# LSP: Collective Cross-Page Prefetching for NVM

Haiyang Pan*†, Yuhang Liu*†‡, Tianyue Lu*†‡, Mingyu Chen*†‡
*University of Chinese Academy of Sciences, Beijing, China
†State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS, Beijing, China
‡Peng Cheng Laboratory, Shenzhen, China
{panhaiyang, liuyuhang, lutianyue, cmy}@ict.ac.cn

***Abstract*—As an emerging technique, non-volatile memory (NVM) provides valuable opportunities for boosting the memory system, which is vital for the computing system performance. However, one challenge preventing NVM from replacing DRAM as the main memory is that NVM row activation's latency is much longer (by approximately 10x) than that of DRAM. To address this issue, we present a collective cross-page prefetching scheme that can accurately open an NVM row in advance and then prefetch the data blocks from the opened row with low overhead. We identify a memory access pattern (referred to as a ladder stream) to facilitate prefetching that can cross page boundary, and propose the ladder stream prefetcher (LSP) for NVM. In LSP, two crucial components have been well designed. Collective Prefetch Table is proposed to reduce the interference with demand requests caused by prefetching through speculatively scheduling the prefetching according to the states of the memory queue. It is implemented with low overhead by using single entry to track multiple prefetches. Memory Mapping Table is proposed to accurately prefetch future pages by maintaining the mapping between physical and virtual addresses. Experimental evaluations show that LSP improves the memory system performance with no prefetching by 66%, and the improvement over the state-of-the-art prefetchers, Access Map Pattern Matching Prefetcher (AMPM), Best-Offset Prefetcher (BOP) and Signature Path Prefetcher (SPP) is 26.6%, 21.7% and 27.4%, respectively.**

***Index Terms*—Prefetch, DRAM Cache, Non-volatile Memory**

## I. Introduction

To meet the increasing memory capacity requirements of diverse data-intensive applications, Intel released a memory product called the Optane™ DC Persistent Memory Module (Optane DC PMM) [1]. When it works in the memory mode, DDR4 DIMM acts as the cache of NVM. For simplicity, the hybrid memory structure is referred to as the external DRAM cache and non-volatile main memory (EDC-NVMM) in this paper. EDC-NVMM combines the advantages of both DRAM (low latency) and NVM (large capacity), providing large capacity with short access latency.

However, EDC-NVMM faces a performance issue when serving data-intensive applications [1] due to two factors: (1) NVMM has low row buffer locality. An evaluation [2] has shown that the row buffer hit rate of NVMM is less than 50%. (2) The overhead of an NVM row buffer miss is much higher than that of DRAM since the latency of NVM activation (i.e., the delay in transferring data from the memory module to the row buffer) is approximately 10x that of DRAM activation [3].

Prefetching technique has the potential to address this issue by opening future pages and obtain the data blocks in advance. We refer to this as cross-page prefetching. To adapt to the characteristics of EDC-NVMM, the design of cross-page prefetching must consider the following three requirements.

First, cross-page prefetching requires exploiting long-term future patterns. Existing prefetchers often avoid late prefetching through enlarging the prefetch distance or depth [6]–[9]. Due to NVM's long access latency, the prefetch distance for NVM needs to be longer than that for DRAM. Therefore, cross-page prefetching for EDC-NVMM needs to exploit patterns that can hold over a more long-term future. However, most existing prefetchers [6]–[9] designed for DRAM main memory rarely utilize the patterns that cross page boundary, so the effect of reducing late prefetching for NVM is limited.

Second, cross-page prefetching requires trading wasteful prefetching for high coverage and fewer row activations. Once a page is opened, the blocks that may be accessed on the page in the future should be prefetched as many as possible, avoiding open the page again when the missed data block is accessed in the future. However, previous work [10] has shown that the access locality in low level of a memory hierarchy has been weakened by out-of-order, high-parallel execution. Such irregularity of access patterns is prone to incur waste prefetching. To avoid wasteful prefetching, prefetchers often deliberately reduce their prefetching coverage [6].

Third, cross-page prefetching requires accurate prediction of future physical pages. However, it is not easy since hardware prefetchers located in the low-level of a memory hierarchy lack the mapping between virtual and physical addresses [6]. A previous evaluation [4] found that in real applications, more than 50% of consecutive virtual pages map to non-consecutive physical pages. The interrupted mapping hurts the prefetch accuracy of cross-page prefetching. Transparent huge page is a practical approach to avoid the interrupted mapping. However, it is not ideal and often disabled in many large memory applications due to the memory fragmentation problem [5].

The goal of this study is to design a prefetcher that can satisfy the above three requirements. First, we analyzed the long-term access patterns of many data-intensive applications and found a memory pattern from two aspects: (1) Inter-page accesses often have a constant stride; (2) Most intra-page accesses are relatively concentrated of consecutive data blocks. In this paper, this pattern is referred to as "ladder stream", and these consecutive intra-page blocks is "rung".

Using the ladder stream pattern, LSP can solve the first two issues discussed above as long as it can predict the rungs in far future pages. For a rung whose length is K blocks, LSP can

978-3-9819263-5-4/DATE21/©2021 EDAA

aggressively issue K consecutive requests even if the accesses are out-of-order or interrupted. Note that these intra-page prefetching do not incur new activations. Thus the overhead of incorrect intra-page prefetching is low. LSP can also solve the third problem by safely trading wasteful prefetching for high coverage and fewer row activations. However, issuing many requests would occupy the limited memory queue entries and significantly interfere with demand requests.

To reduce the interference, we propose a structure called Collective Prefetch Table (CPT). Prefetch requests do not directly enter into the request queue but are temporarily stored in CPT. CPT speculatively schedules these requests according to the states of the memory queue, thereby reducing interference. To temporarily store as many requests as possible, CPT uses single entry to track multiple prefetches for a rung. It only records the starting address and length of a rung. Thus it can temporarily store a large number of prefetches with limited hardware resources.

To improve cross-page predicting accuracy, we propose a structure called memory mapping table (MMT) to maintain the mapping between the virtual and physical memory addresses. The large capacity of the external DRAM cache makes it achievable to store this mapping in external DRAM.

In this study, we make the following contributions:

❶ We explore the necessity of cross-page prefetching for NVM. It requires exploiting long-term future patterns, high coverage, and accurately predicting future pages since the overhead of NVM row activation is significant.

❷ We find that the ladder stream pattern, which facilitates the cross-page prefetching for NVM, widely exists in diverse applications. In addition, we propose the structure and method to identify the ladder streams.

❸ Utilizing the ladder stream pattern, we build ladder stream prefetcher (LSP), which performs accurate inter-page prefetching and aggressive intra-page prefetching simultaneously. For inter-page accesses, LSP predicts future pages over the long-term using the regular strides between the rungs of ladder streams. For intra-page accesses, LSP trades wasteful prefetching for high coverage and fewer row activations by roughly fetching the future rungs.

❹ We propose two structural optimizations for LSP. CPT reduces the interference with demand requests by speculatively scheduling the prefetch requests according to the state of memory queue. Moreover, CPT saves the hardware resources by tracking multiple prefetches for a rung using a single entry. MMT helps LSP to work in virtual space and improves the accuracy of inter-page prefetching. MMT consumes 3.4% of the DRAM storage and incurs less than 0.9% performance overhead. Extensive evaluations show that, LSP improves the memory system performance by 66%, while the state-of-the-art prefetchers, AMPM [11], BOP [7], and SPP [7] achieve improvement by 26.6%, 21.7%, and 27.4%, respectively.

The rest of this paper is organized as follows. §II discusses the background and motivations. §III introduces the details of LSP. §IV presents the experimental evaluation of LSP. We discuss related work in §V and conclude in §VI.

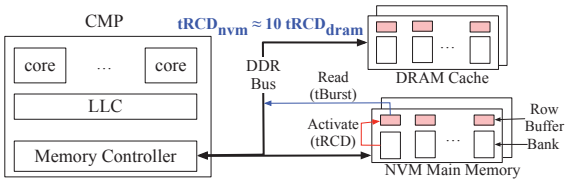


Fig. 1. The Structure of EDC-NVMM.

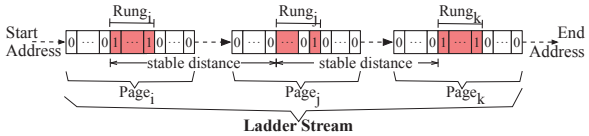


Fig. 2. Sketch diagram of a ladder stream.

## II. Background and Motivation

### A. Architecture of EDC-NVMM

Figure 1 shows a schematic of the EDC-NVMM architecture. Both the NVM and DRAM are accessed via traditional memory buses and managed by memory controller hardware. Each DRAM or NVM bank comprises rows and columns of memory cells. Each bank has a row buffer to cache the most recently accessed data in the bank. Each access to a given bank needs to read/write data from/to the row buffer.

The latency of an NVM row buffer read (i.e., transferring data from row buffer to data bus) is similar to that of DRAM. However, the operations of NVM precharge and activate (i.e., transferring data between row buffer and memory cells in banks) are much slower than those of DRAM. A typical configuration [3] shows that NVM's tRCD is approximately 10x that of DRAM. As a result, NVM's read performance is significantly lower than that of DRAM when the row buffer miss rate is high.

### B. Ladder Stream Pattern

Figure 2 is a sketch diagram of ladder stream pattern. It shows the access bitmaps of three pages, each bit of which illustrates whether the relevant block has been accessed (i.e., "1" indicates that it has been accessed, "0" indicates that it has not been accessed). Intra-page accesses are concentrated into a red area and the red areas have a stable distance. We define "Ladder Stream" as a memory flow that includes both intra-page accesses and inter-page accesses. The intra-page accesses are roughly concentrated into a series of approximately consecutive blocks. They form a pattern that looks similar to the rungs of a ladder. Besides, the inter-page accesses have a constant stride, meaning that the distance between the start addresses of rungs on the same ladder is fixed and quite stable.

Many applications have ladder stream pattern due to their algorithm logic and data structure. For instance, an analysis [12] has shown that in graph processing workload, the accesses to the vertex array may give rise to a ladder stream. In addition, in real applications, the distance between rungs may not be stable as ideal, but we find that the distance between the pages where the rungs are located is often stable.

The ladder stream pattern facilitates cross-page prefetching of EDC-NVMM from two perspectives: (1) Ladder streams provide easily predicted inter-page patterns in long-term future. (2) Intra-page prefetching for rungs only needs to roughly predict

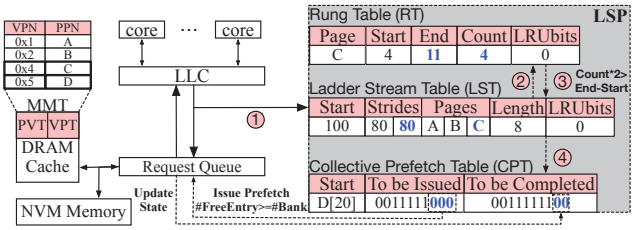


Fig. 3. Design of Ladder Stream Prefetcher.

a series of consecutive blocks, and the overhead of incorrect intra-page prediction is relatively low without incurring new row activations.

## III. Ladder Stream Prefetcher

We propose LSP in the memory controller. LSP tracks the memory requests arriving from LLC to the memory controller, identifies the ladder streams and issues prefetches into the memory request queue. The structure and execution are shown in Figure 3. When a memory access arrives, LSP first checks whether it belongs to an existing ladder stream. If it does, LSP merges it into this ladder stream (step 1); otherwise, LSP inserts it into the rung table (RT) and identify the accesses that can form the rungs (step 2). LSP then merges the identified rung into the ladder stream table (LST), predicts the future rungs in the ladder stream and inserts them into the CPT (step 3). According to the state of the memory request queue, CPT speculatively issues and tracks the prefetch requests (step 4). Besides, step 3 and 4 involve the address translation between virtual addresses and physical addresses using MMT. We describe this detailed process in §III-D.

### A. Rung Identification

LSP identifies rungs using RT. Each RT entry tracks the historical accesses to a physical page. "Page" is the physical page number. "Start" and "End" are the minimum and maximum block numbers accessed in this physical page, respectively. "Count" records the number of accesses to this page. RT entries are replaced according to LRU policy.

A set of intra-page accesses forms a rung in a ladder stream when access count is no less than half the rung length. It means that the accesses are concentrated into the continuous blocks. Unlike other prefetchers that learn intra-page access patterns, such as Stream Buffer Prefetcher (SBP) [13] and AMPM [11], LSP ignores the access order and therefore can tolerate the intricate patterns caused by out-of-order or interrupted accesses. In the example shown in Figure 3, when an access request for the 11th block in page C (denoted by C[11]) enters, it hits in an RT entry and the count turn to be 4. The length is 8, and the access count is equal to length/2; thus, the rung C[4, 11] is identified. After address translation, the rung {Start: 260, Length: 8, Page: C} is merged into LST.

### B. Rung Merging

LSP merges the rungs within a fixed address area into a ladder stream using LST. Each LST entry represents a ladder stream until the entry is evicted following the LRU policy. "Start" is the minimum block address of the ladder stream in virtual space. "Strides" represents the numbers of blocks between the rungs in this ladder stream. "Pages" is the array of physical page numbers where the rungs are located. "Length" represents the maximum rung length.

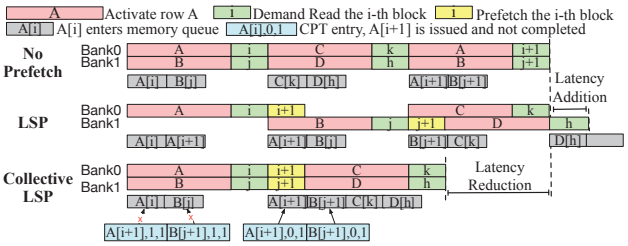


Fig. 4. The latency addition of aggressive intra-page prefetching, latency reduction of collective prefetch and states of the memory queue. A[i] represents the i-th block on Row$_a$.

When the difference between the starting address of a rung and the starting address of a ladder stream is less than some value (denoted by INTER), LSP identifies that rung as belonging to that ladder stream. LSP then calculates and inserts the new stride. If "Strides" is full, LSP calculates the stride values' distribution and determines this ladder stream's dominant stride value. Besides, LSP attempts to find the stable distance between the pages where the rungs are located if there is no dominant stride.

As shown in Figure 3, the page size is 4KB and the block size is 64B, the address mapping between physical and virtual is recorded in MMT. When the new rung C[4, 11] arrives, it hits in LST entry, and the new stride value is 80. The stride array is full, and the stable stride value is 80, so a ladder stream is identified. The starting block addresses of rungs in the virtual address space are "100, 180, 260". In physical address space, the rungs are "A[36, 43], B[52, 59], C[4, 11]". The rung length of this ladder stream is 8. Therefore, the i-th future rung is predicted to be 260+i$\times$80. The first future rung, D[20, 27], is a candidate for collective prefetch.

LSP dynamically adjusts the value of INTER. The higher the value of INTER is, the more likely it is that a series of rungs will be merged into a ladder stream. It can affect the prefetch accuracy and aggressiveness. Initially, LSP detects many values, obtains the prefetch accuracy under different parameters, and adopts the parameter that can provide the highest accuracy. Then, the prefetch accuracy is monitored. If the prefetch accuracy sharply drops (to less than half of the initial accuracy), the parameter detection mechanism is triggered again.

### C. Collective Prefetching Table

Figure 4 illustrates the inference with demand requests caused by aggressive intra-page prefetching and the latency reduction of collective prefetching. Suppose that there are four NVM memory rows in two banks: Row$_a$ and Row$_c$ are in Bank0, Row$_b$ and Row$_d$ are in Bank1. Bank0 and Bank1 share a request queue with a capacity of 2. It means that only up to two accesses can be outstanding at a time. The original block addresses of the memory requests are A[i], B[j], C[k], D[h], A[i+1], B[j+1].

In the case of no prefetching, each bank has three activations. In the case of LSP without collective prefetching, A[i+1]

and B[j+1] is prefetched and the sequence turns to be A[i], **A[i+1]**, B[j], **B[j+1]**, C[k], D[h]. Note that A[i+1] prevents B[j] from entering the memory queue and B[j] prevents C[k] from entering the memory queue. As a result, the overall latency is even longer than that of no prefetching, although LSP can eliminate two row activations.

In collective LSP, the prefetch requests do not directly enter the request queue but are temporarily stored in CPT. CPT speculatively issues them when the number of free memory queue entries is more than or equal to the number of banks. It ensures that the prefetch requests will not block the requests of each bank. To reduce CPT's storage overhead, we assemble the states of multiple prefetch requests for a rung into one entry. As shown in Figure 4, the delay for the waiting memory queue can be eliminated accordingly.

As shown in Figure 3, "Start" represents the starting address of collective prefetch. "To be Issued" is a bit map, each bit of which indicates whether the request of the related data block needs to be issued. "To be Completed" is a bitmap, each bit of which indicates whether the relevant data block has not been fetched. In Figure 3, the requests for D[20,22] have been issued, and the requests for D[20,21] have been completed. When the bitmaps are zero, this means the prefetching of this rung has been issued and completed. The CPT entry is then free.

### D. Memory Mapping Table

LSP maintains most mapping relations between virtual and physical addresses in two tables, including physical-to-virtual table (PVT) and virtual-to-physical table (VPT). In rung merging, LSP lookups PVT to find the virtual address of "Start" of the identified rung. If the "Strides" is full, LSP predicts future rungs via virtual addresses, then uses VPT to generate the real prefetch request by the physical address.

PVT is indexed by Physical Page Number (PPN) and translates the physical address to the virtual one. VPT is indexed by Virtual Page Number (VPN) and translates the virtual address to the physical one. We modify the page fault process in the operating system (OS). When a page fault occurs and a new mapping between physical and virtual pages is created, OS writes this mapping into MMT. Memory controller identifies the memory access by address. As shown in Figure 3, if the access is to the grey area in DRAM, MMT will be accessed. Otherwise, the access enters DRAM or NVM as usual.

Suppose that the size of NVM is 256GB and the page size is 4KB, meaning that the main memory contains 64M physical pages. For x64 architecture, one VPN requires 36 bits, so PVT needs 288MB (i.e., 64M×36 bits). As for VPT, the completed storage overhead is 208GB (i.e., $2^{36} \times 26$ bits). Completely recording VPT in DRAM is unrealistic. Besides, the case of multiple processes sharing memory needs to be considered. We use a simple hash function to convert process id and the VPN to an index that has 28 bits, so the hashed VPT requires 832MB (i.e., $2^{28}\times 26$ bits). We have evaluated that the accuracy of hashed VPT is close to that of complete VPT. For a typical EDC-NVMM configuration, the capacity of DRAM cache is 32GB. Therefore, MMT occupies approximately 3.4% of DRAM capacity. Regarding the performance overhead, each lookup of MMT requires one DRAM access.

The overhead of MMT is tolerable since DRAM cache has a large capacity. As future work, we can resort to multi-level MMT to reduce the storage overhead, at the cost of adding more external DRAM accesses. We would not discuss this case in this paper.

## IV. Experimental Evaluations

In this section, we represent evaluations for the proposed LSP. First, we describe the methodology, including the selection of workloads and competitors. Second, we present the results and analysis in terms of several metrics, including accuracy, timeliness, bandwidth utilization and energy consumption.

### A. Methodology

*1) Simulation:* Considering the speed and accuracy of simulation, we use a trace-driven simulator called NVMain [14], which simulates a memory system by taking the memory access traces as inputs. We use the model presented in [12] to evaluate the energy consumption of DRAM and NVM. The default configurations are shown in Table I. We run the workloads on an Intel Xeon E5 server with 32GB per memory channel. To simulate the multi-thread or multi-process execution, we run the workloads in 24 threads/processes, which is the same as the number of CPU cores of the server. We collect memory access traces using a well-known hardware memory tracing tool, HMTT [15], which has been strictly verified and widely used.

TABLE I

| Configurations For NVMain | DRAM Cache | NVMM |
|---|---|---|
| Capacity | 4GB | 32GB |
| Structure | 1 channel,8 banks | 1 channel,8 banks |
| Row Size | 2KB | 2KB |
| Bus Frequency | 64-bit, 1600MT/s | 64-bit, 800MT/s |
| Request queue entry counts | 128 | 128 |
| Array read(write) | 1.17(0.39) pJ/bit | 2.47(16.82) pJ/bit |
| Row buffer read(write) | 0.93(1.02)pJ/bit | 0.93(1.02)pJ/bit |
| tRCD-tRP-tBURST-tRTP-tCCD | **9-9**-4-3-2 | **90-270**-4-3-2 |

*2) Workloads:* Table II shows the workloads. We consider typical HPC and big-data applications and we configure the running memory (footprint) of workloads to almost occupy the entire main memory of the host server. For SPECCPU2006, we mix six memory-intensive workloads (400, 429, 440, 457, 463 and 470).

TABLE II

| Workloads | Description | Abbr. |
|---|---|---|
| GraphX(BFS,CC,PR,LP,TC) [12] | Graph Analysis | W1-W5 |
| K-Means [16] | Clustering Algorithm | W6 |
| Bayes [16] | Sort | W7 |
| Mixed SPECCPU2006 [17] | High Performance | W8 |
| Terasort [16] | Sort Algorithm | W9 |
| SPECJbb2005 [18] | Java Virtual Machine | W10 |
| SOAPdenovo2 [19] | Oligonucleotide Analysis | W11 |
| High Performance Linpack [20] | High Performance | W12 |
| NPB(CG,FT,LU,MG,SP) [21] | High Performance | W13-W17 |

*3) Competitor:* As competitors, we select four prefetchers that do not require program counter (PC) information-namely, SBP [13], AMPM [11], SPP [6], and BOP [7]. The baseline is a no-prefetching system. All the implementations of these competitors are from DPC2 [22]. Table III shows their on-chip

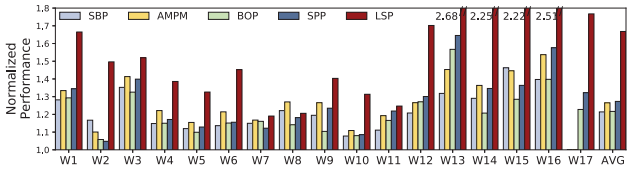


Fig. 5. Normalized Performance of different prefetchers versus the baseline.

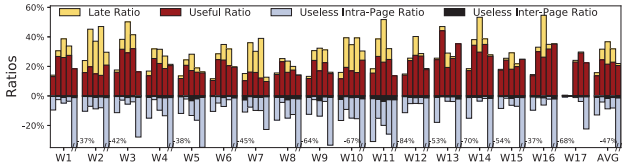


Fig. 6. Prefetch coverage and useless ratios.

storage overheads. Note that since MMT is stored in external DRAM, it does not take up LSP's on-chip storage overhead.

TABLE III

| Prefetcher | Overheads of the components | Overall Storage |
|---|---|---|
| SBP | 64b×64. | 512B |
| AMPM | Tag (52b), Prefetch map (64b), access map (64b), LRU (8b), Entry Count (256). | 5.88KB |
| SPP | Signature Table (43b×1024), Pattern Table (48b×16384), Prefetch Filter (8b×512). | 101.9KB |
| BOP | Scores (255b), Throttle (42b), Delay Queue (473b). | 97B |
| LSP | RT ((26+6+6+4+6)b×64), LST ((48+10×5+26×6+6+6)b×64), CPT ((32+32+32)×128). | 32KB |

### B. Results

*1) Performance:* We represent the memory system performance using the inverse of completion time in NVMain. Figure 5 shows the normalized memory system performance versus the baseline. On average, LSP achieves a speedup of 66%, while the speedups of SPP and AMPM are 27.4% and 26.6%, respectively. In particular, LSP achieves a significant improvement for ladder stream abundant workloads. For instance, LSP achieves a 95% greater speedup than SPP on NPB-MG. However, LSP has a limited effect when the workload lacks ladder stream patterns. For example, on SPEC-MIX, the speedup of LSP is lower than that of AMPM by 7.8%.

Moreover, when MMT and CPT are off, LSP has little speedup. Using virtual addresses without collective prefetching (i.e., MMT is on and CPT is off) can improve the performance by 12.4% on average. As discussed in §II, prefetch requests occupy hardware resources, such as memory queue entries; consequently, an excessive number of prefetch requests may result in performance loss. Therefore, when collective prefetch is included (i.e., MMT and CPT are on), the performance of LSP is significantly improved, by 48% on average. Moreover, on average, the accesses of MMT incur a performance overhead of 0.86%.

*2) Coverage, Accuracy and Timeliness:* We analyze the prefetching using three metrics: *useful* ratio, *late* ratio and *useless* ratio. The useful ratio is the proportion of blocks loaded into the cache before their first demand access. The late ratio is the proportion of blocks to which access is demanded before the corresponding prefetch requests have been satisfied. It reflects the timeliness of prefetching. The useless ratio is the proportion of prefetched blocks that are not required by demand accesses. Moreover, we split the useless ratio into inter-page and intra-page ratios.

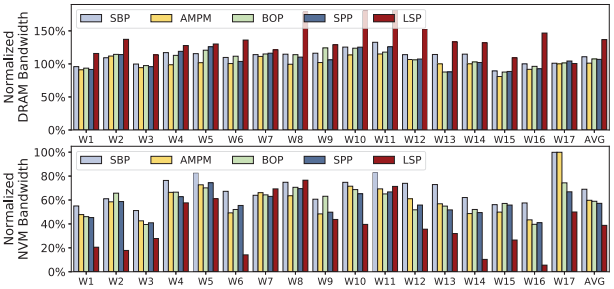


Fig. 7. Normalized DRAM and NVM bandwidth versus the baseline.

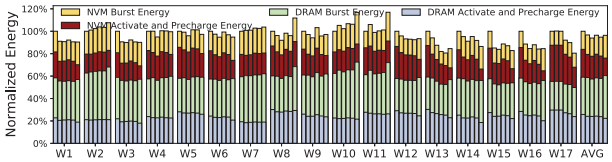


Fig. 8. Normalized dynamic energy consumption versus the baseline.

Figure 6 shows the results in terms of these metrics. The result of each workload contains five bars, indicating the prefetchers SBP, AMPM, BOP, SPP and LSP. The accuracies are 24.4%, 21.5%, 22.9% and 20.5%, respectively. Besides, LSP shows the lowest late ratio since LSP performs inter-page prefetching utilizing the ladder stream in the long-term future. As a result of aggressive intra-page prefetching, LSP shows the highest over-prefetching. On average, its useless ratio is 39% higher than that of SPP. Fortunately, its useless inter-page ratio is lower than AMPM, BOP, SPP. As shown in Table I, for row buffer hits, the read latency of the NVM is similar to that of DRAM. Thus, the overhead of useless intra-page prefetching is small, allowing LSP to achieve the highest speedup eventually.

*3) Bandwidth Utilization:* Figure 7 show the normalized DRAM and NVM bandwidth utilization of different prefetchers versus the baseline. The DRAM bandwidth represents the amount of data transferred from the NVM row buffer to the DDR bus. The NVM bandwidth represents the amount of data transferred from the NVM module to the NVM row buffer.

On the one hand, LSP utilizes more DRAM bandwidth due to its aggressive intra-page prefetching. On average, LSP consumes more DRAM bandwidth than SBP, AMPM, BOP and SPP by 36.7%, 35.6%, 29.2% and 30%, respectively. On the other hand, LSP reduces the NVM bandwidth utilization by 61.2% due to the accurate inter-page prefetching and aggressive intra-page prefetching. Specifically, LSP requires 21%, 20.2%, and 18.5% fewer NVM row activations than AMPM, BOP and SPP, respectively. As discussed in §I, the most critical challenge in EDC-NVMM is to hide or reduce the number of NVM row activations. Therefore, LSP can significantly improve the overall performance at the expense of occupying more DRAM bandwidth.

*4) Energy Consumption:* Figure 8 shows the results for the normalized dynamic energy cost versus the baseline. The result

of each workload contains six bars, indicating the prefetchers No-Prefetching, SBP, AMPM, BOP, SPP and LSP. On average, LSP consumes more DRAM energy than SBP, AMPM, BOP and SPP by 1.4%, 3%, 2% and 2.6%, respectively and consumes less NVM energy than SBP, AMPM, BOP, and SPP by 5%, 2%, 0.6%, 2.1% and 0.9%, respectively. Besides, due to aggressive intra-page prefetching, LSP consumes less energy of activation and precharge with more energy of burst. LSP saves the normalized energy versus the baseline by 3.6%, while the saved energy of SBP, AMPM, BOP, and SPP is 3.3%, 6%, 3.6% and 5.2%, respectively.

## V. Related Work

Existing hardware prefetchers often exploit memory patterns classified in terms of the physical page number or PC information. However, the PC information is lost in LLC or lower-level caches. Thus, some prefetchers, such as the spatial memory streaming prefetcher (SMS) [23], may lose their effectiveness without PC information since SMS requires the PC as the index of accessing the memory pattern table. To extend SMS to cross-page prefetching of EDC-NVMM, we have tried replacing PC value with the physical page number. The results show that this modification of SMS has little effect.

SBP [13] and its enhanced variants [8], [9] detect the stream within a page. They adjust the prefetch distance and the degree following prefetch feedback. AMPM [11] uses a bitmap to track the accesses in some fixed area and issues prefetch requests following the learned historical pattern. Thus, it can make accurate predictions for complex patterns. The effectiveness of SBP or AMPM for EDC-NVMM is limited since the mapping between physical and virtual addresses is often non-continuous. SPP [6] tracks the signature of each memory page and records a count for each signature. When the count for a particular signature is sufficiently high, prefetching is triggered. It supports cross-page accesses by recording delta patterns that cross page boundaries. The effectiveness of SPP is limited for two reasons: (1) The complex intra-pattern makes it difficult to find stable signature; (2) Large amount of memory pages makes it difficult to predict future pages due to the limited capacity for recording delta patterns. BOP [7] improves the timeliness of the sandbox prefetcher [24]. It tests various offsets and finds the best offsets following prefetching feedback. However, it considers the potential offsets within the physical page so it only needs to evaluate 63 values. When it is extended to cross-page prefetching, the potential offsets will be much more. Therefore, identifying a suitable offset from so many offsets is difficult.

## VI. Conclusion

In this paper, we propose collective cross-page prefetching to address the issue incurred by the long latency of NVM row activations. We found that the ladder stream pattern widely exist and can facilitate the NVM-friendly prefetching. Utilizing the ladder stream pattern, we propose LSP to perform accurate inter-page prefetching and aggressive intra-page prefetching simultaneously. For inter-page prefetching, LSP directly uses virtual addresses to identify the strides of inter-page accesses using a light-weight structure (i.e., MMT), which significantly increases the accuracy of inter-page prefetching. For intra-page prefetching, LSP aggregates multiple sequential intra-page prefetches into a single collective prefetch, thereby reducing interference with demand requests and saving hardware resources. LSP improves memory system performance in terms of execution time by 66%, while the improvement of SBP, AMPM, SPP and BOP is 21.4%, 26.6%, 27.4%, and 21.7%, respectively. Moreover, LSP saves the normalized energy versus the baseline by 3.6%, while the saved energy of SBP, AMPM, BOP, and SPP is 3.3%, 6%, 3.6% and 5.2%, respectively.

## VII. Acknowledgment

This work is supported by National Key Research and Development Plan of China 2017YFB1001602, National Natural Science Foundation of China (No. 61772497), Alibaba Group through Alibaba Innovative Research (AIR) Program under Grant No. SCCS542019015384, and Huawei Research Program under Grant No. TC20200827005_SOW20202021_001, and TC20201208005. Yuhang Liu is the corresponding author.

## References

[1] Yang, Jian, et al. “An empirical guide to the behavior and use of scalable persistent memory.” (FAST 20). 2020.
[2] Meza, Justin, et al. “Evaluating row buffer locality in future non-volatile main memories.” arXiv preprint arXiv:1812.06377 (2018).
[3] Young, Vinson, et al. “Accord: Enabling associativity for gigascale dram caches by coordinating way-install and way-prediction.” ISCA 2018.
[4] Dreslinski, Ronald G., et al. “Analysis of hardware prefetching across virtual page boundaries.” CF 2007.
[5] Panwar, Ashish, Aravinda Prasad, and K. Gopinath. “Making huge pages actually useful.” ASPLOS 2018.
[6] Kim J, et al. “Path confidence based lookahead prefetching.” MICRO 2016.
[7] Michaud, Pierre. ”Best-offset hardware prefetching.” HPCA 2016.
[8] Heirman, Wim, et al. “Near-side prefetch throttling: Adaptive prefetching for high-performance many-core processors.” PACT 2018.
[9] Srinath S, et al. “Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers.” HPCA 2007.
[10] Wang, Jiajun, Reena Panda, and Lizy Kurian John. “Prefetching for cloud workloads: An analysis based on address patterns.” ISPASS 2017.
[11] Ishii, Yasuo, Mary Inaba, and Kei Hiraki. “Access map pattern matching for data cache prefetch.” ICS 2009.
[12] Basak, Abanti, et al. “Analysis and optimization of the memory hierarchy for graph processing workloads.” HPCA 2019.
[13] Liu, Gang, et al. “Enhancements for accurate and timely streaming prefetcher.” J Instr Level Parallelism 13 (2011).
[14] Poremba, M. “NVMain 2.0: A User-Friendly Memory Simulator to Model NVM Systems.” CAL 2015.
[15] Bao, Yungang, et al. “HMTT: a platform independent full-system memory trace monitoring system.” SIGMETRICS 2008.
[16] Wang, Lei, et al. “Bigdatabench: A big data benchmark suite from internet services.” HPCA 2014.
[17] Spradling, Cloyce D. “SPEC CPU2006 benchmark tools.” ACM SIGARCH Computer Architecture News 35.1 (2007): 130-134.
[18] SPEC, OMP. “SPECJbb2005.” (2013).
[19] Luo R, et al. “SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler.” Gigascience 1.1 (2012): 2047-217X.
[20] Davies, Teresa, et al. “High performance linpack benchmark: a fault tolerant implementation without checkpointing.” ICS 2011.
[21] Bailey, David H., et al. “The NAS parallel benchmarks.” The International Journal of Supercomputing Applications 5.3 (1991): 63-73.
[22] The 2nd Data Prefetching Championship (DPC2). https://comparch-conf.gatech.edu/dpc2. 2015
[23] Somogyi, Stephen, et al. “Spatio-temporal memory streaming.” ACM SIGARCH Computer Architecture News 37.3 (2009): 69-80.
[24] Pugsley, Seth H., et al. “Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers.” HPCA 2014.