Accurately and Maximally Prefetching Spatial Data Access Patterns with Bingo

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

Spatial data prefetching techniques exploit the similarity of access patterns among memory pages to prefetch future memory references. 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. Existing spatial data prefetchers use pieces of information of trigger accesses (i.e., the first accesses to memory pages) as the *event* for storing the correlation records. They 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 find 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 cooperatively 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 146 single- and multi-core SPEC workloads, we show that Bingo improves system performance by 23% (up to 2.4x) over a baseline with no data prefetcher and 4% (up to 76%) over the best-performing prior spatial data prefetcher[1].

1 Introduction

Spatial data prefetchers predict future memory accesses by relying on spatial address correlation: the similarity of access patterns among memory pages[2]. 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 [2, 4, 5, 10, 11].

Whenever an application requests a page, spatial data prefetchers (e.g., [6, 7, 11]) 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[3]. Upon the reoccurrence of the same event, spatial prefetchers use the recorded footprint to prefetch future memory references of the currently requested page.

Inspired by TAGE [9], 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*. In this strategy, instead of relying on a single history table, several history tables, each with specific information, are used to make predictions. 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 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 [1]. 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.

In this work, 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., [6, 7, 11]), 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

[1] An extended version of this work was published in HPCA 2019 [3].

[2] 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.

[3] 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.

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, nonetheless, imposes significant area overhead. 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. By *organizing* the metadata in a single history table, we significantly reduce the storage requirements of Bingo.

2 Background and Motivation

Once a page is requested by an application for the first time (i.e., trigger access), spatial data prefetchers 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 ⟨*event*, *pattern*⟩ 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 [7] 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. [11] 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 use the stored footprint to predict and prefetch future memory references of the currently requested page.

A challenge in spatial data prefetching 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' [7] 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 unseen cache misses, because the exact same page should be requested in order for the stored footprint to be used. 'PC+Offset' [11], on the other hand, is *aggressive* and can cover unseen misses by applying the footprint information of a page to another, but predictions made based on it are not very accurate. In this work, 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.*

Figure 1 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.

[4]See Section 4.1 for the details of our methodology.

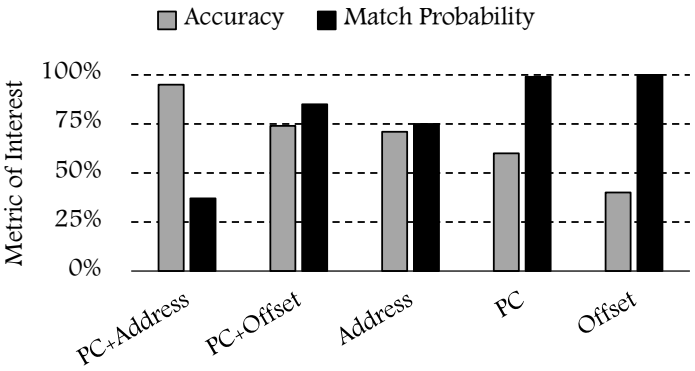

Figure 1. Accuracy and match probability of various heuristics as the event to which access history of pages are associated.

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 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 2 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 2, 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.

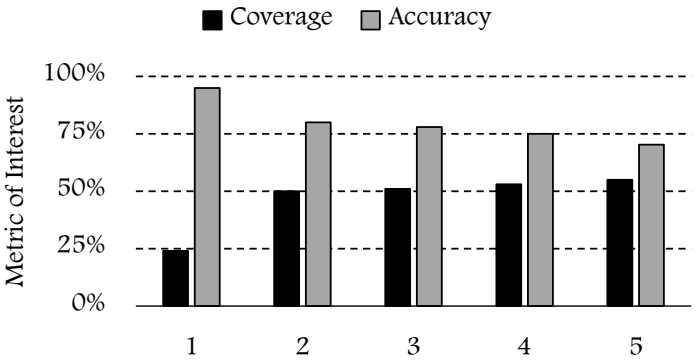

Figure 2. Coverage and accuracy of a TAGE-like prefetcher with varying number of events to which the footprints of pages are associated.

3 The Proposal

Like prior work [11], 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 [11]), 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. We carry out an experiment to measure the amount of redundancy: every time the spatial prefetcher needs to make a prediction, we determine if the long and short events offer the *same* prediction. We find that there is a significant amount of redundancy (66% on average) in the metadata tables of long and short events.

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 3 details our practical 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

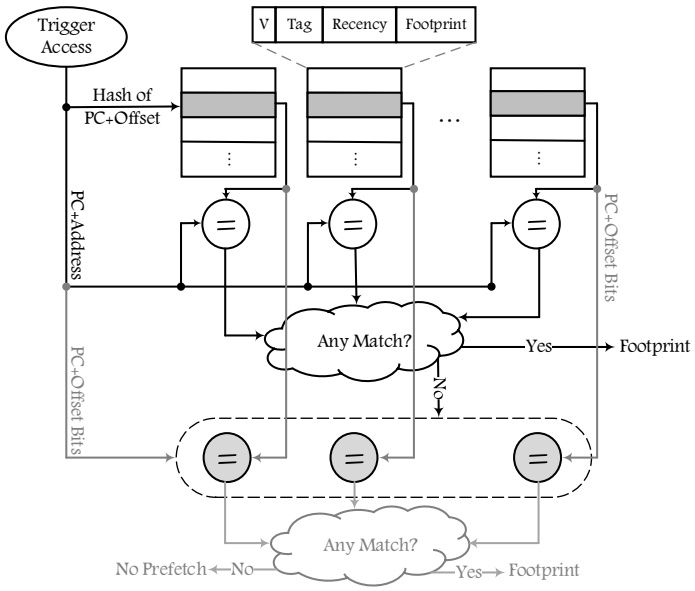

Figure 3. The details of the history table lookup in Bingo prefetcher. Gray parts indicate the case where lookup with long event fails to find a match. Each large rectangle indicates a physical way of the history table.

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 are 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 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 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

[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 [9] where multiple history lengths are used to index the metadata tables.

[6]Again, note that the bits corresponding to '`PC+Offset`' are carried in '`PC+Address`.'

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 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 multiple 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 the following heuristic gives the best performance with the configuration provided for DPC3: All matching entries are considered when issuing prefetch requests. If a cache block is present in the footprint of at least 75% of the matching entries, it is prefetched into the L1 cache. Otherwise, if a cache is present in the footprint of less than 75% but no less than 25% of the matching entries, it is prefetched into the L2 cache. If a cache block is present in the footprint of less than 25% of entries, no prefetch request is issued for the block.

4 Evaluation

4.1 Methodology

We evaluate our proposal in the context of the simulation framework provided with DPC3. We follow the evaluation methodology of the championship and run simulations for all 46 provided single-threaded applications and 100 randomly selected MIX workloads. For better readability, we only report the simulation results for applications whose performance is highly affected by the evaluated prefetchers, as well as the average of all simulated applications. We compare our proposal against prior state-of-the-art data prefetchers: Slim AMPM [12] (an enhanced version of AMPM [6], the champion of DPC1), BOP [8] (the champion of DPC2), and SMS [11] (the method on top of which our proposal is built). For all prefetching methods, including Bingo, we enlarge metadata tables to the extent that either the performance improvement of the prefetcher plateaus or DP3's rules are violated. All prefetchers sit in the L1 data cache and are trained by L1-D miss streams.

4.2 Results

Due to space limitations, we only include the performance (IPC) results of competing prefetchers. Figure 4 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. On average, Bingo improves performance by 23%, outperforming the second best-performing method (BOP) by 4%.

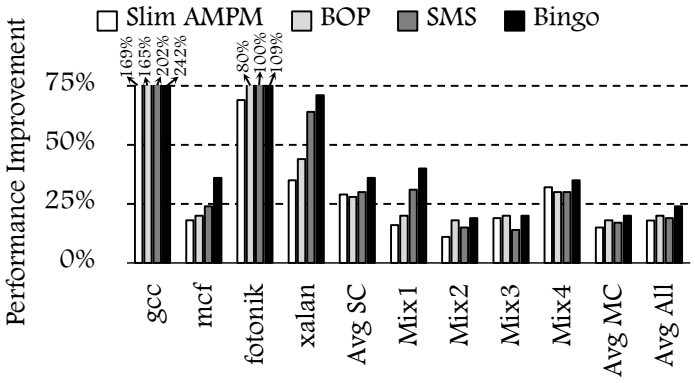

Figure 4. Performance comparison of prefetching techniques, normalized to a baseline system with no prefetcher. Mix1={lbm, pop2, roms, bwaves}, Mix2={fotonik, wrf, gcc, mcf}, Mix3={roms, fotonik, mcf, xalan}, and Mix4={wrf, mcf, lbm, roms}. 'Avg SC/MC/All' stands for the average of single-core/multi-core/all workloads.

Miss coverage, accuracy, and timeliness of prefetches are the main contributors to Bingo's superior performance improvement. By associating footprint metadata to more than one event, and matching the longest event, Bingo accurately and maximally extracts spatially-correlated data access patterns and significantly reduces the number of cache misses. On average, Bingo covers 8% more cache misses than the second best-performing method (SMS). The accuracy of Bingo's prefetch requests is 10% more than SMS, which is the second accurate prefetching approach among the evaluated ones. Finally, Bingo offers superior timeliness as compared to methods like BOP. Bingo, like other footprint-based prefetching approaches (e.g., SMS), gathers a footprint for every page and issues prefetch requests for all expected-to-be-used blocks of the page *at once*, which is not the case for methods like BOP.

[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.

References

[1] M. Bakhshalipour *et al.*, "Domino Temporal Data Prefetcher," in *HPCA*, 2018.
[2] M. Bakhshalipour *et al.*, "Fast Data Delivery for Many-Core Processors," *IEEE TC*, 2018.
[3] M. Bakhshalipour *et al.*, "Bingo Spatial Data Prefetcher," in *HPCA*, 2019.
[4] M. Bakhshalipour *et al.*, "Evaluation of Hardware Data Prefetchers on Server Processors," *ACM CSUR*, 2019.
[5] M. Bakhshalipour *et al.*, "Reducing Writebacks Through In-Cache Displacement," *ACM TODAES*, 2019.
[6] Y. Ishii *et al.*, "Access Map Pattern Matching for Data Cache Prefetch," in *ICS*, 2009.
[7] S. Kumar and C. Wilkerson, "Exploiting Spatial Locality in Data Caches Using Spatial Footprints," in *ISCA*, 1998.
[8] P. Michaud, "Best-Offset Hardware Prefetching," in *HPCA*, 2016.
[9] A. Seznec, "A Case for (Partially)-Tagged Geometric History Length Predictors," *JILP*, 2006.
[10] M. Shakerinava *et al.*, "Multi-Lookahead Offset Prefetching," *The Third Data Prefetching Championship*, 2019.
[11] S. Somogyi *et al.*, "Spatial Memory Streaming," in *ISCA*, 2006.
[12] V. Young and A. Krishna, "Towards Bandwidth-Efficient Prefetching with Slim AMPM," *The Second Data Prefetching Championship*, 2015.