Cache in Hand: Expander-Driven CXL Prefetcher for Next Generation CXL-SSDs

Miryeong Kwon[*†], Sangwon Lee[*†], Myoungsoo Jung[*†]
[*]*Computer Architecture and Memory Systems Laboratory*, KAIST
[†]Panmnesia, inc.

ABSTRACT

Integrating compute express link (CXL) with SSDs allows scalable access to large memory but has slower speeds than DRAMs. We present ExPAND, an expander-driven CXL prefetcher that offloads last-level cache (LLC) prefetching from host CPU to CXL-SSDs. ExPAND uses a heterogeneous prediction algorithm for prefetching and ensures data consistency with CXL.mem's back-invalidation. We examine prefetch timeliness for accurate latency estimation. ExPAND, being aware of CXL multi-tiered switching, provides end-to-end latency for each CXL-SSD and precise prefetch timeliness estimations. Our method reduces CXL-SSD reliance and enables direct host cache access for most data. ExPAND enhances graph application performance by 3.5×, surpassing CXL-SSD pools with diverse prefetching strategies.

1 INTRODUCTION

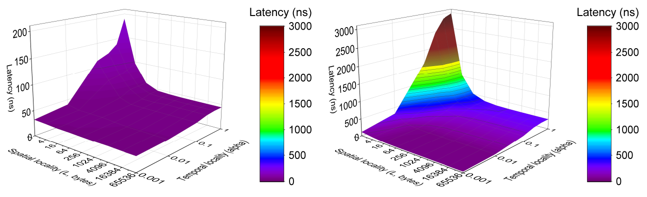
Compute Express Link (CXL) is receiving considerable attention as an emerging interface that separates memory resources from computing servers, allowing users to access large-capacity memory scalably. In terms of capacity, storage class memory (SCM) technologies such as PRAM [1], Z-NAND [2], and XL-Flash [3] offer greater advantages over DRAMs. As a result, both industry and academia strive to introduce byte-addressable solid-state drives (SSDs) using the CXL protocol and SCM's memory instruction semantics. For instance, one method integrates CXL into Optane SSDs for hierarchical memory expansion, while several proof-of-concepts (PoCs) employ new flash like Z-NAND and XL-Flash to develop CXL-SSDs [4–6].

While CXL-SSDs target capacity needs for memory disaggregation, their backend media remain slower than DRAMs. Specifically, PRAMs are 7× slower than DRAMs [7], and the new flash technologies exhibit latencies 30× slower [2]. To address this, industrial PoCs employ SSD-side DRAM buffers as internal caches, resembling high-performance NVMe storage with larger internal DRAMs. Although these buffers effectively handle write latency issues, they struggle to mask the long read latency caused by SCM backend media. Unlike file system-managed block devices, CXL-SSDs should serve memory requests (load/store) without relying on the host-side storage stack. Concealing long read latency necessitates understanding execution behaviors of host applications and managing the corresponding CPU cache hierarchy, appropriately. Regrettably, these aspects are neglected by existing SSD technologies, as they have solely handled block requests thus far.

When CXL-SSDs are placed in the system memory space as host-managed device memory, existing CPU-side cache prefetching mechanisms can still be beneficial. However, two main unaddressed challenges prevent current prefetchers within the cache hierarchy from fully utilizing the advantages of LLC with CXL-SSDs: i) *hardware logic size constraints* in handling a wide range of memory access patterns possibly encountered in the extensive CXL memory pooling space, and ii) *latency variations* experienced by different CXL-SSDs located in diverse positions within the CXL switch network.

In particular, rule-based cache prefetchers, such as spatial [8–10] and temporal prefetching algorithms [11–13], require tens of MB storage that is similar to the actual *last-level cache* (LLC) of a CPU [9]. As a result, modern CPUs employ a simpler stream cache prefetching algorithm [14], which unfortunately is unable to mask the increased latency introduced by CXL-SSDs. Another contributing factor is the interconnect network topology used in CXL-based memory disaggregation. To boost memory capacity in a scalable way, CXL introduces a multi-level switch architecture where each level can potentially increase memory expander latency, depending on the target's position within the network. This is because the processing time taken by CXL switches at different levels cannot be overlooked. Consequently, existing prefetchers are

HotStorage '23, July 9, 2023, Boston, MA, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0224-2/23/07…$15.00
https://doi.org/10.1145/3599691.3603406



(a) LocalDRAM. **(b) CXL-SSD.**

Figure 1: Analyzing the impact of locality.

unable to retrieve data from various locations of CXL-SSDs in a timely fashion.

This position paper introduces an expander-driven CXL prefetcher, *ExPAND*, designed to offload primary LLC prefetch tasks from host CPU to CXL-SSDs, with CPU design area constraints in mind. Implemented in CXL-SSDs, ExPAND manages data prefetching across various expander accesses using a heterogeneous machine learning algorithm for address prediction. ExPAND's host logic ensures CXL-SSDs remain aware of host-side CPU execution semantics, while its SSD-side logic maintains data consistency between LLC and CXL-SSDs through CXL.mem's *back-invalidation* (BI). This bidirectional, collaborative approach allows user applications to access most data directly at the host, significantly reducing dependence on CXL-SSDs.

On the other hand, accurately estimating data prefetching latency is crucial for effectively utilizing the limited capacity of on-chip caches. We examine the concept of *prefetch timeliness* to denote this latency requirement in CXL. To maintain a comprehensive understanding of prefetch timeliness for different CXL-SSDs, ExPAND identifies the underlying CXL network topology and device latencies during PCIe enumeration and device discovery. Utilizing this information, it calculates more precise end-to-end latency for each CXL-SSD in the network and communicates it by writing the value to each device's PCIe configuration space. As a result, the offloaded cache prefetching algorithm can determine the optimal timing for data retrieval to the host LLC, effectively reducing the long latency imposed by the SSD's backend media.

Our evaluation results show that ExPAND can significantly enhance the performance of various graph applications, characterized by highly irregular memory access patterns, by 3.5×, 2.1×, and 1.5× compared to a CXL-SSD expanded memory pool without a CXL-prefetcher, with rule-based prefetchers, and simple ML-based prefetchers, respectively.

2 NEW CACHE/MEMORY HIERARCHY

2.1 Memory Pooling using CXL

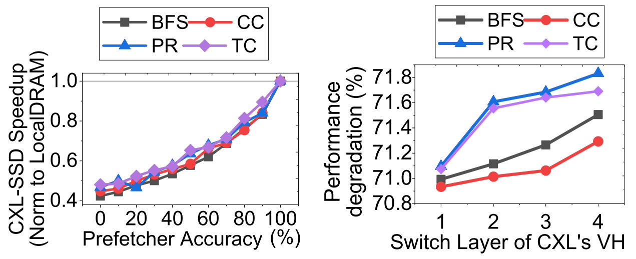
Protocol primary. CXL is a cache-coherent interconnect for heterogeneous devices, enabling scalable memory expansion. CXL comprises three sub-protocols: *CXL.io*, *CXL.cache*, and *CXL.mem*. Since CXL is built on the existing PCIe physical layer, CXL.io serves a functionally equivalent role to the PCIe protocol. CXL.cache is designed to allow accelerators to access host memory efficiently. Lastly, CXL.mem enables hosts to access device-attached memory anywhere within the corresponding CXL network. Consequently, CXL.io and CXL.mem are employed to connect multiple memory expanders and create a large-scale pool. Note that, even in the absence of CXL.cache, memory expanders can still be connected to the host's system memory, which appears to CPUs as if it were locally-attached memory. This is because CXL enables *endpoint* (EP) devices to be mapped in the cacheable memory space, whereas all PCIe devices must be connected to the noncacheable memory space [15].

Incorporating CXL into storage. CXL.mem and CXL.io allow CPU cache access to memory via load/store instructions, using a CXL message packet called a *flit*. This flit-based communication enables various memory and storage media to be integrated into the CXL pool. SCM has a greater capacity than DRAM, leading to interest in integrating CXL into block storage, known as *CXL-SSDs*. Though it is infeasible to combine all SCM features into one SSD technology, CXL-SSDs often use large internal DRAM caches. These caches store data ahead of backend SCMs, achieving performance akin to DRAM-based EP expanders. Samsung's PoC employs Z-NAND and a 16GB DRAM cache, claiming an 18× write latency improvement over NVMe SSDs due to cache size [4]. Kioxia's PoC uses XF-Flash and a sizable DRAM prefetch buffer, asserting DRAM-like speeds by combining prefetching with hardware compression [5].

2.2 Go Beyond Pooling

Enhanced memory coherence. CXL's flit-based communication effectively separates memory resources from processor complexes, enabling a practical memory pooling approach. However, CXL.mem lacks cache coherence capabilities, which are present in CXL.cache. CXL.cache unfortunately introduces significant overhead for CXL EPs in managing their internal memory space. For example, when a CXL-SSD utilizes CXL.cache for host-side cache updates, most memory accesses within internal DRAM and/or backend SCM media must be continuously monitored and approved by the host to ensure cache coherence. To tackle this issue, the recent implementation of *back-invalidation* (BI) in CXL 3.0 [16] allows CXL.mem to back-snoop the host's cache lines. This feature enables EPs to autonomously invalidate host cache lines, potentially allowing CXL-SSDs to maintain coherent memory states without relying entirely on CXL.cache.

Multi-tiered switching. Multiple EP expanders in a pool can be interconnected via one or more CXL switches. A CXL



(a) Accuracy sensitivity. **(b) Switch impact.**

Figure 2: CXL-SSD prefetching performance analysis.

switch comprises several *upstream ports* (USPs) and *downstream ports* (DSPs), with each group capable of connecting CPUs and EP expanders. A fabric manager (FM) can link and configure a switch's USPs and DSPs, allowing each host to access its own EP expanders through a unique data path called a *virtual hierarchy* (VH). Notably, CXL previously supported only a single switch layer, limiting the capacity of each VH. However, CXL 3.0 introduces *multi-tiered switching*, enabling switch ports to connect to other switches, significantly increasing the number of devices (up to 4K) in each VH with various CXL sub-protocols. This advancement substantially enhances resource disaggregation scalability.

3 MOTIVATION AND CHALLENGES

Locality impact. Figures 1a and 1b examine the latency of locally-attached DRAM (*LocalDRAM*) and CXL-SSDs at varying locality levels, respectively. We employ a global data access benchmark [17] to synthetically alter locality levels and assess their effects. In this analysis, L denotes vector-length, representing spatial locality, while α signifies temporal locality. An α value of 1 indicates minimal locality (pure random), with smaller values corresponding to higher temporal localities. This captures the range of possible performance and the influence of cache hierarchy. The test environment is consistent with all remaining analyses, detailed in §6.

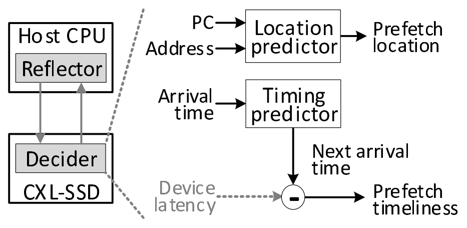
In low locality scenarios (α=0.1~1 and L=4~16), CXL-SSD performance is, on average, 738% slower than LocalDRAM. This significantly lower performance is considered unacceptable in many computing domains. However, as locality increases, CXL-SSD's average latency approaches that of LocalDRAM. Notably, high locality results in nearly identical average latency between the two. Specifically, when CXL-SSD operates with high locality ($\alpha \leq$0.01 and L$\geq$16), it is only 35% slower than LocalDRAM on average, which can be attributed to data being primarily loaded from LLC rather than backend SCMs. Interestingly, existing CXL-SSD research has overlooked this aspect, mainly focusing on optimizing internal DRAM cache usage. While utilizing internal DRAM caches is crucial for reducing long tail latency in cases of minimal locality, we argue that enhancing cache hit rates is vital for improving user experience with CXL-SSDs.

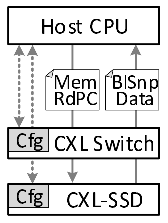
Impact of prefetching and latency variation. In Figure 2a, we evaluate latency speedup by integrating a synthesized prefetcher with varying accuracy, ranging from zero to perfect (ideal). For this analysis, we use four representative, large-scale graph workloads sourced from [18, 19]. The workload characteristics are further explained in §6. We calculate the speedup by averaging the CXL-SSD latency and normalizing it to LocalDRAM. The figure shows a consistent performance improvement as prefetcher accuracy increases, with CXL-SSD latency becoming nearly equivalent to LocalDRAM when accuracy exceeds 90%, on average. This occurs because prefetching retrieves data likely to be accessed in the near future, thereby increasing cache hit rates.

However, even an oracle prefetcher boasting over 90% accuracy can't prevent performance degradation when multi-tiered switching induces CXL-SSD latency variation. Traditional prefetchers don't account for the added latency due to switches, leading to data unavailability at the point when the CPU needs it. Consequently, many memory requests, which should result in cache hits, turn into cache misses. This implies that the Oracle prefetcher's accuracy can't be maintained in the CXL system, causing performance to degrade as depicted in Figure 2b. As the switch layer deepens, latency in memory expander increases, making prefetching increasingly important and causing further degradation in performance. With each increment by 4 in the switch layer, there is a 1% decrease in performance. Thus, deeper switch layers would result in even more significant performance degradation.

4 EXPANDER-DRIVEN PREFETCHING

CPU-side cache prefetchers have been extensively studied and proposed within the computer architecture community [8–13]. Although numerous prefetcher designs exist, they can be classified into two main categories based on industry adoption: i) *spatial prefetchers* and ii) *temporal prefetchers*. Spatial prefetchers prefetch data from memory by adding an offset to the currently accessed address in advance [8–10]. During runtime, this offset is optimized to minimize cache miss rates. In contrast, temporal prefetchers record cache miss sequences and deliver data from locations where cache misses are likely to reoccur to the host cache lines [11–13]. These rule-based prefetchers have been integrated into various industrial processors, such as Intel Core2 [20] and Intel Golden Cove [14]. However, their accuracy is known to be limited (ranging from 9% to 76%) for large-scale, irregular, and random memory access patterns [21]. Such accuracies are inadequate to accelerate CXL-SSD performance to a level comparable with LocalDRAM (e.g., 90%).





(a) Architectural design. (b) Interface.

Figure 3: Expander-driven prefetcher (ExPAND).

To meet the accuracy requirement, we can explore sophisticated prefetchers augmented with deep learning or machine learning (ML) techniques [20–27]. Prefetching essentially involves prediction, making it plausible that ML-enhanced prefetchers would perform better. Although ML prefetchers have the potential to attain prediction accuracy similar to or exceeding the requirement, they are impractical for implementation within an on-chip CPU. This is due to the extensive storage space needed for model computation and metadata management within the CPU [28].

4.1 Prefetching Delegation and Collaboration

In this paper, we propose an expander-driven prefetcher (ExPAND) that delegates cache prefetching decisions to the underlying CXL-SSDs and allows them to autonomously update the CPU's cache lines. As the decision-making process occurs on the EP side, complex prefetching algorithms can be implemented, leveraging the larger form factor and computing power of SSD EPs compared to on-chip CPUs. Figure 3 illustrates ExPAND's architectural design and the end-to-end process of our cache prefetching. ExPAND comprises two main components: the *reflector* and the *decider*.

Specifically, the reflector is implemented in the host-side CXL root complex (RC) and LLC controller, while the decider is employed in the EP-side CXL-SSD controller. The primary function of the reflector is to supply the decider with essential information for decision-making, such as the program counter (PC) and the switch depth where the CXL-SSD is located, and to relay cache prefetching results determined by the decider. To achieve this, the reflector uses a small buffer (16 KB) that logs cache line updates prefetched by the underlying deciders. It also ensures that each host's LLC controller within the CXL network first checks the reflector's buffer. If any data is already present in the CXL RC, the LLC controller directly serves the data from the buffer, bypassing the need to traverse the entire CXL-SSD pool. On the other hand, the decider in this paper employs a heterogeneous ML prefetcher, which is highly optimized for irregular memory accesses with random patterns [21]. Using the provided information (PC and memory address), each CXL-SSD decider determines the appropriate prefetch location and transfers the data from the speculated location to the reflector buffer. ExPAND's decider also maintains a record of the provided information for online fine-tuning of prefetching patterns. The details of its prefetcher and interaction methods between the reflector and decider will be explained in §4.2 and §5, respectively.

4.2 Prefetch Address and Timing Speculation

ExPAND's decider features two distinct predictors, each responsible for determining the location and timing of cache prefetch operations. The former is inspired by a multi-modality transformer model [21], while the latter consists of a simple rule-based predictor. Specifically, the address predictor requires generating a sequence of memory addresses for prefetching, using a transformer as its sequence model. In addition, it employs a multi-modality attention network [29] to enhance the understanding of the relationship between PC and memory access patterns. To accurately prefetch data, the address predictor also monitors changes in an application's execution behavior with a decision tree classifier [30].

On the other hand, the timing predictor maintains request arrival time information within its buffer (80B) and estimates a future memory request time by averaging past arrival times. To predict the next arrival time, all the past arrival times in its history window must be present in the buffer. However, relevant information may be missed if the memory requests are served by the LLC. In this case, the reflector communicates the cache hit event to the decider via CXL.io, allowing the timing predictor to precisely determine the future request time, even when no request is observed due to a cache hit. Note that the actual prefetch timeliness is calculated by considering each device's latency variation, combined with the results from the timing predictor. A detailed explanation of this prefetch timeliness estimation can be found in §5.1.

5 CROSS-LAYER INTERACTION ON CXL

5.1 CXL Topology-Aware Prefetch Timeliness

CXL switch hierarchy discovery. The reflector effectively identifies the switch level of individual CXL-SSDs during the PCIe enumeration process. The host accesses the configuration space of all connected devices (using CXL.io) and organizes the system buses within their CXL network. Notably, during PCIe enumeration, buses are segregated upon identifying each new device, and a unique number is assigned to every bus. Since a CXL switch operates as a PCIe bridge device with its own bus number, it allows determining the number of switches between a host CPU and a target CXL-SSD device. The reflector stores this information on its RC side, which aids in estimating accurate prefetch timeliness, discussed in more detail below.

Timeliness speculation. Prefetching data too early could contaminate the LLC, reducing its hit ratio, while prefetching too late may excessively delay execution. Therefore, pinpointing the exact prefetch timeliness is essential. As all CXL EPs must appropriately manage the *data object exchange* (DOE) capability in the PCIe configuration space [31], each CXL-SSD can determine its device latency by referring to DOE. However, this latency cannot be directly used for prefetch timeliness estimation due to latency variations caused by multi-tiered switching. During device enumeration, the reflector retrieves each CXL-SSD's latency by extracting the *device scoped latency and bandwidth information structure* (DSLBIS) from the DOE. It then calculates the latency overhead incurred in VH between the RC and target CXL-SSD. The reflector combines this VH latency with the DSLBIS latency and stores the end-to-end latency in the corresponding device's configuration space. Consequently, the decider can estimate prefetch timeliness by subtracting the end-to-end latency from the time predicted by its timing predictor.

5.2 Bidirectional Communication on CXL

Downward: piggybacking on CXL.mem. To accurately predict addresses, timely transmission of PCs and corresponding memory requests is vital. CXL.mem's master-to-subordinate (*M2S*) transactions include request without data (*Req*), request with data (*RwD*), and back-invalidation response (*BIRsp*). Req is primarily for memory read opcode (*MemRd*) without payload, while RwD carries payload for memory write opcode (*MemWr*). RwD allows 13 custom opcodes, enabling an opcode for memory reads with PCs (*MemRdPC*). When a read misses the LLC, the reflector sends an M2S transaction using MemRdPC, including the current PC. Consequently, the target decider can access the memory address and PC in the host's execution environment. Note that BIRsp responds to the CXL-SSD's BI snoop command, discussed shortly.

Upward: leveraging BI. When it reaches the time to prefetch (estimated in §4.2), the decider must update the reflector buffer with data obtained from its address predictor results. However, the existing CXL.mem lacks the capability to update host-side on-chip storage. To address this limitation, we use the CXL.mem subordinate-to-master (S2M) transaction's BISnp. BISnp, similar to CXL.mem's Req, is a non-payload message. We introduce a new BI opcode, termed *BISnpData*, to the S2M transaction message, allowing up to 10 custom opcodes [32]. Using BISnpData, the decider generates a payload accompanying its message, containing data for updating the host. When the reflector detects BISnpData, it awaits the corresponding payload and inserts the prefetched data into its buffer, enabling the LLC controller to fetch it for execution.

6 EVALUATION

Workloads and methodologies. We test ExPAND using four graph algorithms: Breadth-First Search (BFS), Connected Components (CC), PageRank (PR), and Triangle Counting (TC). These algorithms come from well-known graph frameworks [18, 19]. To evaluate ExPAND's utility in real-world situations, we use Amazon's product co-purchasing network as our dataset [33]. Since no CXL-SSDs are currently available, we use CXL hardware RTL modules in a full system simulation model. We conduct this simulation using gem5 [34] and SimpleSSD [35]. These CXL RTL modules have been validated using a real CXL end-to-end system at the cycle-level [36]. Table 1 provides the main parameters of our simulation. We compare ExPAND, our proposed expander-driven prefetcher, with several modern rule-based and ML-based prefetchers. These include a spatial prefetcher [8] (Rule1), a temporal prefetcher [11] (Rule2), an LSTM-based prefetcher [28] (ML1), and a transformer-based prefetcher [27] (ML2).

CPU	OoO 4cores@4GHz 224-entry ROB, 56-entry LSQ
L1 I-cache	32KB 2-way, 2-cycle latency
L2 D-cache	64KB 2-way 2-cycle latency
L2 Cache	1MB 8-way 20-cycle latency
DRAM	tRP = tRCD = tCAS = 14.16ns, 8 Rank, 8 Bank, 2channel,

(a) Host system.

NAND Flash	Read: 3μs

DRAM Cache	tRP = tRCD = 9.1ns, tRAS=19ns,
	size = 1.5GB

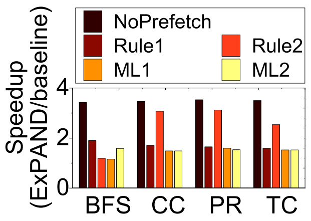
Location Predictor	Attention dimension: 64 Modality fusion dimension: 128 Transformer dimension: 128
Timing	Buffer entries: 10

(b) CXL-SSD with ExPAND.

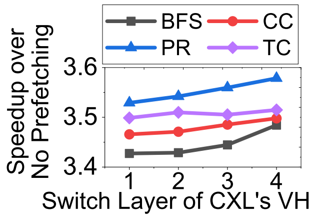
Table 1: Simulation setup.

Impact of Prefetching. We conduct an investigation into the performance influence and the distinct contribution of our expander-driven CXL prefetcher, ExPAND. The focus of our experiment is on quantifying the acceleration provided by ExPAND in relation to other baseline prefetchers. Figure 4a indicates that ExPAND imparts a speedup of 3.5× relative to the non-prefetching baseline (NoPrefetch). The average speedup metrics are derived using the geometric mean from all the graph algorithms. Beyond the improvement over NoPrefetch, ExPAND also facilitates speedups of 2.1× for rule-based prefetchers and 1.5× for basic ML-based prefetchers. The rule-based temporal prefetcher (Rule2) shows superior prediction accuracy for the BFS algorithm. However, other graph algorithms do not significantly benefit from Rule2 in terms of enhancing application performance. This can be attributed to the fact that the BFS algorithm, as a graph traversal algorithm, necessitates accessing child node data from previously navigated parent nodes. Therefore, a record-and-replay-based temporal prefetcher like Rule2 is well-matched to BFS.

Impacts of Timeliness. Figure 4b illustrates the influence of our CXL topology-aware prefetch timeliness across all graph algorithms. We conduct sensitivity analysis by gradually augmenting the switch level of CXL's VH from 1 to 4. This maximum switch level is decided based on the optimal configuration that takes into account the highest feasible placement of memory expanders in a balanced tree structure. For this experiment, we juxtapose the effectiveness of ExPAND prefetching with a non-prefetching baseline. The



(a) Prefetching techniques.



(b) Timeliness.

Figure 4: Execution time speedup.

goal is to understand the impact of not having a practical prefetching implementation on the CPU side. As shown in Figure 4b, ExPAND delivers an average speedup of 4.1x, even with a deeper switch layer (with VH levels from 2 to 4). This outcome can be traced back to ExPAND's proficiency in prefetching data location efficiently, which in turn, enhances overall performance.

7 CONCLUSION

We propose an expander-driven CXL prefetcher that offloads LLC prefetching to CXL-SSDs, employing a heterogeneous prediction algorithm. ExPAND ensures data consistency and provides precise prefetch timeliness estimates, reducing CXL-SSD reliance and enhancing graph application performance by 3.5× compared to other prefetching strategies.

ACKNOWLEDGEMENTS

The authors thank anonymous reviewers for their constructive feedback. This work is protected by one or more patents. Myoungsoo Jung is the corresponding author (mj@camelab.org).

REFERENCES

[1] JH Oh, Jae Hyo Park, YS Lim, HS Lim, YT Oh, Jong Soo Kim, JM Shin, Young Jun Song, KC Ryoo, DW Lim, et al. Full integration of highly manufacturable 512mb pram based on 90nm technology. In *2006 International Electron Devices Meeting*, pages 1–4. IEEE, 2006.

[2] Wooseong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, Chulseung Lee, Youra Choi, Shine Kim, Dongku Kang, Geunyeong Yu, et al. A flash memory controller for 15μs ultra-low-latency ssd using high-speed 3d nand flash with 3μs read time. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 338–340. IEEE, 2018.

[3] Toshiyuki Kouchi, Noriyasu Kumazaki, Masashi Yamaoka, Sanad Bushnaq, Takuyo Kodama, Yuki Ishizaki, Yoko Deguchi, Akio Sugahara, Akihiro Imamoto, Norichika Asaoka, et al. 13.5 a 128gb 1b/cell 96-word-line-layer 3d flash memory to improve random read latency with t prog= 75μs and t r= 4μs. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 226–228. IEEE, 2020.

[4] Samsung Semiconductor. Samsung electronics unveils far-reaching, next-generation memory solutions at flash memory summit 2022, 2022.

[5] Kioxia Corporation. Kioxia launches second generation of high-performance, cost-effective xl-flash™ storage class memory solution, 2022.

[6] Myoungsoo Jung. Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd). In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, pages 45–51, 2022.

[7] Yiying Zhang and Steven Swanson. A study of application performance with non-volatile main memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2015.

[8] Pierre Michaud. Best-offset hardware prefetching. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 469–480. IEEE, 2016.

[9] Rahul Bera, Anant V Nori, Onur Mutlu, and Sreenivas Subramoney. Dspatch: Dual spatial pattern prefetcher. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 531–544, 2019.

[10] Mohammad Bakhshalipour, Mehran Shakerinava, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Bingo spatial data prefetcher. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 399–411. IEEE, 2019.

[11] Akanksha Jain and Calvin Lin. Linearizing irregular memory accesses for improved correlated prefetching. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 247–259, 2013.

[12] Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Domino temporal data prefetcher. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 131–142. IEEE, 2018.

[13] Stephen Somogyi, Thomas F Wenisch, Anastasia Ailamaki, and Babak Falsafi. Spatio-temporal memory streaming. *ACM SIGARCH Computer Architecture News*, 37(3):69–80, 2009.

[14] Part Guide. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, 2(11), 2011.

[15] Chander Chadha. Nvme ssds with persistent memory regions. 2018.

[16] CXL Consortium. Compute express link 3.0 specification. pages 128–130, 2022.

[17] Erich Strohmaier and Hongzhang Shan. Apex-map: A global data access benchmark to analyze hpc systems and parallel programming paradigms. In *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 49–49. IEEE, 2005.

[18] Kartik Lakhotia, Rajgopal Kannan, Sourav Pati, and Viktor Prasanna. Gpop: A scalable cache-and memory-efficient framework for graph processing over parts. *ACM Transactions on Parallel Computing (TOPC)*, 7(1):1–24, 2020.

[19] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 17–30, 2012.

[20] Saami Rahman, Martin Burtscher, Ziliang Zong, and Apan Qasem. Maximizing hardware prefetch effectiveness with machine learning. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 383–389. IEEE, 2015.

[21] Pengmiao Zhang, Rajgopal Kannan, and Viktor K Prasanna. Phases, modalities, temporal and spatial locality: Domain specific ml prefetcher for accelerating graph analytics. *arXiv preprint arXiv:2212.05250*, 2022.

[22] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns. In *International Conference on Machine Learning*, pages 1919–1928. PMLR, 2018.

[23] Peter Braun and Heiner Litz. Understanding memory access patterns for prefetching. In *International Workshop on AI-assisted Design for*

Architecture (AIDArc), held in conjunction with ISCA, 2019.

[24] Ajitesh Srivastava, Ta-Yang Wang, Pengmiao Zhang, Cesar Augusto F De Rose, Rajgopal Kannan, and Viktor K Prasanna. Memmap: Compact and generalizable meta-lstm models for memory access prediction. In *Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part II 24*, pages 57–68. Springer, 2020.

[25] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*, pages 48–53, 2018.

[26] Pengmiao Zhang, Ajitesh Srivastava, Anant V Nori, Rajgopal Kannan, and Viktor K Prasanna. Transformap: Transformer for memory access prediction. *arXiv preprint arXiv:2205.14778*, 2022.

[27] Pengmiao Zhang, Ajitesh Srivastava, Anant V Nori, Rajgopal Kannan, and Viktor K Prasanna. Fine-grained address segmentation for attention-based variable-degree prefetching. In *Proceedings of the 19th ACM International Conference on Computing Frontiers*, pages 103–112, 2022.

[28] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. A hierarchical neural model of data prefetching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 861–873, 2021.

[29] Carey Jewitt, Jeff Bezemer, and Kay O'Halloran. *Introducing multimodality*. Routledge, 2016.

[30] Anthony J Myles, Robert N Feudale, Yang Liu, Nathaniel A Woody, and Steven D Brown. An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 18(6):275–285, 2004.

[31] CXL Consortium. Compute express link 3.0 specification. pages 398–399, 2022.

[32] CXL Consortium. Compute express link 3.0 specification. page 129, 2022.

[33] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[34] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[35] Donghyun Gouk, Miryeong Kwon, Jie Zhang, Sungjoon Koh, Wonil Choi, Nam Sung Kim, Mahmut Kandemir, and Myoungsoo Jung. Amber: Enabling precise full-system simulation with detailed modeling of all ssd resources. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 469–481. IEEE, 2018.

[36] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access,{High-Performance} memory disaggregation with {DirectCXL}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 287–294, 2022.