PAPER

PMOP: Efficient Per-Page Most-Offset PrefetcherKanghee KIM^{†a)}, Member, Wooseok LEE^{††}, and Sangbang CHOI^{†b)}, Nonmembers

SUMMARY Hardware prefetching involves a sophisticated balance between accuracy, coverage, and timeliness while minimizing hardware cost. Recent prefetchers have achieved these goals, but they still require complex hardware and a significant amount of storage. In this paper, we propose an efficient Per-page Most-Offset Prefetcher (PMOP) that minimizes hardware cost and simultaneously improves accuracy while maintaining coverage and timeliness. We achieve these objectives using an enhanced offset prefetcher that performs well with a reasonable hardware cost. Our approach first addresses coverage and timeliness by allowing multiple Most-Offset predictions. To minimize offset interference between pages, the PMOP leverages a fine-grain per-page offset filter. This filter records the access history with page-IDs, which enables efficient mapping and tracking of multiple offset streams from diverse pages. Analysis results show that PMOP outperforms the state-of-the-art Signature Path Prefetcher while reducing storage overhead by a factor of 3.4.

key words: *memory hierarchy, cache, hardware prefetching*

1. Introduction

Hardware prefetching is essential to bridge the speed gap between the processor and memory [1]. By learning the memory access history of a running program, a prefetcher can predict future memory access and proactively cache the expected data. This eliminates memory latency, resulting in improved performance. However, this performance improvement requires a sophisticated balance between coverage, accuracy, and timeliness [2]. An increase in the amount of prefetched data is required to boost prefetcher coverage. Unfortunately, the additional data can also include unnecessary data, which can potentially degrade the overall performance. The prefetcher should also perform prefetches before a request arrives. Untimely prefetched data can have a negative impact on the bandwidth and cache capacity because the data occupies resources without providing any benefits. Therefore, an elaborate scheme balancing among accuracy, coverage and timeliness is necessary to improve performance.

Such a delicate technique requires a significant hardware cost. The previous delta prefetcher (VLDP) [3] tracks irregular memory access patterns by identifying them from

the memory address difference history and isolating interference between pages. This per-page delta tracking, however, requires a significant amount of storage for better performance [4]. A recent delta prefetcher, the Signature Path Prefetcher (SPP) [5], improves performance by prefetching delta paths that show high confidence while allowing tracking across physical page boundaries. Although the additional hardware resources incorporated by these prefetchers can improve performance, the designs are expensive and the hardware implementation is challenging.

Another recent offset prefetcher (BOP) [6] achieves good performance with relatively low hardware cost. Unlike delta prefetchers, which identify individual memory access streams, the offset prefetcher regards memory access as one global stream and identifies a best common offset to improve coverage and timeliness. Unfortunately, this approach still has some performance limitations compared to the state-of-the-art delta prefetcher because of useless prefetches that result from its aggressive prefetching policy. These prefetches of unwanted data also prevent the BOP from further increasing its aggressiveness due to potential performance degradation from cache interference and useless memory bandwidth usage.

In this paper, we propose an efficient Per-page Most-Offset Prefetcher (PMOP) that can minimize hardware cost and simultaneously improve accuracy while maintaining coverage and timeliness. First, we use Most-Offset to improve the aggressiveness of the offset prefetcher. Second, to filter useless data prefetching, we increase the granularity of memory stream monitoring to the page level. This fine-grain offset filtering facilitates improved accuracy management, leading to enhanced balancing, and thus, better performance. Finally, we minimize the storage overhead by proposing a page-ID indexing approach. The page-ID can represent repetitive memory access patterns in a page, allowing efficient offset filtering. Our evaluation results show that PMOP outperforms SPP by 8.1% on average while using less than 3.4 times the storage.

The paper is organized as follows. Section 2 shows related works. In Sect. 3, we describe why increasing aggressiveness in the offset prefetcher causes performance degradation. In Sect. 4, we propose our approach and detail how we implement the PMOP. We evaluate our approach in Sect. 5 and conclude the paper in Sect. 6.

Manuscript received September 27, 2018.

Manuscript revised February 25, 2019.

Manuscript publicized April 12, 2019.

[†]The authors are with the Inha University, 100 Inha-ro, Michuhol-gu, Incheon, Republic of Korea.

^{††}The author is with the University of Texas at Austin, 2501 Speedway, Austin, TX 78712, USA.

a) E-mail: kkh913@gmail.com

b) E-mail: sangbang@inha.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2018EDP7328

2. Related Work

Hardware costs and storage overhead for hardware prefetchers depend on how memory address differences are stored and handled. The prefetcher should know the address difference for prefetching. This is because the prefetcher predicts the difference between the current access address and subsequent access addresses based on the difference between the previous access addresses and the current access address. Therefore, the hardware costs and storage overhead differ depending on the number of address differences and the type of data structures involved.

Stride/Stream prefetcher. A *stride* [7] is defined when address differences are consecutively the same. A stride prefetcher issues a prefetch by adding a stride to the current address when a stable stride is detected. A *stream* [8] is defined when address differences have the same direction. A stream prefetcher prefetches data in the direction of the detected stream. Stride and stream prefetchers use a simple data structure table that indexes a program counter (PC) to detect addresses referenced from the same instruction and are embedded in many commercial processor products for high performance computing (HPC) [9]–[11]. However, they have poor performance against recent workloads, which have complicated memory access patterns that make it difficult for prefetchers to detect strides or streams. Therefore, they do not perform prefetches or produce inaccurate prefetches.

Delta prefetcher. Unlike stride, *delta* [3], [5], [12]–[15] allows a variety of address differences. That is, the difference from the previous address may not be the same as the difference from the prefetching address. Delta prefetching requires strict delta pattern matching, which can minimize inaccurate prefetches even with irregular memory access patterns. Of course, it is highly desirable to reduce unnecessary resource consumption by unpredictable patterns. However, the hardware complexity and storage overhead are somewhat greater than the performance improvements. Prefetching essentially boosts performance at the cost of additional hardware and energy. Because prefetching is very speculative, useless prefetches will inevitably occur. Therefore, the complex hardware logic required to retain robustness against such a risky policy is a significant challenge for people who produce real products. The Variable Length Delta Prefetcher (VLDP) [3] finds performance-beneficial delta sequences, tracks them on a delta history buffer (DHB) on a per-page basis, and prefetches deltas from the deepest history of a delta prediction table (DPT). Moreover, to compensate for a second cold miss near the page boundary, a small amount of speculative prefetching is made using an offset prediction table (OPT). The Signature Path Prefetcher (SPP) [5] employs compressed delta sequences called a signature, deeply prefetches deltas through looking ahead to the signature path, and prefetches more deltas across the page boundary.

Offset prefetcher. A *offset* [6], [16] is the address dif-

ference that would lead to optimal performance improvement from the current access address. The offset is not related to the access sequence of previous addresses, but merely depends on the frequency of occurrence. The offset prefetcher updates the most appropriate offsets at each learning interval. Since the offset is commonly applied to all addresses regardless of the physical page, the offset prefetcher does not require a data structure for per-page tracking. It requires only an additional structure to count the occurrence of the offsets during the interval. However, if the memory access pattern is complicated, an offset prefetcher tends to issue prefetches even if they are not the best offset for the page. This misleading globalization can cause a mass-production of useless prefetches. The Sandbox prefetcher (SBP) [16] pushes the addresses of the prefetch requests to a table called a sandbox instead of requesting them directly from physical memory to reduce harmful resource occupation due to useless prefetches. At the end of the learning interval, the SBP activates most prefetchers, and issues prefetches by adding offsets to the subsequent demand requests. However, SBP fails to provide timely prefetches due to the lack in consideration of memory latency, and produces excessive useless prefetches for complex, sparse, and irregular access stream. The Best-Offset prefetcher (BOP) [6] stores and processes the address of the prefetch response in the recent request (RR) table for timely prefetches, taking memory latency into consideration. Under offset evaluation, if the score of the offset exceeds the threshold SCOREMAX, BOP selects the corresponding offset as the BO (Best Offset), and begins a prefetch request using the BO for subsequent demand requests. Despite BOP's common access pattern recognition and timely prefetches, BOP still conservatively prefetches data due to its single offset based prefetching.

3. Motivation

In this section, we evaluate current offset prefetchers and discuss a promising method to improve performance as well as the limitations. To improve performance, a prefetcher needs to be able to process more data. For offset prefetchers, it is relatively straightforward to increase the amount of prefetching by increasing the number of prefetching offsets, defined as *prefetch degree*, from one best BOP to N sub-optimal [6]. We define this approach as a multi-degree offset prefetcher (MDOP). MDOP is an extension of BOP that allows a maximum degree of N .

Figure 1 shows examples and overall L2 useful/late/useless prefetches and corresponding IPC speedups of MDOP with various prefetch degrees in 20 memory intensive workloads. We mainly show L2 results because we find that the useful/late/useless prefetches in L2 are primary factors in overall performance of offset prefetcher; Pearson correlation between IPC speedup and (L2 Useful + L2 Late)/L2 Useless/(LLC Useful + LLC Late) are 0.87/–0.16/–0.09, respectively. We detail LLC information in Sect. 5.

For some workloads such as *libquantum* (Fig. 1 (a)),

Fig. 1 L2 useful/late/useless prefetches and corresponding IPC speedup of offset prefetchers at various prefetch degrees (x-axis). Plots are shown for libquantum (a) and sphinx3 (b) as examples. Averaged results of 20 workloads are shown in (c).

the performance scales as the degree increases. For some workloads such as *sphinx3* (Fig. 1 (b)), performance scaling is limited and decreases as the degree increases. The major performance difference between the two workloads originates from the number of useless prefetches. If the number of useless prefetches increases significantly when the degree is increased, the performance gain decreases despite the increase in useful prefetches. When the useless prefetches significantly stress the underlying memory subsystem, they negatively affect the overall IPC. We observed this trend with various workloads. Overall, the MDOP best performs at degree 4 (Fig. 1 (c)), and thus, MDOP uses degree-4 for the rest of this paper.

We further investigate the causes of massive useless production of MDOP. Figure 2 shows prefetching by MDOP in *milc* and *sphinx3*, which have different access patterns in a runtime. The x-axis of the graph is the CPU clock cycle, and the y-axis is the block address. The cache miss symbol “+” is used when data does not exist in the cache, the prefetch hit symbol “×” is used when a cache hit occurs on the prefetched block, and the prefetch fill symbol “★” is used when a prefetch block is inserted into the cache. Thus, the symbols represent the memory access pattern over time. MDOP issues prefetch requests for a cache miss or prefetch hit, *prefetch trigger event (PTE)*. If the prefetch block is successfully filled into the cache “★”, the useful prefetch block is later marked as “×”, otherwise “×” is not shown. A regular stream is the most predictable +1 offset stream (Fig. 2 (a)), while a sparse stream is a low access but predictable stream (Fig. 2 (b)). Streams with complex patterns have various strides and are difficult to predict (Fig. 2 (c)) while those with irregular patterns are random and unpredictable (Fig. 2 (d)). The offsets learned in *milc* produce no useless prefetches in Fig. 2 (a) but many useless prefetches in Fig. 2 (b). Unlike Fig. 2 (c), the off-

Fig. 2 Prefetching of MDOP in *milc* and *sphinx3* which have different access patterns per page at the same time.

sets learned in *sphinx3* are not suitable for Fig. 2 (d). In other words, offsets are over-predicted due to preemptive access to some pages as shown in Fig. 2 (a) and Fig. 2 (c), producing a large number of useless prefetches in Fig. 2 (b) and Fig. 2 (d). We call this *offset interference* when the offsets do not fit into different memory access patterns. For some workloads, the offset prefetcher cannot avoid offset interference as shown in Fig. 2. However, to avoid offset interference, throttling suboptimal offsets lowers the performance enhancement for regular streams and simultaneously reduces useful prefetches. Therefore, we need to change the methodology of the existing offset prefetcher, which has no way to selectively filter out useless offsets on individual pages.

Reducing useless prefetching is essential to improve overall performance for the offset prefetcher. Useless prefetching data competes for limited cache resources with useful data. Although an advanced cache subsystem might tolerate a certain amount of useless prefetches, the overall cache performance is degraded in many ways. In addition, useless prefetching reduces the available memory bandwidth, increasing the latency of useful data streams. Thus, degree scaling in an offset prefetcher without controlling useless prefetches will result in poor performance improvement. We detail how to reduce useless prefetching data while increasing the degree for performance improvements in the next section.

4. Per-Page Most-Offset Prefetcher

In this section, we propose an efficient Per-page Most-Offset Prefetcher (PMOP) that improves accuracy while maintaining coverage and timeliness through per-page offset filtering with low hardware cost. PMOP dynamically finds and prefetches most confident offsets (Most-Offset) to increase

Fig. 3 Overview of PMOP

coverage and timeliness. As discussed in previous section, it is essential to control the useless prefetesches. PMOP provides per-page filtering mechanism to increase accuracy. An important insight behind this fine-grain filtering is that each page has its own bias in accessing patterns. This per-page custom filters in PMOP allows localized prefetch issues from globally detected offsets, leading to accurate and timeliness data delivery.

However, granularity at the page level increases the hardware complexity and storage overhead. Large history tables are required to selectively track a page space $10^3 \sim 10^6$ times larger than the cache size. Hardware complexity can also be increased to efficiently search and manage table entries. To limit the hardware budget, PMOP introduces a *page-ID*, defined as the first offset found on the page. This is based on the fact that most repeated access patterns start with the same offset. The page-ID simplifies the hardware structure required for filtering prefetch candidate offsets by replacing a large physical page space with a small number of IDs.

Figure 3 shows an overview of PMOP. Prefetching begins with a PTE. PMOP includes two essential tables for page-IDs, Page Access History Table (PAHT), and Page-ID Table (PIDT). The PAHT is structured to track 64 pages with recent PTEs. Each entry has an 18-bit page tag and a page-ID. The subsequent PTEs can obtain the page-ID required for offset filtering from PAHT. The PIDT generates and manages page-IDs with eight entries. The larger the PIDT capacity, the better the performance gain through accurate filtering. However, the performance gain is limited to some workloads and results in a very large storage overhead requirement. Therefore, we introduce a small table for specially managing some page-IDs that are generated repeatedly.

PMOP uses two bloom filters, address bloom filter (ABF) and offset bloom filter (OBF), to store data for prefetching and filtering. A bloom filter is a hash-based, space-efficient structure [17] and is suitable for monitoring huge memory spaces such as prefetchers with a low hardware cost. The ABF accumulates the addresses of PTEs and the PMOP finds previous access records in the ABF to calculate the offset. The OBF stores data for offset filtering. The inputs for OBF consist of a page-ID, an offset, and a

block location within the page (*block offset*) where the offset was found. The bloom filter can include false positives and requires careful attention to the filter size, and the number of inputs and hashes. In this paper, we define the number of inputs and hashes that do not exceed 1% false positives according to a bloom filter approximation formula. If this input limit is exceeded, the bloom filters are initialized and new data is entered.

The scoreboard (SB) is a table that contains offset scores counted during the learning interval. Offsets whose score exceeds the threshold become prefetch candidates. The prefetch offset list (POL) has the latest offsets determined by the SB. PMOP produces prefetch requests only when the POL has offsets. A number of offsets in the POL indicates that the prefetch gain is high and vice versa, so the prefetch is naturally throttled up and down based on the POL. Page-ID generation, Most-Offset detection with filter data generation, and per-page offset filtering of the PMOP are described in detail in the remainder of this section.

4.1 Page-ID Generation

The PMOP should know whether the PTE is the first or the second access within the page to generate the page-ID. In addition, it should continuously provide filtering data with page-IDs to the OBF until all PTEs have ended within the page. PAHT entries contain the following fields to resolve these issues: (1) page tag, (2) page-ID, (3) init, (4) valid, and (5) least recently used. The PAHT can cover 64 different page tags, which are 18 least significant bits (LSBs) with 4 KB physical page addresses. This coverage is the same as an L2 cache size of 256 KB ($4KB \times 64 = 256KB$) and yields the best performance in our simulation baseline configuration. Page-ID (PID) can be obtained from the PIDT, and indicates an index of the PIDT entry. For example, when a PIDT has eight entries, the page-ID has a range of zero to seven. Init (I) indicates the first accessed block offset in the page, and is required to obtain the first offset needed to define the page-ID. Valid (V) indicates whether the page-ID is valid, because the page-ID cannot be assigned unless the PTE occurs more than twice. The Least Recently Used (LRU) is a field for LRU replacement and allows the PAHT to track recent PTEs without early eviction. The PIDT entry only requires two fields, offset and LRU. As defined earlier, the offset is the block address difference between the first access and the second access. On a 4 KB page, the offset has a range of -63 to 63 . The PIDT also uses an LRU replacement policy to maintain page-IDs that match recent memory access patterns. If the entry of the PIDT is replaced, the PMOP invalidates both the PAHT entries and the OBF related to the corresponding page-ID.

Figure 4 shows the process for page-ID generation. The PAHT searches for a PTE tag. If a tag does not exist, the PAHT replaces the entry with the lowest LRU value to add a new entry for the current PTE, as shown in Fig. 4(a). The absence of a tag in the PAHT indicates that the current PTE is the first access for the page. Therefore, the PAHT

Fig. 4 Page-ID generation using PAHT and PIDT.

Fig. 5 Most offset learning and filter data generation.

sets an I value of 0 and a V value of 0, and waits for the next access necessary to generate a page-ID. When PAHT finds a PTE tag, a V value of 0 indicates that the current PTE is the second access. PMOP accesses the PIDT to define the page-ID. In Fig. 4 (b), the offset for page tag 0x98C becomes 13, which is the difference between init 0 and the current offset 13. Since there is no offset 13 in the PIDT, the offset is inserted into the entry with the lowest LRU. After successfully creating the PIDT entry, the index of the entry is inserted into the PID of the PAHT entry. As a result, the page tag 0x98C has a page-ID of 1, and the subsequent PTEs for page tag 0x98C and the page-ID are used as the filtering information.

4.2 Most-Offset Learning and Filter Data Generation

Degree scaling for the offset prefetcher is straightforward and offsets are evaluated based on their scores. The score of the offset rises whenever the offset is found and prefetching is performed using offsets with a high score, which are called *Most-Offsets*. The MDOP described in Sect. 3 is a prefetcher with only Most-Offset learning. The PMOP uses evaluation offsets extended by prime factorization of one to five [6]. This approach is suitable for offset evaluation since

the scores of useful offsets are overlapped. The candidate offsets are evaluated in a round robin manner for each PTE.

Most-Offset learning in PMOP proceeds as follows. When the address of the PTE is X and the evaluation offset is O , the ABF ensures that the previous access $X - O$ is in the ABF. The ABF contains X s stored every PTE, unless it is refreshed above the input limit. If $X - O$ exists in the ABF, it means that O is useful, so the SB increases the score for O . If the score of O exceeds the SCOREMAX, O becomes a prefetch offset candidate and the score no longer increases. This process finds the Most-Offsets up to the maximum prefetch degree (A value of eight is the best in this paper.) until the end of the ROUNDMAX. If all Most-Offsets are obtained before the ROUNDMAX, the SB sends them into the POL. Once the transfer to the POL is complete, the SB is initialized and evaluation begins again to find new prefetch offset candidates.

The PMOP simultaneously generates data for per-page offset filtering and stores it in the OBF. Filter data is a combination of page-ID, block offset of $X - O$, and O . The presence of Most-Offsets on a page with the same page-ID does not mean that they lead to prefetch hits on the block offset of the current PTE. Therefore, to provide accurate filtering, the block offset of the $X - O$ at which the offset is found should also be stored. Unlike the ABF, the OBF inputs data in a separate space by the page-ID and does not have an input limit. This allows for conflict avoidance between different page-ID filter data, while providing more accurate filtering. The OBF is initialized when the corresponding page-ID is evicted from the PIDT.

4.3 Per-Page Offset Filtering

Most-Offsets prepared in the POL are global offsets with no offset interference considered. The PMOP can avoid offset interference by using filter data stored in the OBF. If there is no data for the PTE with a page-ID, the prefetch request is excluded. If the page-ID for the PTE does not exist, the PMOP issue prefetches to the LLC with relatively available resources to minimize L2 cache pollution.

5. Evaluation

In this section, we evaluate the performance of the PMOP with the following steps: (1) Define the simulation environment. (2) Analyze the single/multi-core simulation results compared to related prefetchers. (3) Check the sensitivity of the PMOP.

5.1 Evaluation Methodology

To evaluate the PMOP, we use the ChampSim simulator adopted for the 2nd cache replacement championship (CRC-2) [18]. ChampSim is a trace-driven micro-architecture simulator which can evaluate performance for cache prefetchers. Implementations of existing prefetchers are provided

Table 1 Detailed specification of system components for simulation baseline configuration.

Core	1-4 cores, 4GHz, OoO
BP	16K entry, gshare
Private L1 Dcache	32KB, 8-way, 4 cycles, 8 MSHRs, LRU
Private L2 cache	256KB, 8-way, 8 cycles, 16 MSHRs, LRU, Non-inclusive
Shared L3 (LLC)	2MB/core, 16-way, 20 cycles, 32 MSHRs, LRU, Non-inclusive
DRAM	4GB, 1-2 64 bits channels, 8 ranks/channel, 8 banks/rank, 1600 MT/s

by CRC-2 or the 2nd data prefetching championship (DPC-2) [4], and parameters are used to achieve the best performance for each prefetcher. We select 20 memory-intensive workloads from SPEC CPU 2006 that are the most prefetch-friendly, and simulate 200 M warmup instructions/1 B simulation instructions. We also choose three sets of multi-programmed workloads for multi-core simulations. Each set of programs combines various memory access patterns. We also compare PMOP against the SPP [5], which is a state-of-the-art delta prefetcher that outperforms the VLDP [3].

Table 1 shows the baseline configuration for each system component. The CPU has a clock rate of 4 GHz per core and operates with out-of-order execution. Four cores are used in multi-core simulations. Except for the L3 cache, which is a shared cache, the L1 and L2 cache are placed independently on each core. The L1 cache is divided into a 32 KB instruction cache and a 32 KB data cache, which communicates directly with the cores. The 256 KB L2 cache is connected to the prefetcher. There are a total of 16 L2 Miss Status Holding Registers (MSHRs), which store information about the request for cache misses or prefetch requests and remove them after receiving responses from the lower memory level. The L3 cache is a cache shared by all cores and has a size of 2 MB (single)/8 MB (multi). All caches have block sizes of 64 B and page sizes of 4 KB. The number of DRAM channels is one channel for a single core and two channels for multi-cores. All prefetchers are triggered with L2 access and fetch the cache block on L2 or LLC.

5.2 Single Core Performance

Figure 6 shows the IPC speedup for four evaluation prefetchers, normalized based on the results without prefetching in the baseline configuration. On average, the PMOP achieves 17.6% and 8.1% improvement over the BOP and SPP, respectively. The PMOP shows more benefits due to an aggressive prefetch strategy for the regular stream pattern that accesses most or all blocks in the page, such as GemsFDTD, lbm, leslie3d, and libquantum. In milc, which has regular but different offsets on each page, the PMOP is 68.3% and 25.6% better than the BOP and SPP respectively because it filters useless offsets to eliminate offset interference. The PMOP has lower performance than the SPP in omnetpp and zeusmp because it has difficulties covering pages over its page tracking capacity with a low hardware cost. Of course, the higher the number of

Fig. 6 Single core performance (IPC speedups normalized by no prefetching).

PIDT entries, the better the performance for those workloads. However, this leads to an exponential increase in the storage overhead. The MDOP shows the best geomean IPC speedup in the given base configuration and has the most aggressive prefetching for the regular stream patterns. However, to achieve this performance, the MDOP wastes a lot of useless resources. This is shown in the following Fig. 7.

Figure 7 shows the prefetch accuracy and coverage of each prefetcher in Fig. 6. The prefix on the x-axis correspond to BOP (B), MDOP (M), SPP (S) and PMOP (P), respectively. Useful means that a prefetch hit occurred. Late means that a demand miss occurred before the prefetch response is fetched to the cache. Although Late cannot generate prefetch hits, it has the positive effect of accelerating the fetching of data. Prefetch coverage is the portion of the prefetches that covers cache misses ($L2\ Useful + LLC\ Useful + L2\ Late + LLC\ Late$). Uncovered indicates a cache miss that the prefetcher did not handle. The sum of the covered and uncovered prefetches is 100%, and thus, other portions are Useless that do not have a positive effect on performance improvement. Prefetch accuracy is the ratio of ($L2\ Useful + LLC\ Useful$) to total prefetches ($L2\ Useful + LLC\ Useful + L2\ Useless + LLC\ Useless$).

On average, the total percentage of useless prefetches for the PMOP is 43.0% and 179.3% less than those for the BOP and MDOP, respectively. The MDOP, which has the best coverage in Fig. 6, produces many useless prefetches for GemsFDTD, gcc, gobmk, mcf, milc, perlbench, soplex, sphinx3, and zeusmp. This mass production of useless prefetches would pollute the cache in a resource-sharing multi-core environment, degrading the performance more than no prefetching. In contrast, the PMOP achieves high prefetch accuracy by avoiding offset interference through per-page offset filtering.

Despite degree scaling, the PMOP has a same aver-

Fig. 7 Prefetch accuracy and coverage. Useful ($L2\ Useful + LLC\ Useful + L2\ Late + LLC\ Late$) is the coverage that means the ability to hide cache misses. Useful + Uncovered = 100%. Otherwise Useless ($L2\ Useless + LLC\ Useless$) that means overpredicted and no performance beneficial.

Table 2 Configuration of multi-programmed workloads.

mix1	GemsFDTD, lbm, leslie3d, libquantum
mix2	astar, lbm, libquantum, milc
mix3	astar, milc, soplex, xalancbmk

age prefetch coverage as 1-degree BOP. This is because prefetches are selectively throttled during per-page filtering according to the workload. For example, offset prefetching in *bzip2* and *omnetpp* has a high coverage with a massive amount of useless prefetches. Offsets that provide useful and useless prefetches simultaneously are mostly filtered in the PMOP, potentially resulting in a small decrease in coverage. In *GemsFDTD*, *lbm*, *leslie3d*, and *libquantum*, Most-Offsets are fully used for prefetching without offset filtering, which greatly increases coverage.

SPP heavily depends on LLC prefetching, which accounts for 51.7% of useful prefetches. Although utilizing a relatively tolerant LLC reduces the burden on L2, such a policy causes untimely prefetches, leading to lost opportunities for boosting performance. However, the PMOP fetches the most useful prefetches in L2 for better timeliness, and LLC prefetch is allowed only when the page-ID does not exist.

5.3 Multi-Core Performance

We configured three sets of multi-programmed workloads, similar to [16], to assess the performance of PMOP in a multi-core system. Table 2 shows the composition of each mix.

Figure 8 shows the weighted IPC speedups [19] of four prefetchers for each mix. On average, the PMOP outperforms the MDOP and SPP by 93.4% and 8.78%, respectively. Mix 1 has the highest prefetch benefit because it is combined with regular streams that are easy to predict. The PMOP shows a weighted IPC speedup, which is 122% and 16% better than the BOP and MDOP, respectively. The SPP exhibits 19.3% worse performance due to untimely

Fig. 8 Weighted IPC speedups for multi-programmed mixes (normalized by no prefetching).

prefetching with excessive LLC dependence. Unlike the SPP, the PMOP offers the best performance through accurate and timely L2 prefetching. For mix 2, the PMOP achieves remarkable performance although *lbm* and *milc* have complex memory patterns, making it difficult to use global offsets. This means that the per-page approach of the PMOP works properly to filter offset interference for mix 2. The PMOP achieves the best performance in mix 3, which includes workloads with irregular patterns. For an irregular access pattern, the offset prefetcher would produce a lot of useless prefetches due to mistaking the patterns. Therefore, it should decide whether to use offsets with very precise verification or to block prefetches. PMOP focuses on the latter and achieves better performance than other offset prefetchers through more stringent per-page offset filtering.

5.4 Sensitivity Test

This section tests the sensitivity which affects the perfor-

Fig. 9 PMOP Page-ID sensitivity.

Fig. 10 PMOP PIDT sensitivity.

mance of PMOP.

5.4.1 Page-ID Sensitivity

PMOP employs the page-ID to separate memory access patterns per page and applies the filter data to overcome the offset interference by useless prefetches. In this section, we measure a sensitivity of the page-ID.

Figure 9 compares the IPC speedups and useful/useless prefetches according to the number of offsets constituting the page-ID in a simulation baseline configuration. One offset results in an IPC speedup of 4.8% and 12.9% compared to two and three offsets, respectively. Since the page-ID represents the first offset of the page, it can be expanded by combining the subsequent offsets in time order. More accurate offset filtering can be performed because the prediction error decreases as the number of offsets increases. However, as the time required for the page-ID increases and the conditions for the page-ID become more stringent, the prefetch coverage deteriorates, reducing the opportunity to improve performance. Therefore, one offset is sufficient for a page-ID representing the repeated pattern.

5.4.2 PIDT Sensitivity

Figure 10 shows IPC speedups, accuracy, and coverage according to the number of PIDT entries in the simulation baseline configuration. The best performance is achieved with 128 entities, which provides 2.34% higher performance than MDOP. However, an increase in the PIDT causes a sharp increase in the OBF. For example, for 128 PIDT entities, the OBF requires a large storage space of 16 KB. In addition, as the PIDP entry increases, the prefetch accuracy

Table 3 Hardware budget of evaluated prefetchers.

Prefetcher	Storage	Parameters
BOP [6]	0.4KB	256 entry RR Table, 26 entry SB
MDOP	0.4KB	BOP + 4B for most offsets
SPP [5]	5.37KB	256 entry ST, 512 entry PT, 1024 entry prefetch filter, 8 entry GHR
PMOP	1.6KB	ABF 256B, OBF 1024B, 64 entry PAHT, 8 entry PIDT, 26 entry SB, 8 entry POL

gradually decreases. Handling a lot of filter data means that the prefetcher can perform more accurate predictions, but conversely, will also maintain a lot of useless data. In this paper, the PMOP uses 8 entries because the OBF has a size of 1 KB, equal to the prefetch filter for the SPP, and provides a reasonable prefetch accuracy of 88.7%.

5.5 Hardware Budget

The storage requirement for PMOP is 1.6KB as shown in Table 3. PMOP increases 1.2KB of storage compared to BOP (0.4KB) [6], but saves 3.77KB of storage compared to SPP (5.37KB) [5]. PMOP is a novel and efficient architecture that can achieve maximum performance return to small storage investment. In particular, combining filters with various per-page storage allows us to maximize performance, higher than SPP, without tracking a significant amount of per-page offset history.

6. Conclusion

Offset prefetching is very cost-effective and provides high performance gains. An offset prefetcher can easily increase the degree to achieve higher performance, but its low prefetch accuracy makes aggressive prefetching difficult, leading to an imbalance among accuracy, coverage, and timeliness. The PMOP enhances the offset prefetcher with low cost offset filtering indexed by page-IDs, which represent different access patterns on each page. This per-page offset filter avoids offset interference to filter useless offsets and results in better performance. The performance of the PMOP is 17.6% and 8.1% better than that of the BOP and SPP, respectively, and the storage overhead is 3.4 times lower than that of the SPP.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2010-0020163)

References

- [1] W. Wulf and S.A. McKee, "Hitting the memory wall: Implications of the obvious," Tech. Rep., Charlottesville, VA, USA, 1994.
- [2] P.G. Emma, A. Hartstein, T.R. Puzak, and V. Srinivasan, "Exploring the limits of prefetching," IBM Journal of Research and Development, vol.49, no.1, pp.127–144, Jan. 2005.
- [3] M. Shevgoor, S. Koladiya, R. Balasubramanian, C. Wilkerson,

S.H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48, New York, New York, USA, pp.141–152, ACM Press, 2015.

- [4] DPC-2, "The 2nd Data Prefetching Championship." <http://comparch-conf.gatech.edu/dpc2/>. accessed Sept. 7. 2018.
- [5] J. Kim, S.H. Pugsley, P.V. Gratz, A.L.N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," Proceedings of the Annual International Symposium on Microarchitecture, MICRO, pp.1–12, IEEE, Oct. 2016.
- [6] P. Michaud, "Best-offset hardware prefetching," Proceedings - International Symposium on High-Performance Computer Architecture, pp.469–480, IEEE, March 2016.
- [7] F. Dahlgren and P. Stenstrom, "Effectiveness of hardware-based stride and sequential prefetching in shared-memory multiprocessors," Proceedings of 1995 1st IEEE Symposium on High Performance Computer Architecture, pp.68–77, IEEE Comput. Soc. Press, 1995.
- [8] N.P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," Proceedings of the 17th annual international symposium on Computer Architecture - ISCA '90, New York, New York, USA, pp.364–373, ACM Press, 1990.
- [9] Intel, "Intel 64 and IA-32 Architectures Optimization Reference Manual," Intel Technology Journal, vol.09, no.03, pp.1–660, 2005.
- [10] AMD, "Software Optimization Guide for AMD Family 17h Processors," June 2017.
- [11] ARM, "ARM Cortex-A75 Core Technical Reference Manual," July 2018.
- [12] D. Joseph, and D. Grunwald, "Prefetching using Markov predictors," Proceedings of the 24th annual international symposium on Computer architecture - ISCA '97, New York, New York, USA, pp.252–263, ACM Press, 1997.
- [13] K.J. Nesbit and J.E. Smith, "Data Cache Prefetching Using a Global History Buffer," 10th International Symposium on High Performance Computer Architecture (HPCA'04), pp.96–96, IEEE, 2004.
- [14] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-46, New York, New York, USA, pp.247–259, ACM Press, 2013.
- [15] D.A. Varkey, B. Panda, and M. Mutyam, "RCTP: Region Correlated Temporal Prefetcher," 2017 IEEE International Conference on Computer Design (ICCD), pp.73–80, IEEE, Nov. 2017.
- [16] 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," Proceedings - International Symposium on High-Performance Computer Architecture, pp.626–637, IEEE, Feb. 2014.
- [17] B.H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol.13, no.7, pp.422–426, July 1970.
- [18] CRC-2, "The 2nd Cache Replacement Championship." <https://googl/pRuovT>. accessed Sept. 7. 2018.
- [19] S. Eyerhan and L. Eeckhout, "System-level performance metrics for multiprogram workloads," IEEE Micro, vol.28, no.3, pp.42–53, May 2008.

Kanghee Kim received the B.S. degree in electronics engineering from Inha University, Incheon, Korea, in 2013. He is currently working toward the Ph.D. degree in the Computer Architecture and Networks Laboratory. His research interests include computer architecture, computer networks, and WSN.

Wooseok Lee received the B.S. degree in electrical engineering from Inha University, Incheon, Korea, in 2003. He was a senior research engineer at LG Electronics from 2005 to 2012. He is currently working toward the Ph.D. degree in The University of Texas at Austin. His research interests include computer architecture and embedded system.

Sangbang Choi earned the M.S. and Ph.D. in electrical engineering from the University of Washington, Seattle, in 1988 and 1990, respectively. He is currently a professor of electronic engineering at Inha University, Incheon, Korea. His research interests include computer architecture, computer networks, wireless communication, parallel and distributed systems.