the recorded footprint in the future, whenever the event reoccurs. The event is usually extracted from the *trigger access*, i.e., the first access to the page[2]. Upon the recurrence of the same event, spatial prefetchers use the recorded footprint to prefetch future memory references of the currently requested page.

Spatial data prefetchers, as compared to other types of data prefetchers, e.g., temporal data prefetchers [22], [23], [24], [25], [26], [27], [28], require orders of magnitude

[1]Here, a page is a chunk of contiguous cache blocks in the memory, holding several kilobytes of data. Such a page is not necessarily the same as an OS page or a DRAM page, in neither nature nor size.

[2]For example, the Program Counter (PC) of the instruction that requests the page for the first time can be an event to which a footprint is assigned.

⋆ Copyright © 2019 IEEE. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in the proceedings of the 25th International Symposium on High-Performance Computer Architecture (HPCA), pp. 399–411, February 2019 (DOI: 10.1109/HPCA.2019.00053).

less storage for metadata. Moreover, spatial prefetchers can prefetch performance-critical compulsory misses (i.e., unseen cache misses) by generalizing access patterns learned from similar pages to new unobserved pages, thereby significantly improving system performance. Finally, as recent work showed [29], not only do spatial prefetchers increase the performance by decreasing the number of off-chip accesses, but also they improve memory system energy-efficiency by reducing the number of energy-hungry DRAM row activations.

Traditionally, miss coverage, i.e., the fraction of cache misses eliminated by the prefetcher, was the single primary consideration in the design of prefetchers. As such, prefetchers have grown in their ability to eliminate cache misses, while other factors such as storage efficiency and prefetching accuracy have been marginalized. Nonetheless, with the widespread use of multi-core processors, other factors like storage requirement and prefetching accuracy are becoming increasingly important. The storage overhead of hardware optimizers, like prefetchers, should be minimal; otherwise, it might be beneficial to eliminate the optimizer and dedicate its silicon real estate to further increase the number of cores [30]. Prefetching accuracy has also become critical because high core count has driven the designs into the memory bandwidth wall, mainly because of poor pin count scalability [31], [32], [33], [34], [35], [36]. Therefore, prefetchers should be highly accurate to efficiently use the limited bandwidth of DRAM modules [37], [38], [39], [40], [41]. Out of the two, prefetch accuracy is of more importance than storage efficiency as designs usually hit the bandwidth wall first [8], [9], [42], [43].

Inspired by TAGE [44], a state-of-the-art branch predictor, many pieces of recent work improved the efficiency of predictor-based hardware optimizers using *multiple cascaded history tables*[3]. In this strategy, instead of relying on a single history table to predict future events (Figure 1-(a)), several history tables, each with specific information, are used to make predictions (Figure 1-(b)). These tables hold the history of *long* and *short events*. Long events refer to the *coincidence of several specific incidents*. For example, "accessing the 3^{rd} cache block of page P_2 with instruction I_5" may be considered a long event (a coincidence of three incidents). Short events, on the other hand, refer to the *coincidence of few specific incidents*. For example, "execution of instruction I_5" may be considered a short event (just one incident).

Each of the multiple cascaded history tables in TAGE-like predictors stores *the history of events with a specific length* and offers *a prediction of what will happen after the stored event*. The predictions made based on long events are expected to have high accuracy; however, the probability

[3]The TAGE branch predictor itself is inspired by prior work [45] on data compression, which studies predictability limits.

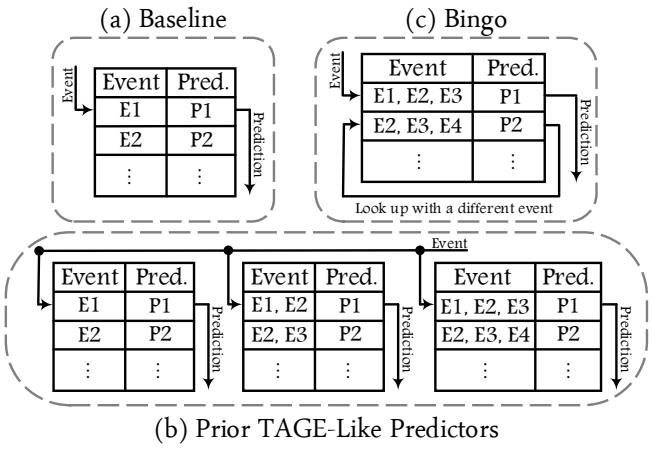

Figure 1. Comparison of proposals for predictor-based hardware optimizers.

of a long event recurring is low. Therefore, if a predictor only relies on a history of long events, it can rarely make a prediction (but when a prediction is made, it is highly accurate). On the contrary, short events have a high chance of recurrence, but predictions based on them are not expected to be as accurate as the predictions based on long events. To get the best of both worlds, TAGE-like predictors record the history of both long and short events. Whenever there is a need for a prediction, they check the history tables, logically one after another; they start from the longest history table (most accurate but least recurring) to make a prediction. If a prediction cannot be made, they switch to the next-longest history table and repeat the process. This process enables the predictor to predict as accurately as possible while not losing the opportunity of making a prediction.

Many pieces of prior work demonstrated great potential by using a TAGE-like strategy to improve the efficiency of various predictor-based hardware optimizers. A TAGE-like strategy is used for branch prediction [46], [47], [48], data prefetching [28], [49], [50], data value prediction [51], [52], [53], memory dependency prediction [54], [55], cache hit/miss prediction [56], quality prediction in approximate computing [57], prediction-based DRAM scheduling [58], and instruction type prediction [59].

In this paper, we leverage this idea in the context of spatial data prefetching and propose Bingo, an efficient mechanism to identify and prefetch spatially-correlated data accesses. Bingo, like prior approaches (e.g., [17], [18], [19], [20], [21]), stores the footprint of each page as the metadata, but unlike them, *associates each footprint to more than one event*. Whenever the time for prefetching comes (i.e., a triggering access occurs), Bingo finds the footprint that is associated with the longest occurred event. As such, Bingo issues accurate prefetch requests without losing prefetch opportunities, proactively supplying the processor with the requested data.

A naive implementation of Bingo requires dedicating

multiple history tables to the prefetcher to keep the metadata. In such an implementation, whenever a footprint needs to be stored, it is inserted into all metadata tables and assigned to events with different lengths in each table. This approach, which has been adopted by prior TAGE-like predictors, imposes significant area overhead, as is evident from Figure 1-(b). We observe that, in the context of spatial data prefetching, a significant fraction of the metadata stored in the cascaded TAGE-like tables is *redundant*. To effectively eliminate the redundancies, we propose an elegant solution to consolidate the metadata of all history tables into a single unified table. With the proposed implementation, *a single history table is looked up multiple times, each time with a different event* to find the prediction associated with the longest event (Figure 1-(c)). By *organizing* the metadata in a single history table, we significantly reduce the storage requirements of Bingo.

In this paper, we make the following contributions:

- We show that relying on a single event to issue prefetch requests is a major source of inefficiency in state-of-the-art spatial data prefetchers.
- We propose a TAGE-like predictor to accurately and maximally extract spatially-correlated data access patterns.
- We suggest a scheme to consolidate multiple history tables into a single unified metadata table, significantly reducing the storage requirements.
- Putting all together, we propose a spatial data prefetcher, named Bingo, and meticulously evaluate it for various big-data applications. We show that our proposal improves the system performance by 60% on average and up to 285% over a baseline with no prefetcher. Meanwhile, it outperforms the best-performing prior spatial data prefetcher by 11% on average and up to 67%.

II. Background

Modern big-data applications have vast datasets that dwarf capacity-limited caches and reside in memory. Therefore, processors executing these workloads encounter frequent cache misses that stall the cores, causing a significant performance loss [3], [4], [28], [34], [60], [61]. Spatial data prefetchers [18], [19], [20], [21], [50], [62], [63], [64], [65], [66] reduce the number of cache misses by prefetching future memory references based on the similarity of access patterns among memory pages.

Spatial data prefetching has long been considered effective due to the unique intrinsic features that it offers. First, spatial prefetchers, as compared to other types of data prefetchers, e.g., temporal data prefetchers [22], [23], [24], [25], [26], [27], [28], require orders of magnitude less storage for keeping metadata information. Unlike temporal prefetchers, spatial prefetchers need to store only an *offset* (i.e., the distance of a block address from the beginning of a page) or a *delta* (i.e., the distance of two consecutive accesses that fall into a page) and are not required to store the full address. As a result, they need significantly less storage for storing the metadata. Moreover, in spatial prefetching, the *order* of accesses inside the page boundaries is of less importance, and as such, it is not required to record intra-page access order, which further reduces the storage requirement. As the memory pages on which spatial prefetchers train and prefetch are usually smaller than or equal to a DRAM row (e.g., 1–2 KB as compared to 2–8 KB), all prefetch requests sent along with the trigger access enjoy a row buffer hit, and hence, all of them are *rapidly* fetched and cached inside the last-level cache (LLC), downplaying the impact of fetch order [29].

Another equally remarkable strength of spatial data prefetchers is their ability to eliminate compulsory cache misses. Compulsory cache misses are a major source of performance degradation in important classes of applications, e.g., scan-dominated workloads, where scanning large volumes of data produces a bulk of unseen memory accesses that cannot be captured by caches [67]. By utilizing the pattern that was observed in a past page to a new unobserved page, spatial prefetchers can alleviate the compulsory cache misses, significantly enhancing system performance.

Finally, an accurate spatial data prefetcher improves not only performance but also memory subsystem energy usage. Spatial data prefetchers have the opportunity to increase the DRAM row buffer hit ratio, by precisely predicting expected-to-be-used cache blocks, and fetching all useful cache blocks in a single DRAM row activation. By doing so, they prevent from multiple energy-hungry DRAM row activations that otherwise would have happened if no spatial prefetcher had been present [29].

We separate prior work on spatial data prefetching into two classes: *Per-Page History* (PPH) and *SHared History* (SHH) methods. PPH refers to approaches that record an access history for each page (usually page footprint), then correlate the recorded history with an event, and finally store the history in a metadata table. On the other hand, SHH points to methods that observe all accesses at the global level, and save the history information (usually deltas) in a shared metadata organization.

The SHH class of spatial prefetchers target storage efficiency as their foremost design consideration. These prefetchers range from straightforward stride prefetchers [68], [69], [70] to more advanced prefetchers with sophisticated heuristics for prefetching [50], [62], [63], [64]. Typically, these prefetchers maintain a single metadata structure to record the patterns observed by all pages. In other words, they do not store a pattern for each page; instead, they fuse the everywhere-observed access patterns into a single unified organization. For example, a delta-based SHH prefetcher (e.g., [50]) might observe three consecutive accesses, say, A_1, A_2, and A_3 in a specific page P_1, and

generate two successive deltas, d_1 and d_2 ($d_1 = A_2 - A_1$ and $d_2 = A_3 - A_2$). In this case, instead of recording that "A_1, A_2, and A_3 were observed in page P_1," it records in a global metadata history that "d_2 follows d_1."

SHH strategy significantly reduces the storage requirements of the prefetcher; however, it also significantly reduces the ability of prefetcher at eliminating cache misses when the prefetcher uses the $\langle d_1, d_2 \rangle$ correlation entry in a page whose behavior is different from P_1. Another important challenge in SHH approaches is the *prefetching degree*: the number of requests that the prefetcher issues at once. In PPH methods, as we discuss shortly, whenever the prefetcher is triggered, it fetches all expected-to-be-used blocks of the page at once as determined by the page footprint. In SHH approaches, however, there is no such wealth of information, and hence, the prefetcher does not know how many prefetches it should issue in order to receive the blocks in a timely manner. SPP [62], an SHH-based method, proposes techniques to adaptively throttle the prefetching degree: a prefetch is only issued if the estimated accuracy of that prediction is above a certain threshold. While such heuristics might be useful at controlling the degree of prefetching, they cause the miss coverage and timeliness of the prefetcher to become increasingly dependent on the accuracy of the throttling decisions.

Another class of spatial prefetching is PPH. Once a page is requested by an application for the first time (i.e., trigger access), PPH approaches start to observe and record later accesses to that page as long as the page is actively used by the application. Whenever the page is no longer utilized (i.e., end of page residency), these approaches associate the recorded access patterns to an event and then store the $\langle event, pattern \rangle$ pair in their history tables.

The recorded history is usually a bit vector, known as the page footprint, in which each bit corresponds to a block of the page: a `'1'` in the footprint indicates that the block has been used during the page residency, while a `'0'` indicates otherwise. The event to which the footprint is associated is usually extracted from the trigger access. For example, Kumar and Wilkerson [17] proposed using the `'PC+Address'` of the trigger access as the event: the `'PC'` of the trigger instruction combined with the `'Address'` that was requested by the trigger instruction. As another case, Somogyi et al. [18] evaluated several heuristics as the event, and empirically found that `'PC+Offset'` performs better than the other ones: the `'PC'` of the trigger instruction combined with `'Offset,'` the distance of the requested cache block from the beginning of the requested page. Later, upon the recurrence of the same event (e.g., for the case of `'PC+Offset,'` an instruction with the same `'PC'` requests a cache block that is in the `'Offset'` distance of a page), these prefetchers apply the stored footprint to predict and prefetch future memory references of the currently requested page.

An associated challenge with these approaches is *finding the best event* to which the footprint of a page should be assigned. Each heuristic has its own advantages and disadvantages. For example, of the two mentioned events, `'PC+Address'` [17] is highly accurate as it *conservatively* waits for the same instruction to be re-executed and the same address to be touched. While accurate, this method is unable to cover compulsory cache misses, because the exact same page should be requested in order for the stored footprint to be used. `'PC+Offset'` [18], on the other hand, is *aggressive* and can cover compulsory misses by applying the footprint information of a page to another, but predictions made based on it are not much accurate. In this paper, we show that *relying merely on one of these heuristics is suboptimal as compared to a mechanism that correlates each footprint with multiple events, and uses the best-matching event for prefetching*.

III. Motivation

Planning to architect a high-performance spatial data prefetcher, we narrow the design space of Bingo to PPH-based approaches. Figure 2 shows the *accuracy* and *match probability* of various heuristics as the event to which the footprints of pages are associated, averaged across all applications[4]. Accuracy is the percentage of all prefetched cache blocks that have been used by the processor before eviction, and match probability is the fraction of events that have been found in the history table.

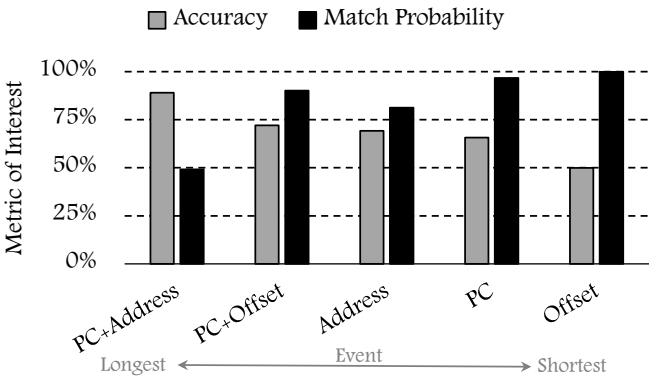

Figure 2. Accuracy and match probability of various heuristics as the event to which access history of pages are associated.

As the event becomes *longer*, the accuracy of predictions increases while the matching probability generally decreases. Among the evaluated heuristics, `'PC+Address'` is the longest event (i.e., the same instruction and the same address should simultaneously happen) which gives the highest prediction accuracy, but with this event, there is less opportunity for prediction as the probability of the event reoccurring is low. Therefore, if the predictor merely relies

[4]See Section V for a complete list of applications.

on this event, its predictions will be accurate, but it will not be able to make a prediction often.

On the other hand, as the events become *shorter*, the accuracy of predictions decreases but the prediction opportunity generally increases. With 'Offset' as the event, which is the shortest event among the evaluated ones (i.e., just the distance of a block from the beginning of the page should reoccur), there is a high opportunity for prediction; but the predictions are not as accurate as those of the longer events. Therefore, if a prefetcher only uses this event, it will often be able to issue prefetch requests, but the prefetches will be unacceptably inaccurate.

This observation motivates a mechanism in which more than one event is used to make predictions. Once a page footprint is recorded, it is *associated with more than one event*, and then stored in the history table. That is, a page footprint is associated with, say, 'PC+Address,' 'PC+Offset,' and 'PC,' and then stored in the history table. Whenever the time for prefetching comes (i.e., a trigger access occurs), the prefetcher looks up the history with the longest event (i.e., 'PC+Address'): if a match is found, the prefetcher issues prefetch requests based on the matched footprint; otherwise, it looks up the history with the next-longest event (i.e., 'PC+Offset'), in a recursive manner. This way, the prefetcher benefits from both high accuracy and high opportunity, overcoming the limitations of previously-proposed spatial prefetchers.

To demonstrate the importance of using more than one event, Figure 3 shows the miss coverage and accuracy of a spatial prefetcher that associates page footprints to multiple events when the number of events varies from one to five. When the number of events is one, the prefetcher always associates page footprints to the longest event (i.e., 'PC+Address'). As the number of events increases, the prefetcher can associate page footprints to shorter events. When the number of events is five, the prefetcher is able to associate page footprints to all events, including the shortest event (i.e., 'Offset').

As shown in Figure 3, increasing the number of events enables the prefetcher to cover more cache misses while maintaining prefetch accuracy. We observe the highest improvement when we go from one event (i.e., 'PC+Address') to two events (i.e., 'PC+Address' and 'PC+Offset'), as there is a significant increase in the miss coverage of the prefetcher. Increasing the number of events beyond two, however, does not result in a major improvement; therefore, for the sake of simplicity, we use two events for the proposed spatial prefetcher, Bingo.

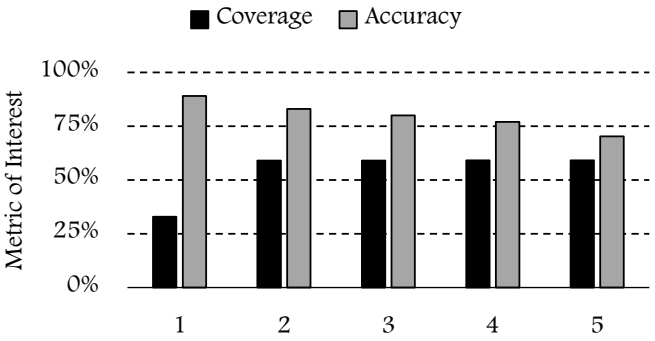

Figure 3. Coverage and accuracy of a TAGE-like prefetcher with varying number of events to which the footprints of pages are associated. When the prefetcher uses one event, the event is PC+Address. As the number of events increases, the shorter events become available to the prefetcher. When the number of events becomes five, all events are available to the prefetcher.

IV. Bingo Spatial Prefetcher

Like prior work [18], Bingo uses a small auxiliary storage to record spatial patterns while the processor accesses spatial regions. Upon an access to a new page (i.e., trigger access), Bingo allocates an entry in its auxiliary storage for the page and starts to record a footprint for it. At the end of the residency of the page (i.e., whenever a block from the page is invalidated or evicted from the cache [18]), Bingo transfers the recorded pattern to its history table and frees the corresponding entry in the auxiliary storage.

Unlike prior work, Bingo uses both 'PC+Address' and 'PC+Offset' events for prefetching. A naive implementation of Bingo requires two distinct history tables, just like prior TAGE-like approaches. One table maintains the history of footprints observed after each 'PC+Address,' while the other keeps the footprint metadata associated with 'PC+Offset.' Upon looking for a pattern to prefetch, logically, first, the 'PC+Address' of the trigger access is used to search the long history table. If a match is found, the corresponding footprint is utilized to issue prefetch requests. Otherwise, the 'PC+Offset' of the trigger access is used to look up the short history table. In case of a match, the footprint metadata of the matched entry will be used for prefetching. If no matching entry is found, no prefetch will be issued.

Such an implementation, however, would impose significant storage overhead. We observe that, in the context of spatial data prefetching, a considerable fraction of the metadata that is stored in the cascaded TAGE-like history tables are *redundant*. By *redundancy*, we mean cases where both metadata tables (tables associated with long and short events) offer the same predictions. Figure 4 shows the redundancy of TAGE-like history tables for spatial prefetching. In this experiment, every time the spatial prefetcher needs to make a prediction, we determine if the long and short events offer the same prediction. As shown, there is considerable redundancy ranging from 26% in *SAT Solver* to 93% in *Mix2*.

To efficiently eliminate redundancies in the metadata storage, instead of using multiple history tables, we propose *having a single history table but look it up multiple times, each time with a different event*. Figure 5 details our practical

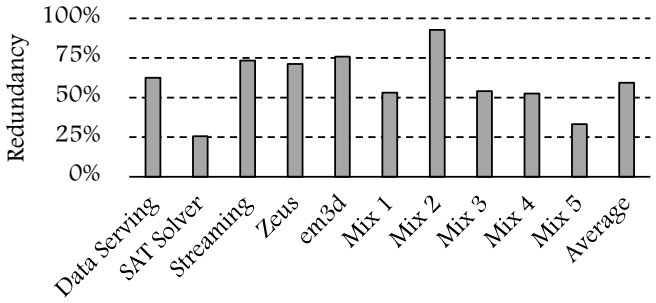

Figure 4. Redundancy in the history metadata of TAGE-like predictors for spatial prefetching. Redundancy is defined as the fraction of lookups for which both long and short events offer an identical prediction.

design for Bingo which uses only one history table. The main idea is based on the fact that *short events are carried in long events*. That is, by having the long event at hand, we can find out what the short events are, just by ignoring parts of the long event. For the case of Bingo, the information of `'PC+Offset'` is carried in `'PC+Address;'` therefore, by knowing the `'PC+Address,'` we also know what the `'PC+Offset'` is[5]. To exploit this phenomenon, we propose having only one history table which *stores just the history of the long event but is looked up with both long and short events*. For the case of Bingo, the history table stores footprints which were observed after each `'PC+Address'` event, but is looked up with both the `'PC+Address'` and `'PC+Offset'` of the trigger access in order to offer high accuracy while not losing prefetching opportunities.

To enable this, we find that the table should only be *indexed with a hash of the shortest event* but *tagged with the longest event*. Whenever a piece of information is going to be stored in the history metadata, it is associated with the longest event, and then stored in the history table. To do so, *the bits corresponding to the shortest event are used for indexing the history table* to find the set in which the metadata should be stored; however, *all bits of the longest event is used to tag the entry*. More specifically, with Bingo, whenever a new footprint is going to be stored in the metadata organization, it is associated with the corresponding `'PC+Address.'` To find a location in the history table for the new entry, a hash of only `'PC+Offset'` is used to index the table[6]. By knowing the set, the baseline replacement algorithm (e.g., LRU) is used to choose a victim to open a room for storing the new entry. After determining the location, the entry is stored in the history table, but all bits of the `'PC+Address'` are used for tagging the entry.

Whenever there is a need for prediction, the history table

[5]This is also true for events that we discarded and did not use for Bingo. All events like `'PC,'` `'Address,'` and `'Offset'` are known when we know `'PC+Address.'` Moreover, this is also the case for other TAGE-like predictors, including the original TAGE branch predictor [44] where multiple history lengths are used to index the metadata tables.

[6]Repeatedly, the bits correspond to `'PC+Offset'` are carried in `'PC+Address.'`

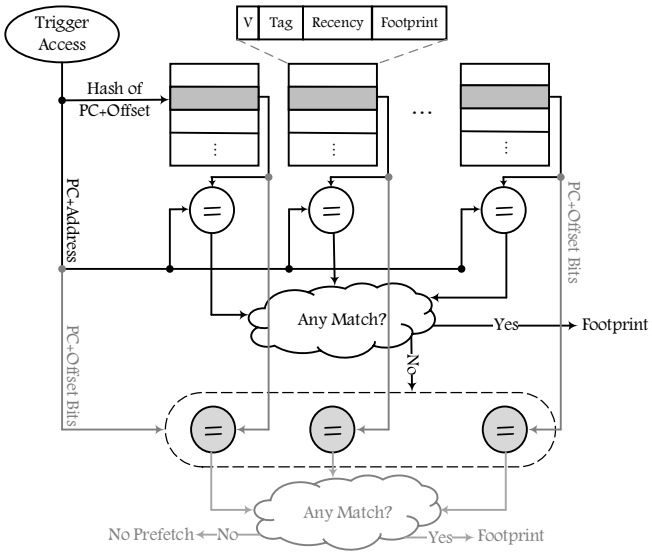

Figure 5. The details of the history table lookup in Bingo prefetcher. Gray parts indicate the case where lookup with long event fails to find a match. Each large rectangle indicates a physical way of the history table.

is first looked up with the longest event; if a match is found, it will be used to make a prediction. Otherwise, the table should be looked up with the next-longest event. As both long and short events are mapped to the same set, there is no need to check a new set; instead, the entries of the same set are tested to find a match with the shorter event.

With Bingo, the table is first looked up with the `'PC+Address'` of the trigger access. If a match is found, the corresponding footprint metadata will be used for issuing prefetch requests. Otherwise, the table should be looked up with the `'PC+Offset'` of the trigger access. As we know both `'PC+Address'` and `'PC+Offset'` are mapped to the same set, there is no need to check a new set. That is, all the corresponding `'PC+Offset'` entries *should be in the same set*. Therefore, we test the entries of the same set to find a match. In this case, however, not all the bits of the stored tags in the entries are necessary to match; only the `'PC+Offset'` bits need to be matched. This way, we associate each footprint with more than one event (i.e., both `'PC+Address'` and `'PC+Offset'`) but store the footprint metadata in the table with only one of them (the longer one) to reduce the storage requirement. Doing so, redundancies are automatically eliminated because a metadata footprint is stored once with its `'PC+Address'` tag.

In the proposed design, whenever the table is looked up with a shorter event, it is possible that more than one match is found. With Bingo, it is possible that none of the entries match the `'PC+Address'` of the trigger access, and in the meantime, more than one entry matches the `'PC+Offset'` of the access. Such cases present a challenge to Bingo in that it should issue prefetch requests based on multi-

ple footprint information that may be dissimilar. Various heuristics can be employed in such cases: e.g., choosing the most recent footprint based on the recency information[7] or issuing prefetch requests for blocks that are indicated in the footprint of all of the matching entries. We evaluated many of such heuristics and empirically found that considering all matching footprint information at issuing prefetch requests gives the best performance: a cache block is prefetched if it is present in the footprint of at least 20% of matching entries.

V. Methodology

We use ChampSim[8] [71], the simulation infrastructure used in the Second Data Prefetching Championship (DPC-2) [72], to meticulously simulate a system whose configuration is shown in Table I. We model a system based on Intel's recent Xeon Processor [73]. The chip has four OoO cores with an 8 MB shared last-level cache (LLC). Two memory channels are used for accessing off-chip DRAM, providing a maximum bandwidth of 37.5 GB/s. The OS uses 4 KB pages and virtual to physical address mapping is accomplished through a random first-touch translation mechanism, enabling long-running simulations [74]. We estimate the delay of the caches using CACTI 7.0 [75]. The cache block size is 64 bytes in the entire memory hierarchy.

Table I
Evaluation parameters.

Parameter	Value
Chip	14 nm, 4 GHz, 4 cores
Cores	4-wide OoO, 256-entry ROB, 64-entry LSQ
Fetch Unit	Perceptron [76], 16-entry pre-dispatch queue
L1-D/I	Split I/D, 64 KB, 8-way, 8-entry MSHR
L2 Cache	8 MB, 16-way, 4 banks, 15-cycle hit latency
Main Memory	60 ns zero-load latency, 37.5 GB/s peak bandwidth

A. Workloads

Table II summarizes the key characteristics of our simulated workloads. In line with the literature [18], [23], [28], [67], we choose several big-data server and scientific applications for evaluation. We also consider five four-core representative mix workloads from a set of memory-intensive SPEC programs [77] whose execution performance is highly sensitive to the latency of memory accesses.

We use the SimFlex [78] methodology to simulate server workloads. For every server application, we create five checkpoints with warmed caches, branch predictors, and prediction tables. Each checkpoint is drawn over an interval of 10 seconds of simulated time as observed by the OS. Then we run 200 K instructions from each checkpoint, using the first 40 K instructions for warming queues (e.g., ROB), and the rest for actual measurements. Based on SimFlex [78], our measurements are computed with 95% confidence and less than 4% error. For SPEC benchmarks, 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 next 80 M for measurements.

Table II
Application parameters.

Application	Description	LLC MPKI
Data Serving	Cassandra Database, 15GB Yahoo! Benchmark	6.7
SAT Solver	Cloud9 Parallel Symbolic Execution Engine	1.7
Streaming	Darwin Streaming Server, 7500 Clients	3.9
Zeus	Zeus Web Server v4.3, 16 K Connections	5.2
em3d	400K Nodes, Degree 2, Span 5, 15% Remote	32.4
Mix 1	lbm, omnetpp, soplex, sphinx3	15.7
Mix 2	lbm, libquantum, sphinx3, zeusmp	12.5
Mix 3	milc, omnetpp, perlbench, soplex	12.7
Mix 4	astar, omnetpp, soplex, tonto	14.7
Mix 5	GemsFDTD, gromacs, omnetpp, soplex	12.6

B. Prefetchers' Configurations

We compare our proposal with state-of-the-art spatial data prefetchers. For every prefetcher, we perform a sensitivity analysis in order to find the storage required to offer reasonable miss coverage in our workload suite. We begin with the configuration suggested in the original work and measure the *average miss coverage* across all workloads. Then we increase the capacity of metadata tables to observe how sensitive the prefetcher is to the allocated storage. If the average miss coverage does not change significantly ($> 5\%$), we allocate the same storage to the prefetcher as stated in the original proposal. Otherwise, we increase the storage of the prefetcher until its average miss coverage plateaus. In what follows, we describe the prefetchers' configurations:

Best Offset Prefetcher: BOP [63] is a recent data prefetcher, as well as the winner of the Second Data Prefetching Championship (DPC-2) [72]. BOP builds upon prior work, namely the Sandbox Prefetcher [64], and attempts to enhance its timeliness. On each access, BOP tests a single offset to determine whether it would have been able to predict the current access. By evaluating various offsets, it attempts to discover offsets that are expected to produce timely prefetches. We evaluate BOP with a 256-entry Recent Requests Table.

Signature Path Prefetcher: SPP [62] is another recent data prefetcher that computes a signature for each delta offset pattern, and estimates the probability of delta predictions for each signature. By utilizing these probabilities, SPP adaptively changes its prefetching

[7]Entries in the history table (just like any other associative structure) store a few replacement bits (e.g., recency) to help choose a victim when the set is full and one entry needs to be evicted (e.g., LRU). Based on this information, we can select the most recent entry among multiple matches.

[8]The source code of our simulator and the implementation of evaluated data prefetchers are available at <https://github.com/bakhshalipour/Bingo>.

degree, putting less bandwidth pressure on the DRAM modules. We evaluate SPP with a 256-entry Signature Table, 512-entry Pattern Table, and 1024-entry Prefetch Filter.

Variable Length Delta Prefetcher: VLDP [50] is a state-of-the-art spatial data prefetcher which uses multiple histories of deltas to predict the stream of accesses in a given page. We simulate VLDP with a 16-entry Delta History Buffer, 64-entry Offset Prediction Table, and three 64-entry Delta Prediction Tables.

Access Map Pattern Matching: AMPM [21] is another state-of-the-art spatial prefetcher and the winner of the First Data Prefetching Championship (DPC-1) [79]. AMPM marks recently accessed cache blocks in a table called the Memory Access Map. Based on the stored information, AMPM detects strided access patterns and then predicts future blocks that will be accessed. We enlarge the Memory Access Map Table to cover the whole capacity of the LLC.

Spatial Memory Streaming: SMS is a powerful spatial data prefetcher and the base of our proposal. SMS associates footprint metadata to the `'PC+Offset'` of the triggering access. SMS has since significantly outgrown its purpose and showed great potential in scopes like spatio-temporal prefetching [67], determining the commodity DRAM fetching granularity [29], and managing die-stacked DRAM caches [36], [80], [81]. We equip SMS with a 16 K-entry 16-way associative history table.

Bingo: Our proposal associates spatial patterns to more than one event and stores all of the patterns in a single unified history table in a capacity-optimized manner. We use a 16-way set-associative structure for the history table and set its capacity based on the sensitivity analysis in Section VI-A.

We study all data prefetchers in the context of the LLC since the fairly large capacity of a multi-megabyte LLC (as compared to primary caches) paves the way for *longer residency* of pages at this level. Whenever pages linger in the cache for a long time, a much larger opportunity is unleashed for different data pieces to be accessed within that page. This enables spatial prefetchers to completely observe the data accesses of each page and precisely learn the intra-page access patterns [28], [36], [81]. We consider every core to have its own prefetcher, independent of others (i.e., no metadata sharing among cores [30]). All methods are triggered upon LLC accesses and prefetch directly into the LLC (i.e., no prefetch buffer [23], [28]).

VI. Evaluation

A. Storage Requirement

The effectiveness of Bingo, like any other data prefetcher, directly depends on the size of the history with which the predictions are made. Figure 6 shows how the miss coverage of Bingo changes when the number of entries dedicated to its history table varies. As the history table becomes larger, the miss coverage also increases because the prefetcher is able to compare the observed event with a distant past history, and hence, the likelihood of a match increases. Beyond 16 K entries, the coverage plateaus, effectively exploiting the available opportunity. Therefore, we decide to devote 16 K entries to the history table of Bingo. With a 16 K-entry history table, the total storage requirement of the prefetcher is 119 KB, accounting for only 6% of the LLC capacity.

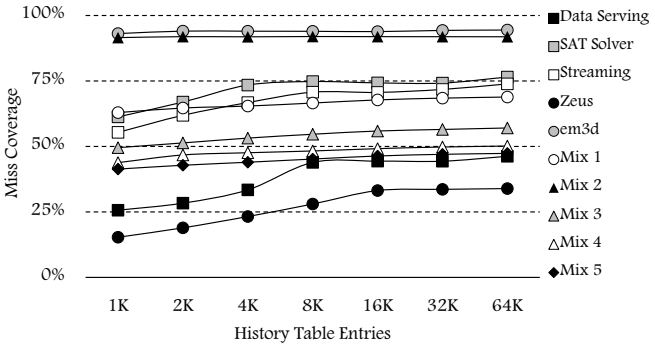

Figure 6. The miss coverage of the proposed data prefetcher as a function of the number of entries in the history table.

B. Miss Coverage & Overpredictions

To evaluate the effectiveness of the proposed prefetcher, Figure 7 shows the coverage and overprediction of all the competing prefetching techniques. Covered misses are the ones that are successfully captured by a prefetcher. Over-predictions are incorrect prefetches, which are normalized to the number of data misses in the baseline system without a prefetcher[9].

As shown, Bingo offers the highest miss coverage across all workloads. By associating footprint metadata to more than one event, and matching the longest event, Bingo maximally and precisely extracts spatially-correlated data access patterns and significantly reduces the number of cache misses. On average, Bingo covers more than 63% of cache misses, outperforming the second-best prefetcher by 8%. The overprediction of Bingo is on par with the rest of the competing prefetchers.

Corroborating many pieces of prior work (e.g., [3], [23], [28]), complex access patterns in the context of modern

[9]Not to be confused with *accuracy* which is referred to the fraction of correct prefetches out of all prefetches.

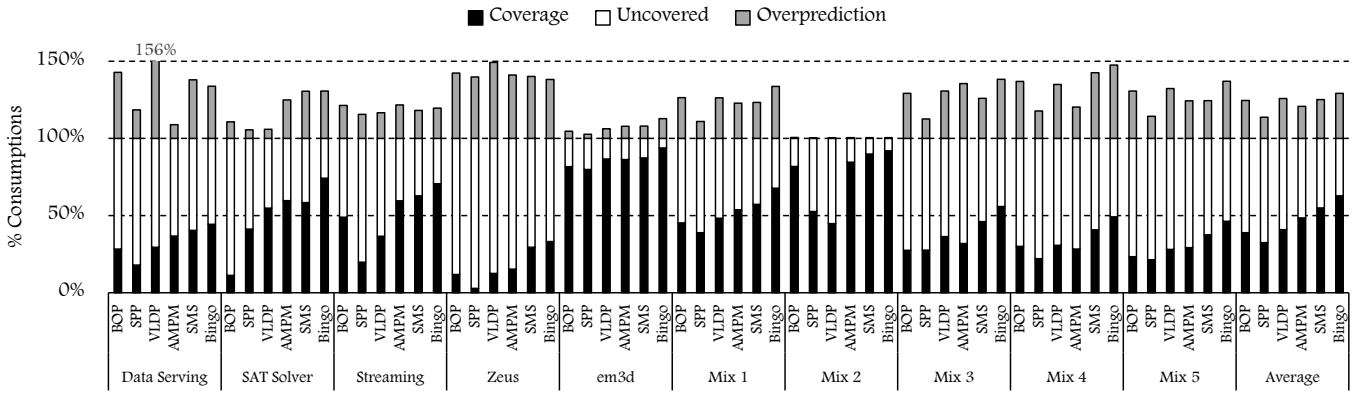

Figure 7. Coverage and overprediction of Bingo as compared to the competing spatial data prefetchers.

server workloads are beyond what can be captured by simple delta-based approaches. Server workloads repeatedly traverse various data structures, resulting in frequent switches among pages (e.g., in a database buffer pool [67]). Whenever various pages are traversed at the same time, many patterns are manifested in a page, which are not necessarily exhibited in other pages; as a consequence, SHH-based spatial prefetchers (e.g., BOP, SPP, VLDP) are faced with significant challenges, which prevent them from offering a high level of miss coverage.

Techniques like what has been proposed in BOP and SPP throttle down the prefetcher when it offers high overpredictions. However, by doing so, they also reduce the miss coverage because prefetch requests, including the correct ones, are *lazily* issued. VLDP uses multiple histories of deltas for prediction, and hence, offers higher miss coverage. However, it suffers from the inefficiencies of its multi-degree prefetching. Upon predicting the next access in a page, VLDP uses the prediction as input to the history tables to make more predictions. We observe that this strategy is inaccurate for server workloads and produces more overpredictions as the prefetching degree increases[10].

By maintaining footprint metadata for each page, AMPM and SMS offer an order of magnitude higher coverage as compared to other prior prefetchers. SMS, however, is more aggressive than AMPM, resulting in higher overprediction and higher miss coverage. SMS correlates footprints with the `'PC+Offset'` of the trigger accesses and applies the learned footprint whenever the same `'PC+Offset'` is observed again. However, as we showed, the `'PC+Offset'` is *not long* enough to offer high accuracy.

C. System Performance

Figure 8 shows the performance improvement of Bingo along with other prefetching techniques, over a baseline without a prefetcher. Bingo consistently outperforms the competing prefetching approaches across all workloads. The performance improvement of Bingo ranges from 11% in *Zeus* to 285% in *em3d*. Miss coverage, timeliness, and the accuracy of prefetches are the main contributors to Bingo's superior performance improvement. For most of the workloads, Bingo provides a significant performance improvement. In *Zeus*, however, memory accesses are more temporally correlated than spatially [28]. Even those accesses that are spatially predictable are already fetched in parallel by the out-of-order processing, resulting in a negligible performance improvement with spatial prefetchers.

As a result of low miss coverage, techniques like BOP and SPP provide lower performance improvement as compared to the other methods, especially on server workloads. VLDP offers higher performance, mainly because of having better miss coverage. However, it falls behind PPH-based methods like AMPM and SMS that keep page footprints and use that to issue more accurate prefetches for all expected-to-be-used cache blocks (cf. Section II). While AMPM and SMS offer high levels of performance improvement, their performance is still significantly less than that of Bingo. By associating footprint metadata to more than one event and matching the longest event, Bingo maximally and accurately extracts spatially-correlated data access patterns and uses them for accurate and timely prefetching.

D. Performance Density

Many pieces of prior work [8], [9], [30], [83] argue that the performance gain of hardware optimizers, like prefetchers, should outweigh their area overhead. Otherwise, using the silicon real estate to further increase the number of cores may be more beneficial. Prior work used a metric called *performance density*, defined as *throughput per unit area*, to quantify how efficiently a design uses the silicon real estate: incorporating the hardware prefetcher into a system is beneficial only if it is able to increase the performance density [8], [30].

[10]This observation was also made by prior work [28], [62] for VLDP in the context of data prefetching. Moreover, a similar observation for the inaccuracy of such a strategy was made by Kolli et al. [82] in the context of instruction prefetching for server workloads.

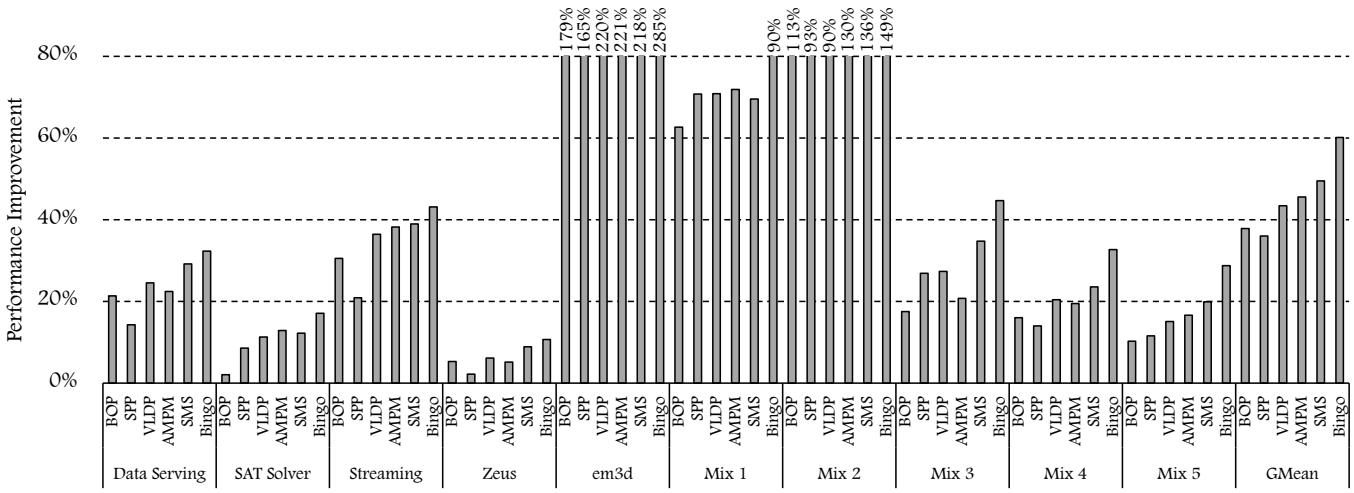

Figure 8. Performance comparison of prefetching techniques, normalized to a baseline system with no prefetcher.

As the area requirement of the evaluated prefetchers are different from one another, to better understand how effectively they use the chip silicon area, we evaluate their performance density. Figure 9 summarizes the results of this study, showing the performance density improvement of all evaluated designs as compared to a baseline system with no prefetcher. For this experiment, we only consider the area of cores, caches, interconnect, and memory channels, neglecting the area of I/O devices.

Bingo improves the performance density of the baseline system by 59%, which is 10% higher than that of the second best-performing method. As compared to other prefetchers, the performance density improvement of Bingo is slightly lower than its performance improvement. The main reason is the larger history table of Bingo in comparison to the metadata table(s) of other approaches. However, the drop in the performance density is insignificant (less than 1%) as the total area overhead of Bingo is less than 6% of the LLC area and a small fraction of the whole chip area. The results indicate that the performance improvement of Bingo far outweighs its storage overhead, making it a practical design for high-performance processors.

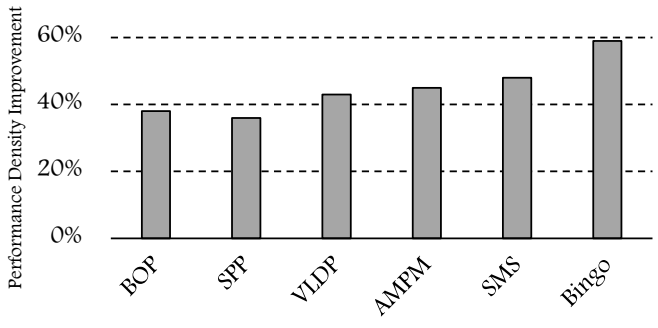

Figure 9. Performance Density improvement of evaluated methods over a baseline system without a prefetcher.

E. ISO-Degree Comparison

Certainly, a significant fraction of the performance improvement of Bingo and other PPH-based methods over SHH-based ones comes from their better timeliness. Bingo, like other PPH-based methods, gathers a footprint for every page and issues prefetch requests for all expected-to-be-used blocks of the page *at once*, resulting in superior timeliness. However, in SHH-based methods, there is no information about the footprint of pages; hence, the methods do not know how many prefetches they should issue whenever they are triggered. Prior work employs various mechanisms in order to enable *multi-degree prefetching* in the context of SHH-based methods. For example, VLDP, upon predicting the next access in the page, uses the prediction as input to the history tables to issue more prefetches. Typically, there is a trade-off between the degree of prefetching and its prediction accuracy: the more aggressive the prefetcher is, the more wrong prefetches are expected. To cope with the overprediction of multi-degree prefetching, prior pieces of work, using various heuristics, attempt to limit the number of multi-degree prefetches. For example, VLDP allows the prefetching degree to be up to four.

In this section, in order to provide an iso-degree comparison, we lift the ban and allow the SHH-based prefetchers to issue further prefetch requests upon each activation. We set the maximum degree of BOP/VLDP to 32 and set the confidence threshold [62] of SPP to 1%. Figure 10 shows the results of this experiment. Due to better timeliness, the performance of SHH-based methods slightly increases; however, their overprediction likewise increases because of many wrong prefetches, resulting in significant off-chip bandwidth pollution. With iso-degree prefetching, Bingo still significantly outperforms all other methods.

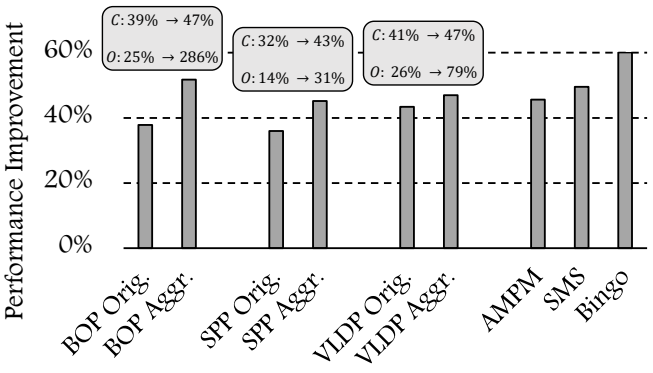

Figure 10. ISO-degree comparison of prefetching methods. 'Orig' indicates the original and so-far–evaluated version of an SHH-based prefetcher; however, 'Aggr' represents the aggressive and high-degree version. Callouts indicate how the coverage and overprediction of prefetchers vary from the original version to the aggressive version: 'C' and 'O' stand for 'Coverage' and 'Overprediction,' respectively.

VII. Related Work

Many pieces of recent work target long latency memory stalls in the context of hardware data prefetchers. IMP [84] identifies and prefetches irregular memory accesses from indirect patterns of the form $A[B[i]]$, which are copious in many applications like graph analytics and sparse linear algebra. TEMPO [85] augments memory controllers to track accesses to page table entries in order to prefetch post-translation accesses. Domino [28] uses a combination of one or two last data misses to find the correct temporal address stream in the history and prefetch the subsequent data misses. B-Fetch [86] utilizes the branch predictor to run ahead of the executing program, thereby prefetching load instructions along the expected future path. A few recent studies [87], [88] also evaluated the use of deep learning algorithms for prediction of memory accesses. None of these methods rely on spatial correlation, and hence, can be used orthogonally with our proposal.

Several approaches use spatial pattern predictors outside the context of data prefetching. SFP [17] uses a predictor based on spatial footprints to recognize useful words of cache blocks and stores such words in a decoupled sectored cache [89]. BuMP [29] determines the DRAM fetch granularity by relying on a spatial footprint predictor that identifies the usage density of the opened DRAM row. Footprint caching [36], [80], [81] proposes to manage the die-stacked DRAM as a page-based cache with multiple-block fetch granularity. Footprint caching approaches employ spatial footprint predictors to fetch and cache only the expected-to-be-used cache blocks of every page, reducing the bandwidth pressure on DRAM modules. Our proposal can be incorporated into such schemes to achieve higher prediction efficiency.

VIII. Conclusion

Long latency off-chip misses, which often stall the processor for the data to arrive, are a major source of performance degradation in big-data applications. Spatial data prefetching is a technique for reducing the number of cache misses or their harmful effect. Spatial data prefetchers exploit the fact that data accesses are spatially correlated over memory regions of several kilobytes, and this correlation is predictable. In this work, we showed that state-of-the-art spatial data prefetchers do not fully take advantage of the existing opportunities because of associating the footprint metadata to just a single event. We proposed a practical approach to associate the footprint metadata to more than one event to improve the coverage and accuracy of spatial prefetching. Moreover, we suggested a general mechanism to eliminate redundancies in the metadata table of history-based predictors. We showed that the proposed spatial prefetcher significantly outperforms the competing spatial prefetchers.

Acknowledgment

We thank the anonymous reviewers for their valuable comments. We appreciate Mark Sutherland from PARSA-EPFL for providing us with the required tools for simulating server workloads in the context of our framework. We thank members of IPM HPC center for maintaining and managing the cluster that is used to carry out the experiments.

References

[1] T. S. Karkhanis and J. E. Smith, "A First-Order Superscalar Processor Model," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 338–349, 2004.

[2] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding Sources of Inefficiency in General-purpose Chips," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 37–47, 2010.

[3] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-Out Workloads on Modern Hardware," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 37–48, 2012.

[4] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Quantifying the Mismatch Between Emerging Scale-Out Applications and Modern Processors," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, pp. 15:1–15:24, Nov. 2012.

[5] A. Mirhosseini, A. Sriraman, and T. F. Wenisch, "Enhancing Server Efficiency in the Face of Killer Microseconds," *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2019.

[6] M. Hashemi, E. Ebrahimi, O. Mutlu, and Y. N. Patt, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 444–455, 2016.

[7] M. Hashemi, O. Mutlu, and Y. N. Patt, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 61:1–61:12, 2016.

[8] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-Out Processors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 500–511, 2012.

[9] P. Esmaili-Dokht, M. Bakhshalipour, B. Khodabandeloo, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Scale-Out Processors & Energy Efficiency," *arXiv preprint arXiv:1808.04864*, 2018.

[10] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 184–195, 2009.

[11] D. Guttman, M. T. Kandemir, M. Arunachalamy, and V. Calina, "Performance and Energy Evaluation of Data Prefetching on Intel Xeon Phi," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 288–297, 2015.

[12] V. Jiménez, R. Gioiosa, F. J. Cazorla, A. Buyuktosunoglu, P. Bose, and F. P. O'Connell, "Making Data Prefetch Smarter: Adaptive Prefetching on POWER7," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 137–146, 2012.

[13] H. Kang and J. L. Wong, "To Hardware Prefetch or Not to Prefetch?: A Virtualized Environment Study and Core Binding Approach," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 357–368, 2013.

[14] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights Landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016.

[15] P. Conway and B. Hughes, "The AMD Opteron Northbridge Architecture," *IEEE Micro*, vol. 27, no. 2, pp. 10–21, 2007.

[16] T. Horel and G. Lauterbach, "UltraSPARC-III: Designing Third-Generation 64-bit Performance," *IEEE Micro*, vol. 19, no. 3, pp. 73–85, 1999.

[17] S. Kumar and C. Wilkerson, "Exploiting Spatial Locality in Data Caches Using Spatial Footprints," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 357–368, 1998.

[18] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial Memory Streaming," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 252–263, 2006.

[19] J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Stealth Prefetching," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 274–282, 2006.

[20] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos, "Accurate and Complexity-Effective Spatial Pattern Prediction," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 276–287, 2004.

[21] Y. Ishii, M. Inaba, and K. Hiraki, "Access Map Pattern Matching for Data Cache Prefetch," in *Proceedings of the International Conference on Supercomputing (ICS)*, pp. 499–500, 2009.

[22] Y. Solihin, J. Lee, and J. Torrellas, "Using a User-Level Memory Thread for Correlation Prefetching," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 171–182, 2002.

[23] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi, "Temporal Streaming of Shared Memory," *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 222–233, 2005.

[24] M. Ferdman and B. Falsafi, "Last-Touch Correlated Data Streaming," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 105–115, 2007.

[25] Y. Chou, "Low-Cost Epoch-Based Correlation Prefetching for Commercial Applications," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 301–313, 2007.

[26] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Practical Off-Chip Meta-Data for Temporal Memory Streaming," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 79–90, 2009.

[27] A. Jain and C. Lin, "Linearizing Irregular Memory Accesses for Improved Correlated Prefetching," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 247–259, 2013.

[28] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Domino Temporal Data Prefetcher," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 131–142, 2018.

[29] S. Volos, J. Picorel, B. Falsafi, and B. Grot, "BuMP: Bulk Memory Access Prediction and Streaming," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 545–557, 2014.

[30] C. Kaynak, B. Grot, and B. Falsafi, "SHIFT: Shared History Instruction Fetch for Lean-Core Server Processors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 272–283, 2013.

[31] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 371–382, 2009.

[32] J. Huh, D. Burger, and S. W. Keckler, "Exploring the Design Space of Future CMPs," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 199–210, 2001.

[33] M. Bakhshalipour, A. Faraji, S. A. Vakil Ghahani, F. Samandi, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Reducing Writebacks Through In-Cache Displacement," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2019.

[34] M. Bakhshalipour, H. Zare, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Die-Stacked DRAM: Memory, Cache, or MemCache?," *arXiv preprint arXiv:1809.08828*, 2018.

[35] H. A. Esfeden, F. Khorasani, H. Jeon, D. Wong, and N. Abu-Ghazaleh, "CORF: Coalescing Operand Register File for GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[36] D. Jevdjic, S. Volos, and B. Falsafi, "Die-Stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 404–415, 2013.

[37] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-Aware Shared Resource Management for Multi-core Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 141–152, 2011.

[38] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated Control of Multiple Prefetchers in Multi-Core Systems," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 316–326, 2009.

[39] E. Ebrahimi, O. Mutlu, and Y. N. Patt, "Techniques for Bandwidth-Efficient Prefetching of Linked Data Structures in Hybrid Prefetching Systems," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 7–17, 2009.

[40] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, "Prefetch-Aware DRAM Controllers," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 200–209, 2008.

[41] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 63–74, 2007.

[42] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward Dark Silicon in Servers," vol. 31, no. 4, pp. 6–15, 2011.

[43] M. Bakhshalipour, P. Lotfi-Kamran, A. Mazloumi, F. Samandi, M. Naderan, M. Modarressi, and H. Sarbazi-Azad, "Fast Data Delivery for Many-Core Processors," *IEEE Transactions on Computers (TC)*, vol. 67, no. 10, pp. 1416–1429, 2018.

[44] A. Seznec, "A Case for (Partially)-Tagged Geometric History Length Predictors," *Journal of Instruction-Level Parallelism (JILP)*, 2006.

[45] J. Cleary and I. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Transactions on Communications (TCOM)*, vol. 32, no. 4, pp. 396–402, 1984.

[46] A. Seznec, "A New Case for the TAGE Branch Predictor," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 117–127, 2011.

[47] A. Seznec, J. San Miguel, and J. Albericio, "The Inner Most Loop Iteration Counter: A New Dimension in Branch History," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 347–357, 2015.

[48] P. Michaud, "An Alternative TAGE-Like Conditional Branch Predictor," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, pp. 30:1–30:23, Aug. 2018.

[49] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "An Efficient Temporal Data Prefetcher for L1 Caches," *IEEE Computer Architecture Letters (CAL)*, vol. 16, no. 2, pp. 99–102, 2017.

[50] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently Prefetching Complex Address Patterns," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 141–152, 2015.

[51] A. Perais and A. Seznec, "Practical Data Value Speculation for Future High-End Processors," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 428–439, 2014.

[52] A. Perais and A. Seznec, "EOLE: Paving the Way for an Effective Implementation of Value Prediction," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 481–492, 2014.

[53] A. Perais and A. Seznec, "BeBoP: A Cost Effective Predictor Infrastructure for Superscalar Value Prediction," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 13–25, 2015.

[54] A. Perais and A. Seznec, "Cost Effective Physical Register Sharing," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 694–706, 2016.

[55] A. Perais, F. A. Endo, and A. Seznec, "Register Sharing for Equality Prediction," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 4:1–4:12, 2016.

[56] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 247–257, 2012.

[57] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh, "Prediction-Based Quality Control for Approximate Accelerators," in *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2015.

[58] K. Kuroyanagi and A. Seznec, "Service Value Aware Memory Scheduler by Estimating Request Weight and Using Per-Thread Traffic Lights," in *Memory Scheduling Championship (MSC)*, 2012.

[59] N. Prémillieu and A. Seznec, *SPREPI: Selective Prediction and Replay for Predicated Instructions*. PhD thesis, INRIA, 2013.

[60] F. Khorasani, H. A. Esfeden, N. Abu-Ghazaleh, and V. Sarkar, "In-Register Parameter Caching for Dynamic Neural Nets with Virtual Persistent Processor Specialization," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[61] A. Vakil-Ghahani, S. Mahdizadeh-Shahri, M.-R. Lotfi-Namin, M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Cache Replacement Policy Based on Expected Hit Count," *IEEE Computer Architecture Letters (CAL)*, vol. 17, no. 1, pp. 64–67, 2018.

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

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

[64] S. H. Pugsley, Z. Chishti, C. Wilkerson, P.-f. Chuang, R. L. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, "Sandbox Prefetching: Safe Run-Time Evaluation of Aggressive Prefetchers," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 626–637, 2014.

[65] K. J. Nesbit, A. S. Dhodapkar, and J. E. Smith, "AC/DC: An Adaptive Data Cache Prefetcher," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 135–145, 2004.

[66] K. J. Nesbit and J. E. Smith, "Data Cache Prefetching Using a Global History Buffer," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 96–96, 2004.

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

[68] J.-L. Baer and T.-F. Chen, "An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty," in *Proceedings of the International Conference on Supercomputing (ICS)*, pp. 176–186, 1991.

[69] S. Iacobovici, L. Spracklen, S. Kadambi, Y. Chou, and S. G. Abraham, "Effective Stream-Based and Execution-Based Data Prefetching," in *Proceedings of the International Conference on Supercomputing (ICS)*, pp. 1–11, 2004.

[70] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 364–373, 1990.

[71] "ChampSim." https://github.com/ChampSim/, 2017.

[72] S. Pugsley, A. Alameldeen, C. Wilkerson, and H. Kim, "The Second Data Prefetching Championship (DPC-2)," 2015.

[73] "Intel Xeon Processor E3-1220 v6." https://www.intel.com/content/www/us/en/products/processors/xeon/e3-processors/e3-1220-v6.html/, 2017.

[74] S. Franey and M. Lipasti, "Tag Tables," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 514–525, 2015.

[75] "CACTI 7.0: A Tool to Model Caches/Memories, 3D Stacking, and Off-Chip IO." https://github.com/HewlettPackard/cacti/, 2017.

[76] D. A. Jiménez and C. Lin, "Dynamic Branch Prediction with Perceptrons," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 197–206, 2001.

[77] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, 2006.

[78] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "SimFlex: Statistical Sampling of Computer System Simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, 2006.

[79] "The First JILP Data Prefetching Championship (DPC-1)." https://www.jilp.org/dpc/online/papers/DPC-1-intro.pdf/, 2009.

[80] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient Footprint Caching for Tagless DRAM Caches," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 237–248, 2016.

[81] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 25–37, 2014.

[82] A. Kolli, A. Saidi, and T. F. Wenisch, "RDIP: Return-Address-Stack Directed Instruction Prefetching," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 260–271, 2013.

[83] P. Lotfi-Kamran, M. Modarressi, and H. Sarbazi-Azad, "Near-Ideal Networks-on-Chip for Servers," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 277–288, 2017.

[84] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "IMP: Indirect Memory Prefetcher," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 178–190, 2015.

[85] A. Bhattacharjee, "Translation-Triggered Prefetching," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 63–76, 2017.

[86] D. Kadjo, J. Kim, P. Sharma, R. Panda, P. Gratz, and D. Jimenez, "B-Fetch: Branch Prediction Directed Prefetching for Chip-Multiprocessors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 623–634, 2014.

[87] M. Hashemi, K. Swersky, J. A. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan, "Learning Memory Access Patterns," in *International Conference on Machine Learning (ICML)*, 2018.

[88] L. Peled, U. Weiser, and Y. Etsion, "Towards Memory Prefetching with Neural Networks: Challenges and Insights," *arXiv preprint arXiv:1804.00478*, 2018.

[89] A. Seznec, "Decoupled Sectored Caches: Conciliating Low Tag Implementation Cost," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 384–393, 1994.