CHALMERS
UNIVERSITY OF TECHNOLOGY

CrossFetch: A Prefetching Scheme for Cross-Page Prefetching in the Physical Address Space

CrossFetch: A Prefetching Scheme for Cross-Page Prefetching in the Physical Address Space

Qi Shao and Per Stenstrom, *Life Fellow, IEEE*

***Abstract*—Prefetching is an important technique to reduce the miss penalty in deep memory hierarchies, employing multiple levels of cache and memory. Unfortunately, state-of-the-art techniques avoid prefetching across page boundaries in physically addressed memory because contiguous virtual pages are not guaranteed to map to contiguous physical pages. Apart from low accuracy, prefetching across page boundaries can break protection domains, opening up security vulnerabilities. This paper proposes CrossFetch — the first prefetching technique that accurately and securely prefetches data across physical page boundaries. It uses a simple and novel translation mechanism that combines a conventional TLB, called forward TLB (FTLB), with a reverse TLB (RTLB) that provides mappings of physical pages to virtual. CrossFetch leverages a conventional Page Table Walker invoked by a conventional TLB to load mappings into the FTLB and RTLB. The paper demonstrates how CrossFetch can hide far-memory misses in hybrid main-memory systems and last-level cache misses. We show that CrossFetch can improve IPC by 5.7% (up to 27.7%) compared to intra-page prefetchers on SPEC2017 benchmarks where the tolerance of far-memory misses dominates.**

***Index Terms*—Prefetching, heterogeneous memory systems, hybrid memory, translation-lookaside buffer (TLB).**

I. Introduction

COMPUTER systems employ deep cache/memory hierarchies to bridge the performance gap between fast processors and slow storage. In recent years, main memory itself has evolved into a two-tier structure: a fast Near Memory (NM) tier and a slower Far Memory (FM) tier. Coherent Express Link (CXL) [1] is an example, where part of the memory is local and part of it is global. Another example, considered in this paper, pairs DRAM (NM) with Non-Volatile Memory, e.g., Phase Change Memory (FM) [2], [3], [4]. In such systems, a portion of NM is often reserved as a cache for FM, referred to as a *Near-Memory Cache* (NM cache) [5], [6], [7]. Such systems suffer from a significant *miss penalty* both concerning last-level cache (LLC) and NM cache misses.

Prefetching is a well-known technique to reduce miss penalties by predicting future accesses and fetching data before it is demanded. Prefetchers operate throughout the memory hierarchy, from L1 via L2 to LLCs and between NM and FM. However, existing prefetchers that use *physical addresses* (prefetching beyond L1) avoid prefetching across page boundaries [8], [9], [10], [11]. The reason is twofold: 1) *Accuracy:* Although virtual pages are typically accessed contiguously, their physical mappings are often non-contiguous due to virtual memory management. Naïve cross-page prefetching may therefore fetch data that is not accessed. 2) *Security:* Prefetching across pages in the physical address space might cross different access permissions which risks violating protection domains. As a result, prefetchers conservatively restrict themselves to intra-page locality, leaving significant spatial locality unexploited.

Recent work has shown that *cross-page prefetching* is feasible at the L1 cache level, where the TLB provides information on virtual-to-physical mappings [12], [13]. However, extending this capability to lower levels, such as the LLC or NM cache, remains an open and challenging problem. These memories operate in the physical address space and lack visibility into the virtual to physical page mapping, preventing them from exploiting cross-page locality. Consequently, today's LLC and NM caches are fundamentally *limited to intra-page prefetching,* missing important optimization opportunities.

This paper proposes CrossFetch, a simple and novel hardware extension that, for the first time, enables secure and accurate cross-page prefetching in physically addressed memory systems. CrossFetch bridges the virtual and physical views of memory using a bidirectional translation structure: a conventional *forward TLB (FTLB)* for virtual-to-physical page mappings and a *reverse TLB (RTLB)* for physical-to-virtual page mappings. It interacts with existing Page Table Walkers to provide translation capability with minimal overhead. Knowing the physical address of the next page, the paper shows that CrossFetch can prefetch metadata, which tracks the placement of far-memory (FM) data within the near-memory (NM) cache. It also shows how CrossFetch can proactively cache superblocks ahead of demand misses and inform the LLC to prefetch the corresponding blocks in advance.

This paper makes the following contributions:

- We propose CrossFetch, the first scheme to enable accurate and secure cross-page prefetching at physically addressed lower levels of the memory hierarchy (e.g., NM caches and the LLC).
- CrossFetch builds on our proposal of a new address translation mechanism that combines a conventional TLB for virtual-to-physical page mappings with a reverse TLB for physical-to-virtual page mappings. We show how it can be integrated in conventional virtual memory managers by interacting with the Page Table walker, which opens up for accurate and secure cross-page prefetching.

Received 13 November 2025; revised 1 December 2025; accepted 3 December 2025. Date of publication 8 December 2025; date of current version 31 December 2025. This work was supported in part by Dr. Angelos Arelakis for his feedback, in part by Wallenberg WASP program, in part by the Swedish Foundation for Strategic Research under Grant CHI19-0048, and in part by National Academic Infrastructure for Supercomputing in Sweden under Grant 2022-06725. *(Corresponding author: Per Stenstrom.)*

The authors are with the Computer Science and Engineering, Chalmers University of Technology, Gothenburg SE 412 96, Sweden (e-mail: qisha@chalmers.se; per.stenstrom@chalmers.se).

Digital Object Identifier 10.1109/LCA.2025.3640965

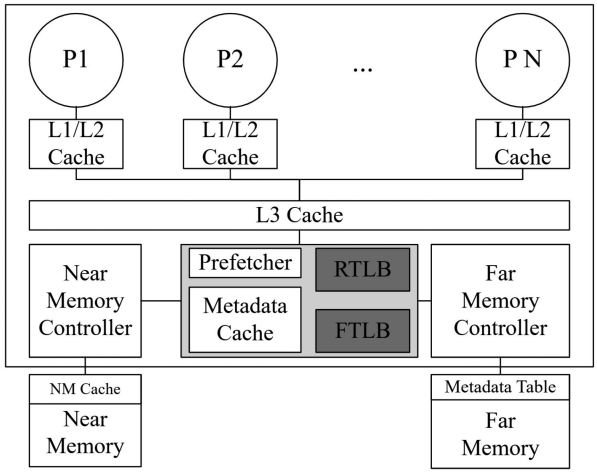

Fig. 1. Baseline hardware-managed NM cache with CrossFetch extension (dark gray).

- We experimentally evaluate CrossFetch and find that it effectively exploits the locality between pages, improving IPC by 5.7% (up to 27.7%). on SPEC2017 benchmarks compared to intra-page prefetchers.

The rest of the paper is organized as follows. Section II presents CrossFetch. Sections III and IV present the methodology and results, respectively. Finally, we conclude in Section V.

II. CrossFetch: Prefetching Across Page Boundaries in Physical Memory Space

A. Baseline System

Fig. 1 shows the baseline system with the CrossFetch extension. Main memory is organized into two tiers: Near Memory (NM) and Far Memory (FM). Without loss of generality, NM is DRAM and FM is phase-change memory (PCM), although other memory technologies could be considered. Because PCM incurs roughly 4× higher latency than DRAM [14], part of NM is dedicated to a cache for FM data. An LLC request to FM may thus hit the NM cache or access FM directly.

To manage the two-tier memory hierarchy, the system maintains metadata that track the placement of FM blocks in NM. Tracking data at cache block granularity (64B) is too costly, while page-level caching leads to overfetching. Therefore, NM caches [5], [15] strike a compromise and adopt *superblock granularity* (256B–2KB), balancing metadata overhead and flexibility of placement. Recently accessed metadata are cached on chip in a metadata cache (Fig. 1).

Metadata provides information where (in FM or in the NM cache) to find all superblocks in a page (4 KB), yielding eight superblocks of 512B per page. On a metadata miss, the metadata for the entire page is fetched from FM. The metadata for each superblock comprises a *cached* status bit and the *location in the NM cache*. If the status bit is set, the NM cache is accessed with the location. Otherwise, the request is forwarded to FM where the entire metadata table is provided.

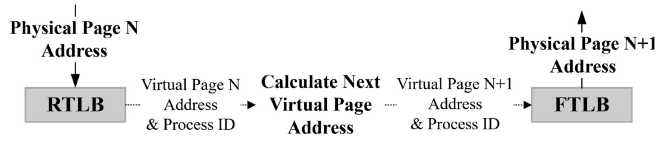

Fig. 2. CrossFetch translation flow: Current physical page to next virtual page's physical location.

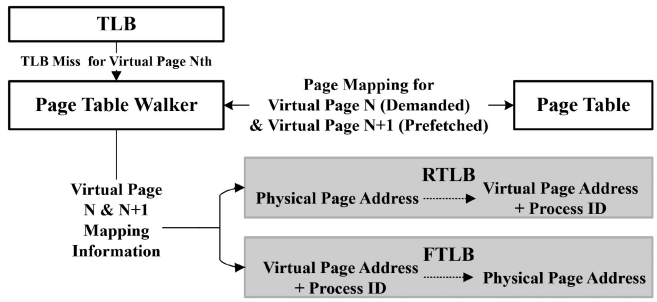

Fig. 3. CrossFetch interacts with the hardware Page Table Walker to populate the RTLB and the FTLB.

B. CrossFetch: Operation and Implementation

The objective of CrossFetch is to support next-virtual-page prefetching *in the physical address space* by providing page information to establish the next physical page address along with protection information. We show that this offers accurate and secure prefetching.

CrossFetch offers a simple and novel mechanism that enables accurate and secure cross-page prefetching in the physical address space. It comprises two structures (see Fig. 1): a reverse TLB (RTLB) maps physical pages to virtual pages (along with the process ID), and a forward TLB (FTLB) maps virtual pages to physical pages like a conventional TLB. Together, these structures provide the bidirectional translation required for physical cross-page prefetching.

Fig. 2 shows the operation of CrossFetch using the structures above. For an LLC access, CrossFetch extracts the physical page number, queries the RTLB to obtain the virtual page number and process ID, increments the virtual page number to identify the next page, and queries the FTLB to obtain the corresponding physical page number. On an FTLB or RTLB miss, CrossFetch conservatively halts cross-page prefetching. CrossFetch integrates naturally with virtual page management and must adhere to flush and invalidate operations initiated by CPU TLBs. Hence, they work in synchronicity with CPU TLBs. CrossFetch ensures security by storing access-permission bits in the FTLB. Prefetching proceeds *only if the next page shares identical permissions with the current one*. This also prevents attempts to train the prefetcher to prefetch blocks from pages belonging to an adversary as the permissions will differ.

CrossFetch avoids extra page table lookups for updating the RTLB and FTLB by being interfaced to the hardware Page Table Walker (Fig. 3). On a CPU TLB miss, the Page Table Walker retrieves the virtual-to-physical mapping and forwards it

to CrossFetch, which updates the RTLB and FTLB entries. This requires memory-mapped register interfaces in CrossFetch. The Page Table Walker also must be extended to provide the physical address of the next virtual page. This is done with a next-page table entry prefetcher.

CrossFetch also incorporates cross-page prefetching to tolerate cold-start latency and improve coverage. On a metadata-cache miss, CrossFetch checks whether the metadata of the next page is already cached; if not, it proactively prefetches it to ensure that subsequent accesses hit in the metadata cache. Likewise, when the last superblock of a page is accessed, CrossFetch verifies whether the first superblock of the next page is present in NM and, if absent, CrossFetch prefetches it to overlap the cold-start miss. In this way, prefetching metadata and superblocks for the next page follows a mechanism analogous to a next-line cache prefetcher. CrossFetch also supports LLC prefetching of blocks contained in a cross-page prefetched superblock. By default, CrossFetch prefetches half of the blocks.

To mitigate the risk of fetching useless blocks, CrossFetch employs adaptively cross-page prefetching, inspired by prior adaptive prefetching mechanisms. Concretely, the aggressiveness is then tuned dynamically based on the *utilization ratio*, defined as the fraction of cross-page prefetched blocks that are later accessed by the CPU [8], [9], [16], [17]. Each block uses two bits to determine whether the prefetched block has been used: The first bit is set when the block is cross-page prefetched and the other when the block is accessed in the LLC.

Modern operating systems support larger page sizes, such as 2 MB. The performance of CrossFetch remains largely insensitive to page size, as it operates in the physical address space. However, in the near-memory cache, awareness of page size helps reduce metadata fetches from far memory, slightly improving efficiency. Sequential prefetching behaves identically, since inter-page prefetching at 4 KB effectively becomes intra-page prefetching with larger pages. Overall, CrossFetch is expected to deliver comparable or slightly improved performance with larger page sizes without requiring any design modifications.

III. Experimental Methodology

Simulation setup: We evaluate CrossFetch using Gem5 [18]. The simulation setup is shown in Table I. The timing of NM is based on the DDR4-2400 and FM is based on the PCM-NVM. For prefetching in the NM cache, we use a simple next-superblock prefetcher although other prefetch mechanisms could be used. A 64-MB NM cache is considered to cache LLC requests to FM, based on demand.

Benchmarks: We evaluate CrossFetch using the SPEC2017 benchmark suite, excluding the workloads `roms` and `omnetpp`, which fail to execute correctly on our Gem5 setup.

Models: We compare three simulation models, all of which employ sequential prefetching across the evaluated systems:

- *Baseline 1 (BL1)*: L1/LLC cache with intra-page prefetching only using state of the art prefetchers (Berti [10] for L1 and Signature Path Prefetching (SPP) [11] for LLC). No NM cache cross-page prefetching.

TABLE I
Simulator Configuration

Cores	4 cores, out-of-order, ARM, 3.2GHz
L1 Cache	Private, 64KB, 4-way, 4 cycle access latency
L2 Cache	Private, 512KB, 8-way, 6 cycle access latency
L3 Cache	Shared, 8MB, 8-way, 12 cycle access latency
Metadata Cache	512 entries, 4-way, 2 cycle access latency
Prefetcher	Next-superblock prefetcher
RTLB/FTLB	512 entries, 4-way, 2 cycle access latency
Near Memory	DDR4-2400, 1GB, 64-bit channel, 8 banks, tRCD=14.16ns, tCL=14.16ns, tRP=14.16ns
Far Memory	NVM-2400, 4GB, 64-bit channel, 8 banks, tREAD=150ns, tWRITE=500ns, tSEND=14.16ns, tBURST=3.332ns

- *Baseline 2 (BL2)*: Extends BL1 by prefetching not only the next page metadata but also the first superblock of the next page into the NM cache.
- *Baseline 3 (BL3)*: Extends BL1 by prefetching following next blocks of the next page into the LLC without adaptiveness.
- *Baseline 4 (BL4)*: Extends BL1 with adaptive cross-page prefetching in the LLC (detailed in Section II-B).
- *CrossFetch*: Extends BL1 with all three enhancements: prefetching the next page's metadata and superblock into the NM cache, adaptive and footprint-based cross-page prefetching in the LLC.

The comparison between BL2 and BL1 isolates the benefit of metadata and superblock prefetching, which removes the cold-start latency in the NM cache. The comparison between BL3 and BL4 quantifies the advantage of adaptiveness prefetching compared to non-adaptive prefetching in the LLC. The comparison between BL4 and BL1 isolates the benefit of adaptive cross-page prefetching by itself.

IV. Experimental Results

Section IV-A analyzes the performance improvement of CrossFetch relative to the baseline systems, while Section IV-B provides sensitivity analysis of RTLB/FTLB sizes.

A. Performance Improvement

Fig. 4 presents the IPC improvements of BL2, BL3, BL4 and CrossFetch over BL1. CrossFetch delivers up to 27.7% speedup, with an average IPC gain of 5.74%. BL2 demonstrates the benefit of prefetching next-page metadata and superblocks into the NM cache, improving IPC by 5.1% (up to 24.5%). Workloads such as *blender*, *cactuBSSN* gain more than 5% due to reduced cold-start misses in metadata and the NM cache. BL3 and BL4 provide IPC improvements of 0.85% and 1.2%, respectively.

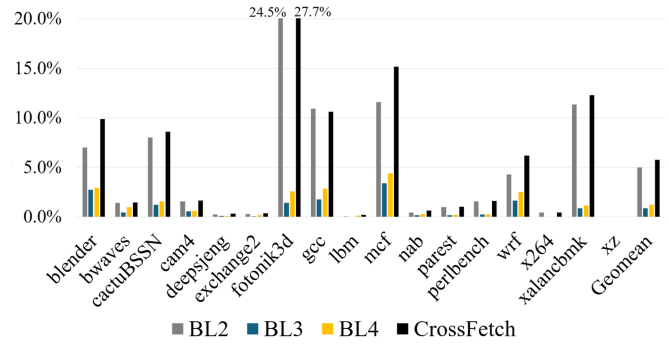

Fig. 4. IPC improvement of BL2, BL3, BL4 and CrossFetch relative to BL1.

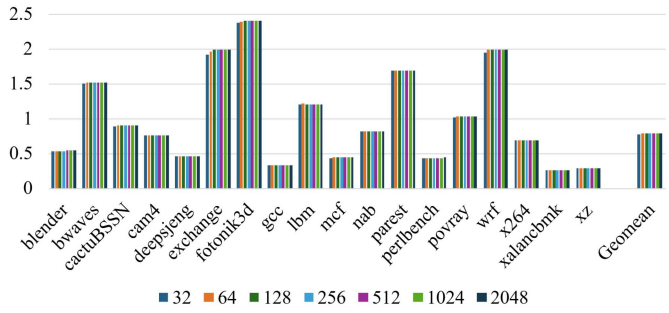

Fig. 5. IPC sensitivity to RTLB/FTLB size.

By integrating all three prefetching mechanisms, CrossFetch consistently achieves the highest performance across workloads.

B. Sensitivity Study

We next evaluate the sensitivity of CrossFetch to the number of RTLB/FTLB entries between 32 and 2048 entries. Because RTLB and FTLB maintain a bidirectional mapping, their sizes are identical.

As shown in Fig. 5 IPC is virtually unaffected by the size of the RTLB/FTLB. To balance performance gains with hardware overhead, we recommend 256 entries for both RTLB and FTLB in our system. However, the number of RTLB/FTLB entries scales with the number of cores in the system. As shown in Fig. 5, IPC is insentitive to the size of RTLB. The insensitivity of IPC to RTLB size arises from our update mechanism: both RTLB and FTLB are synchronized with the CPU TLB. Whenever a new or frequently accessed page table entry is updated or stored in the CPU TLB, it is also propagated to the RTLB and FTLB.

V. Conclusion

We present CrossFetch, the first prefetching technique that accurately and securely prefetches across physical page boundaries. It uses a translation mechanism that combines a conventional TLB (FTLB) with a reverse TLB (RTLB) that provides mappings of physical pages to virtual. CrossFetch leverages the Page Table Walker invoked by the conventional TLB to load mappings into FTLB and RTLB. CrossFetch improves performance by 5.7% (up to 27.7%).

References

[1] D. D. Sharma, R. Blankenship, and D. Berger, "An introduction to the compute express link (CXL) interconnect," *ACM Comput. Surveys*, vol. 56, no. 11, pp. 1–37, 2024, doi: 10.1145/36699.

[2] R. Salkhordeh and H. Asadi, "An operating system level data migration scheme in hybrid DRAM-NVM memory architecture," in *Proc. Des. Automat. Test Europe Conf. Exhib.*, 2016, pp. 936–941.

[3] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, "Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using Intel Optane DC persistent memory modules," in *Proc. Int. Symp. Memory Syst.*, 2019, pp. 288–303, doi: 10.1145/3357526.3357541.

[4] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Software-managed energy-efficient hybrid DRAM/NVM main memory," in *Proc. 12th ACM Int. Conf. Comput. Front.*, 2015, Art. no. 23, doi: 10.1145/2742854.2742886.

[5] Q. Shao, A. Arelakis, and P. Stenström, "HMComp: Extending near-memory capacity using compression in hybrid memory," in *Proc. 38th ACM Int. Conf. Supercomputing*, 2024, pp. 74–84, doi: 10.1145/3650200.3656612.

[6] G. Loh and M. D. Hill, "Supporting very large DRAM caches with compound-access scheduling and MissMap," *IEEE Micro*, vol. 32, no. 3, pp. 70–78, May/Jun. 2012.

[7] Y. Kim, H. Kim, and W. J. Song, "NOMAD: Enabling non-blocking OS-managed DRAM cache via tag-data decoupling," in *Proc. 2023 IEEE Int. Symp. High- Perform. Comput. Archit.*, 2023, pp. 193–205, doi: 10.1109/HPCA56546.2023.10071016.

[8] P. Michaud, "Best-offset hardware prefetching," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2016, pp. 469–480.

[9] Y. Cui, W. Chen, X. Cheng, and J. Yi, "Hyperion: A highly effective page and PC based delta prefetcher," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 4, Nov. 2024, Art. no. 68, doi: 10.1145/3675398.

[10] A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibáñez, V. Viñals-Yúfera, and A. Ros, "Berti: An accurate local-delta data prefetcher," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture*, 2022, pp. 975–991, doi: 10.1109/MICRO56248.2022.00072.

[11] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–12.

[12] G. Vavouliotis, M. Torrents, B. Grot, K. Kalaitzidis, L. Peled, and M. Casas, "To cross, or not to cross pages for prefetching?," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2025, pp. 188–203, doi: 10.1109/HPCA61900.2025.00025.

[13] G. Vavouliotis, G. Chacon, L. Alvarez, P. V. Gratz, D. A. Jiménez, and M. Casas, "Page size aware cache prefetching," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture*, 2022, pp. 956–974, doi: 10.1109/MICRO56248.2022.00070.

[14] G. Psaropoulos, I. Oukid, T. Legler, N. May, and A. Ailamaki, "Bridging the latency gap between NVM and DRAM for latency-bound operations," in *Proc. 15th Int. Workshop Data Manage. New Hardware*, 2019, Art. no. 13, doi: 10.1145/3329785.3329917.

[15] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked DRAM cache," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 25–37.

[16] I. Hur and C. Lin, "Memory prefetching using adaptive stream detection," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2006, pp. 397–408.

[17] B. S. Gill and L. A. D. Bathen, "AMP: Adaptive multi-stream prefetching in a shared cache," in *Proc. 5th USENIX Conf. File Storage Technol.*, 2007, Art. no. 26.

[18] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.