HYDRA: Hybrid DRAM Caching for Dynamic Complementary Multi-to-One Page Mapping

Gangqi Huang
Computer Science Engineering
University of California, Santa Cruz
Santa Cruz, CA, USA
ghuang49@ucsc.edu

Heiner Litz
Computer Science Engineering
University of California, Santa Cruz
Santa Cruz, CA, USA
hlitz@ucsc.edu

Yuanchao Xu
Computer Science Engineering
University of California, Santa Cruz
Santa Cruz, CA, USA
yxu314@ucsc.edu

***Abstract*—CXL-enabled tiered memory combines high-capacity remote memory with limited host-side local DRAM caches. However, existing DRAM caching designs underutilize local DRAM due to coarse page-granularity management and intra-page sparsity, where only a subset of each page is frequently accessed. We observe that different pages often have complementary intra-page access patterns, enabling multiple remote pages to share one local DRAM cache page. Supporting such dynamic multi-to-one mapping is challenging because PT-based DRAM caches enforce dynamic one-to-one mappings, while tag-based designs rely on static mappings.**

We propose HYDRA, a DRAM caching architecture that enables dynamic complementary multi-to-one page mapping. HYDRA combines tag-based and page-table-based caching with an intermediate physical address space to support efficient multi-page sharing with low hardware overhead. It further introduces device-side cache management to learn intra-page access patterns and revoke ineffective mappings. Across memory-intensive workloads, HYDRA improves local DRAM cache utilization, reduces remote traffic, and outperforms prior DRAM caching designs.

***Index Terms*—DRAM cache, tiered memory, Compute eXpress Link (CXL)**

I. INTRODUCTION

Compute eXpress Link (CXL) [1]–[3] has emerged as a promising solution to address the growing memory demands of modern datacenter workloads, including large language models [4], high performance computing [5], and graph analytics [6]. CXL enables disaggregated tiered memory architectures that combine high performance local memory (e.g., DDR5 DRAM and HBM) in compute nodes with large capacity remote memory modules connected through high speed CXL links, providing increased memory capacity and higher aggregate bandwidth while maintaining cache line granularity access semantics. To efficiently leverage CXL tiered memory, prior studies propose **DRAM caching** [7]–[14], which organizes local memory as a cache for remote memory, allowing frequently accessed data to be served from local memory while infrequently accessed data resides in remote memory. This approach reduces remote memory traffic and improves overall system performance. Recent Intel commercial processors [7] further support hardware managed DRAM caching for CXL tiered memory systems.

To architect a portion of local DRAM as a local DRAM cache, DRAM cache designs must maintain mapping metadata that records whether a remote memory block is cached locally and identifies the corresponding location in local memory. Based on how they maintain the mapping and mapping metadata, prior DRAM cache studies can be broadly categorized into two types: **Tag-Based** DRAM caches and **Page-Table-Based** (PT-based) DRAM caches. Tag-based DRAM caches [8], [10], [11] typically operate at cacheline granularity and employ static multi-to-one direct or set-associative mappings between remote memory and local DRAM. These designs require additional per cache line tags to identify which remote memory blocks reside in the local DRAM cache. In contrast, PT-based DRAM caches [12]–[14] maintain mappings at page granularity using dynamic fully-associative one-to-one mappings between remote pages and local DRAM cache pages. Each local DRAM page is mapped to one remote memory page, with mapping information stored in Page Table Entries (PTEs), eliminating the need for per cache line tags. Cache replacement in PT-based designs is also performed at page granularity rather than cacheline granularity.

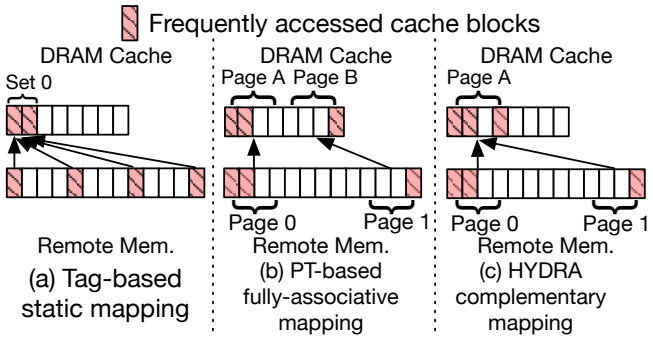

Fig. 1: Mapping illustration of (a) Tag-Based, (b) PT-Based, and (c) HYDRA Complementary Page Mapping DRAM Caching.

However, both tag-based and PT-based DRAM caching suffer from local DRAM cache underutilization due to different limitations. Tag-based DRAM caching employs static set-associative mappings (Figure 1(a)), which can lead to access imbalance across sets [9], [15], [16]. Heavily accessed sets experience frequent conflict evictions, while other sets remain underutilized. In contrast, PT-based DRAM caching uses fully

associative, page-granularity mapping and can utilize all local DRAM pages (Figure 1(b)). However, memory accesses often exhibit intra-page sparsity, where only a portion of each transferred page is accessed [9], [17], [18], resulting in intra-page underutilization.

We propose **HYDRA**, a DRAM caching mechanism based on **Dynamic Complementary Multi-to-One Page Mapping**. Our key observation is that different pages exhibit distinct intra-page sparsity patterns, meaning that accessed cacheline locations vary across pages. HYDRA identifies pages with complementary access locations and dynamically maps them to the same local DRAM page, improving intra-page utilization and reducing potential cache line conflicts. For example, as shown in Figure 1(c), only the first half of page 0 and the last quarter of page 1 are frequently accessed. HYDRA maps these complementary remote pages to local page A, improving intra-page utilization and reducing conflicts.

Efficiently supporting dynamic multi-to-one page mapping is challenging. Prior PT-based DRAM caches enforce one-to-one page mappings, while tag-based DRAM caches rely on static mappings, limiting their ability to dynamically map multiple complementary pages. A naïve extension of PT-based designs with per-cacheline tags requires additional metadata to track dynamic page-to-tag associations, as the tag-to-page mapping cannot be statically determined. This introduces extra lookup overhead and increases design complexity.

To address this challenge, we propose a **Hybrid Tag-based and PT-based DRAM Caching Design** using an **Intermediate Physical Address Space**. HYDRA maintains dynamic one-to-one page mappings from remote memory to intermediate physical addresses while applying static multi-to-one cacheline mapping from intermediate physical addresses to local DRAM cache pages. This design enables efficient dynamic multi-to-one page mapping without requiring the metadata about page-to-tag associations. Furthermore, HYDRA performs cacheline-level data movement and employs lightweight 4-bit tags per cacheline to track which page each cached cacheline in local DRAM belongs to, reducing both remote memory traffic and hardware overhead while efficiently supporting complementary page mapping.

To effectively realize this design, we introduce a device-side cache management mechanism that offloads learning and metadata management from the critical path and limited CPU-side resources. HYDRA dynamically learns intra-page access patterns to guide complementary mapping and employs conflict-aware revocation to adapt to evolving access behaviors. Together, these techniques enable efficient and practical dynamic complementary multi-to-one page mapping.

We implement HYDRA and other baselines on ChampSim [19], [20], a trace-based cycle-level simulator, integrated with the Ramulator 2.0 [21] DRAM simulator. Our evaluation on memory-intensive datacenter workloads shows that HYDRA provides, on average, 1.22x, 1.14x and 1.43x speedup over the state-of-the-art direct-mapped tag-based, set-associative tag-based and PT-based DRAM cache schemes, respectively.

Overall, this paper makes the following contributions:

- We identify an opportunity to improve local DRAM cache utilization by exploiting intra-page sparsity through **dynamic complementary multi-to-one page mapping**.
- We propose **HYDRA**, a hybrid tag-based and PT-based DRAM caching design using an **intermediate physical address space** to efficiently support dynamic multi-to-one mapping.
- We design a **device-side cache management mechanism** that learns access sparsity and performs conflict-aware mapping revocation.
- We evaluate HYDRA on memory-intensive workloads and demonstrate significant improvements over prior DRAM caches.

II. BACKGROUND AND PRIOR WORK

A. CXL Tiered Memory

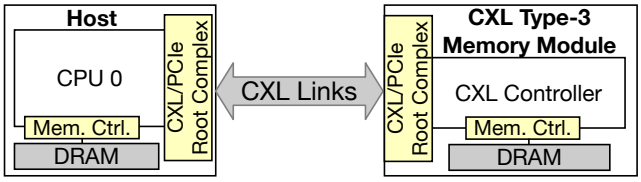

Fig. 2: CXL memory expansion.

CXL [1] enables high-speed, low-latency communication between host processors and CXL devices. Type-3 CXL devices expand host memory by exposing CXL memory as byte-addressable, cache-coherent memory [3], [8], [22], [23]. As shown in Figure 2, CXL memory accesses traverse the CXL Root Complex (RC) and CXL link, adding about 50–100 ns latency over local DRAM [3]. Existing systems expose CXL memory as a memory-only NUMA node [1], [24], so the OS manages it with the same page-table-based translation mechanism as local DRAM. Virtual pages can be mapped to either local DRAM or CXL physical pages, with accesses routed by physical address and remapping handled through PTE updates and TLB shootdowns.

B. DRAM Caching

DRAM caching manages heterogeneous memory systems by architecting local memory as a hardware-managed cache for remote memory. Prior industry [7], [8] and academic [9], [11]–[15], [25]–[31] designs differ in tag management, mapping associativity, granularity, and replacement policies, as summarized in Table I. Existing designs broadly fall into two categories: **Tag-Based** and **Page-Table-Based (PT-based)**. Tag-based DRAM caches employ static multi-to-one mappings with explicit tags, whereas PT-based designs use one-to-one fully associative dynamic page mapping by storing mapping information in PTEs.

1) Tag-Based DRAM Caches: Tag-based DRAM caches store per-cacheline tags in memory along with the data, and introduce a **DRAM Cache Controller (DCC)** to perform DRAM-cache-related operations such as tag parsing and comparison. Table I summarizes three representative types

TABLE I: Comparison with State-Of-The-Art (SOTA) DRAM Cache Designs.

Type	Mapping and Replacement (SOTA)	Local–Remote Bandwidth Efficiency	Local DRAM Cache Utilization	Storage & Hardware Overheads	DRAM Cache Miss Penalty
Tag-based	Static multi-to-one cacheline; direct-mapped (Intel FMM [7])	*high*	*low*	*low*	*low*
	Static multi-to-one cacheline; set-associative (NDC [15])	*high*	*medium*	*high*	*low*
	Static multi-to-one page; set-associative (Unison Cache [9])	*high*	*medium*	*medium*	*high*
PT-based	Dynamic one-to-one page; full-associative insert-on-fetch (Genie Cache [12])	*low*	*low*	*low*	*high*
HYDRA (ours)	Dynamic complementary multi-to-one page mapping with cacheline-granularity replacement; periodically adjusted complementary page mapping	*high*	*high*	*low*	*low*

of tag-based DRAM cache designs with different mapping granularities and associativities, each exposing distinct trade-offs in local-remote bandwidth efficiency, local DRAM cache utilization, hardware overhead, and miss penalty.

Cacheline-granularity, direct-mapped designs, such as Intel FMM[1] [7], [8], [16], use per-cacheline tags for fine-grained caching. Because only one cache block is transferred on a miss, they achieve high local–remote bandwidth efficiency and low miss penalty. However, static direct mapping introduces access conflicts, where frequently accessed data maps to a limited portion of the local DRAM cache, leading to under-utilization.

Cacheline-granularity, set-associative designs mitigate access conflict imbalance by allowing multiple cache blocks per set, improving utilization compared to direct-mapped designs. However, set associativity requires multiple tag accesses per lookup, increasing storage overhead, lookup latency, and bandwidth consumption. The State-Of-The-Art design (SOTA), Native DRAM Cache (NDC) [15], reduces tag-transfer overhead through in-DRAM tag matching, but introduces higher hardware complexity.

Page-granularity, set-associative designs, such as Unison Cache [9], further reduce tag storage overhead by caching at page granularity. These designs typically leverage footprint-based fetching [9], [18], which transfers only accessed cache lines rather than entire pages on replacement, preserving bandwidth efficiency. However, coarse-grained page-level management leads to intra-page underutilization, limited associativity and higher miss penalties compared to cacheline-granularity designs.

2) PT-based DRAM caches: To eliminate tag storage and lookup overheads, PT-based DRAM caches adopt fully associative dynamic one-to-one page mapping and store mapping information directly in the PFN fields of PTEs. Fully associative placement alleviates conflict imbalance and improves page-level utilization. However, page-granularity management becomes inefficient under sparse intra-page access patterns. When only a subset of cachelines within a page is accessed, fetching an entire page on a miss transfers unnecessary data, reducing bandwidth efficiency and effective local DRAM cache utilization while increasing miss penalties.

[1]We use Intel Flat Memory Mode (FMM) to refer to Intel's hardware tiering technique that stores per-line tags in unused ECC bits for direct-mapped DRAM caching. This technique is also known as Intel Memory Mode or Intel 2-Level Memory (2LM) in hybrid HBM-DRAM or DRAM-PMem systems, and Intel Flat Mode in CXL-DRAM systems [7], [8], [16], [32].

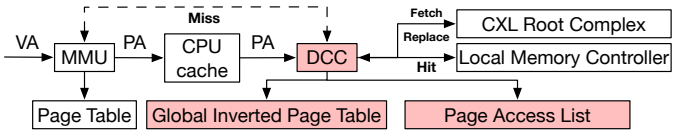

Fig. 3: Architecture of PT-Based DRAM caches.

3) Architectural Design: Figure 3 illustrates the architecture of PT-based DRAM caches [12]–[14], [33]. These designs typically consist of three key components: (1) a **DRAM Cache Controller (DCC)** that handles cache accesses and miss processing, (2) a **Global Inverted Page Table (GIPT)** that maintains inverted mappings from DRAM cache pages to remote memory pages, and (3) an **OS-managed Page Access List** that tracks page usage and selects victim pages following a FIFO policy [14].

PT-based DRAM caches follow an *insert-on-miss* workflow. On a last-level cache miss, the Physical Address (PA) is checked. Accesses within the local remote DRAM space are forwarded to the DCC to trigger DRAM cache miss handling. If no free page is available, the system selects a victim page using the Page Access List and fetches its destination Page Frame Number (PFN) from the GIPT. Then, an OS routine is launched to perform TLB shootdowns, CPU cache flushes, data transfers and PTE updates. The DCC then starts fetching page from remote memory to the freed slot. The destination PFN is immediately returned to the MMU to update the PTE, allowing subsequent accesses to be directed to the DRAM cache slot while data is fetched asynchronously [12], [13].

III. Motivation and Opportunity

As discussed in Section II-B, existing DRAM cache designs suffer from local DRAM cache underutilization. Tag-based approaches employ static mappings that can lead to conflict-driven imbalance, while PT-based designs suffer from intra-page underutilization under sparse intra-page access patterns.

To quantify these limitations, we analyze local DRAM cache utilization across representative memory-intensive workloads on CXL memory. Our study reveals significant opportunities to improve utilization through dynamic complementary multi-to-one page mapping, which enables pages with complementary access locations to share the same local DRAM cache space.

We collect traces from real-world memory-intensive workloads, including GAP Benchmark Suite [6], NAS Parallel Benchmark [34], and XSBench [35], and perform large-scale

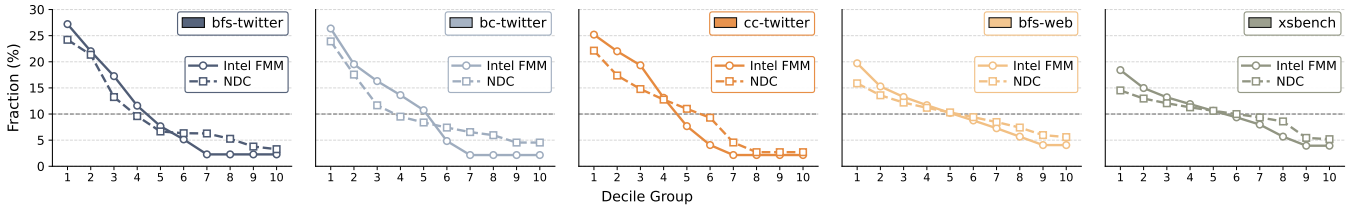

Fig. 4: Fraction of conflicts in each 10% bin of DRAM cache sets sorted by conflict count (Direct-mapped Intel FMM and 16-way set-associative NDC).

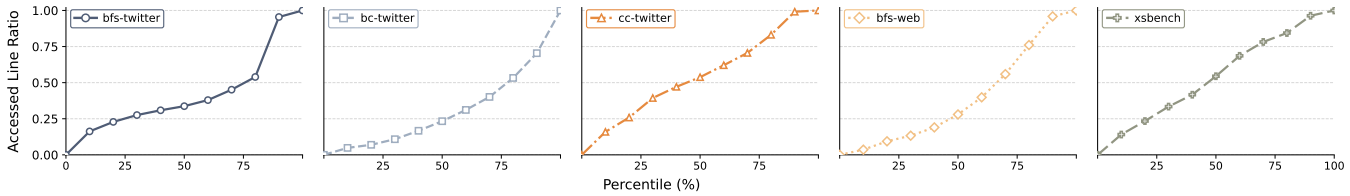

Fig. 5: CDF of intra-page access sparsity (fraction of accessed cache lines).

simulation using ChampSim [19], [20]. The results demonstrate substantial underutilization in existing DRAM cache designs and motivate complementary multi-to-one page mapping. Detailed evaluation settings are described in Section V.

A. Tag-Based DRAM Cache Underutilization from Set-Associative Access Imbalance

Prior work on conventional caches has shown that static direct-mapped and set-associative designs can lead to underutilization due to imbalanced set accesses [9], [16]. Motivated by this observation, we evaluate two SOTA tag-based DRAM cache designs: a 1-way (direct-mapped) design (Intel FMM [16]) and a 16-way set-associative design (NDC [15]). We collect per-set DRAM cache conflict statistics, sort cache sets by their conflict counts, and partition them into deciles (10% bins). We then present the conflict distribution across these bins for five representative workloads.

As shown in Figure 4, access-conflict imbalance is widely observed in both tag-based DRAM cache designs. For example, in the `bfs-twitter` workload, the top 10% of cache sets account for 27.2% and 24.2% of total access conflicts in the 1-way and 16-way set-associative DRAM caches, respectively. Although the 16-way set-associative design exhibits a more balanced distribution than the 1-way scheme, the imbalance remains significant. Moreover, further increasing associativity is difficult due to physical constraints in in-DRAM tag-based implementations. Such imbalanced access conflicts lead to local DRAM cache underutilization and additional CXL remote memory accesses, limiting the performance benefits of DRAM caching.

Take-away #1: Direct-mapped and set-associative DRAM caches suffer from imbalanced access conflicts across cache sets, indicating that static mappings cannot fully exploit the aggregate capacity of the DRAM cache as cache-line accesses are concentrated in a subset of sets and thus suffer higher contention, while other sets remain less utilized.

B. PT-based DRAM Cache Underutilization from Intra-Page Sparsity

As observed in prior work [9], [17], [18], typically only a fraction of a page's cache lines are accessed within a given time period. Therefore, PT-based DRAM caches are expected to suffer from local DRAM cache underutilization due to intra-page access sparsity. To quantify this effect, we collect intra-page access statistics for all accessed pages within each 10-billion-cycle window. We then compute the access ratio for each page and construct a Cumulative Distribution Function (CDF).

The results are shown in Figure 5. On average, 57.3% of accessed pages have fewer than 32 out of 64 (50%) cachelines accessed, indicating that intra-page memory access sparsity is prevalent in real-world workloads. However, existing PT-based DRAM cache designs reserve a full DRAM cache page slot and fetch the entire remote page during each DRAM cache insertion or replacement, even when only a small subset of cache lines is accessed.

Prior approaches, such as footprint-based, page-granularity tag-based DRAM caches [9], [18], [33], mitigate this issue by maintaining per-page access footprints and fetching only the accessed cache lines, reducing remote memory bandwidth consumption. However, these designs still suffer from significant local DRAM cache underutilization, as each DRAM cache slot can accommodate only a single remote memory page, even when that page is sparsely accessed.

Take-away #2: Page-table-based DRAM caches, although supporting dynamic fully associative remote-to-local page mapping, still suffer from local DRAM cache underutilization due to intra-page memory access sparsity.

C. Dynamical Complementary Mapping Opportunities from Intra-Page Sparsity

Because many pages exhibit intra-page sparsity, we explore whether this property can be leveraged to dynamically map

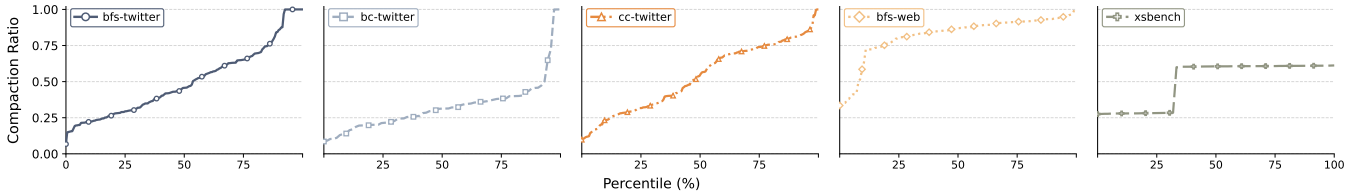

Fig. 6: CDF of compaction ratio after complementary page mapping during each time window.

two or more pages with **Complementary Access Sparsity** into a single local DRAM cache page, thereby improving local DRAM cache efficiency. Complementary access sparsity refers to cases where different pages access distinct and non-overlapping subsets of cache lines within a page. By co-locating such pages, the same local DRAM cache page can store cache blocks from multiple sparse pages, significantly improving local DRAM cache utilization.

To evaluate the potential performance benefits of complementary page mapping, we implement an oracle-based system that collects full memory access traces within each 10-billion-cycle window. For each window, the oracle constructs an ideal page matching plan that minimizes the number of required local DRAM cache pages, such that all pages can be complementarily mapped without any cacheline access overlap. We then compute the compaction ratio for each window, defined as the ratio between the minimum number of local DRAM cache pages required under oracle complementary mapping and the total number of pages accessed in the original program. Finally, we sort the compaction ratios across all windows for each workload and present their CDFs.

As shown in Figure 6, complementary page mapping significantly reduces the required memory capacity by up to 68.2% (53.6% on average across all windows and workloads), demonstrating substantial opportunities to improve local memory utilization through complementary page matching.

Take-away #3: Complementary page mapping reduces the required local DRAM cache capacity (53.6% on average) by mapping pages with complementary intra-page access patterns within a single local DRAM cache page, demonstrating significant potential for improving local DRAM cache utilization.

IV. Design

A. Overview

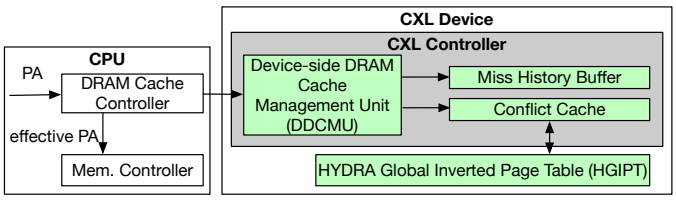

Fig. 7: Architecture overview of HYDRA.

Based on the potential analysis of dynamic complementary page mapping in Section III-C, we propose **Dynamic Complementary Multi-to-one Mapping** to improve DRAM cache utilization and performance in CXL tiered memory systems. However, designing an effective architecture to support this idea presents several challenges. First, there is no straightforward mechanism to efficiently support dynamic complementary multi-to-one mapping, as prior DRAM caching designs typically enforce either multi-to-one static mappings or one-to-one dynamic page mappings, which cannot dynamically map multiple pages into a single DRAM cache page. Second, implementing an efficient page matching mechanism is non-trivial, as the system must identify pages with complementary access distributions at runtime while maintaining low hardware and runtime overhead. Third, supporting effective page mapping revocation is challenging, as existing FIFO- or LRU-based victim selection policies are suboptimal for HYDRA. HYDRA instead needs to revoke mappings for frequently conflicting pages, since previously established complementary mappings may no longer remain effective as access patterns evolve.

We propose HYDRA to effectively tackle these challenges, as shown in Figure 7. HYDRA consists of three key components. To address the first challenge, we introduce **Hybrid DRAM Caching**, which combines dynamic one-to-one page mapping with static multi-to-one cacheline direct mapping using an intermediate physical address space, enabling efficient dynamic multi-to-one page mapping. Cached data block locations are determined using both PTEs and low-overhead 4-bit tags per cacheline. To address the second and third challenges, we introduce a CXL **Device-side DRAM Cache Management Unit (DDCMU)** to offload learning and metadata maintenance from the critical path and avoid reliance on limited CPU-side resources. Specifically, HYDRA maintains a **Miss History Buffer** to track intra-page access sparsity and guide complementary mapping. Following prior PT-based DRAM caches, HYDRA further extends the GIPT into the in-memory **HYDRA Global Inverted Page Table (HGIPT)**, buffered by the **Conflict Cache**, which supports multi-to-one mappings by recording multiple remote pages per local DRAM cache page, along with per-local-DRAM-page conflict counters to guide conflict-aware page mapping revocation.

B. Hybrid DRAM Caching with Intermediate Physical Address Space for Dynamic Multi-to-One Page Mapping

Efficiently supporting **complementary dynamic multi-to-one page mapping** requires new innovation, as prior DRAM cache designs only provide **static multi-to-one mapping** (tag-based) or **dynamic one-to-one mapping** (PT-based), as shown in Figure 8 (a) and (b). Existing DRAM caches adopt a *flat*

physical memory view for both local and remote memory. In this example, the local DRAM size is L (PA range $[0, L-1]$), and the remote CXL memory size is $4L$ (PA range $[L, 5L-1]$). In Figure 8(a), tag-based DRAM caches statically map $[L, 5L-1]$ to $[0, L-1]$, allowing each local cache block to store one of four remote blocks. When a CXL memory block is cached in local DRAM, the mapped PA is computed as $(\text{CXL memory PA} - L) \bmod L$, and the tag, derived from the upper two bits of $(\text{CXL memory PA} - L)$, identifies the resident remote block. In Figure 8(b), PT-based DRAM caches support dynamic one-to-one page mapping from $[L, 5L-1]$ to $[0, L-1]$ at page granularity. When a CXL memory page is mapped to local DRAM, the PFN field of the corresponding PTE is updated to the local DRAM PFN.

However, simply extending PT-based DRAM caches with per-cacheline tags cannot effectively support dynamic multi-to-one page mapping. In a PT-based design, mapping two CXL memory pages to a single local DRAM page requires updating the PFN fields of both PTEs to point to the same local DRAM page slot. Meanwhile, tag-based DRAM caches use a subset of PFN bits (e.g., the upper two bits in Figure 9(a)) as per-cache-line tag values. Therefore, additional mapping metadata must be encoded in the PFN fields of the PTEs to distinguish different mapped pages for multi-to-one page mappings, and these metadata must also serve as per-cache-line tag values.

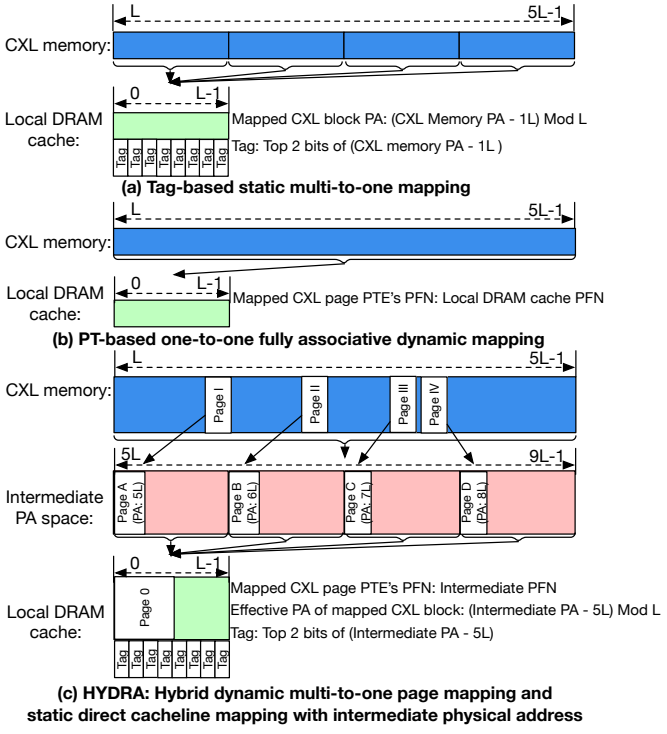

Fig. 8: Mapping illustration of (a) tag-based DRAM caching, (b) PT-based DRAM caching, and (c) HYDRA: Hybrid caching with intermediate physical address (PA) space.

We propose hybrid dynamic one-to-one page mapping with static multi-to-one cacheline direct mapping using an **Intermediate Physical Address Space**, as illustrated in Figure 8(c). The key idea is to introduce an intermediate physical address (PA) space whose size equals the CXL memory space, i.e., $4L$ (PA range $[5L, 9L-1]$), but which does not correspond to any physical DRAM. We then employ hybrid DRAM caching. Specifically, the mapping from CXL memory pages to the intermediate PA space is dynamic one-to-one mapping, while the mapping from the intermediate PA space to the local DRAM cache is static multi-to-one mapping. Tags are then added to each cacheline in local DRAM cache to distinguish which intermediate PA page each cached block originates from. Therefore, unlike PT-based DRAM caches, which transfer an entire page to local DRAM on a cache block miss, HYDRA performs fine-grained data movement by transferring only the requested cache block, similar to tag-based DRAM caches.

To locate the local DRAM cache PA from a CXL memory PA in our hybrid mapping design with an intermediate PA space, HYDRA relies on both PFNs stored in PTEs and tags in per-cacheline ECC bits. We illustrate this process using an example demonstrating dynamic multi-to-one page mapping. Consider dynamically mapping arbitrary CXL memory pages I, II, III, and IV to a single local DRAM cache page 0 with starting PA 0. These pages are first dynamically mapped to intermediate PA space pages A, B, C, and D, whose starting PAs are 5L, 6L, 7L, and 8L, respectively. Pages A, B, C, and D are statically mapped to local DRAM page 0. To record this mapping, the PFN fields of the corresponding PTEs are updated to the PFNs of intermediate pages A, B, C, and D. When a cache block from pages I, II, III, or IV is cached in local DRAM, the effective mapped PA is computed as $(\text{Intermediate PA} - 5L) \bmod L$, which points to page 0 in local DRAM. The tag, derived from the upper two bits of $(\text{Intermediate PA} - 5L)$, identifies whether the cache block originates from page A, B, C, or D.

This design efficiently supports dynamic multi-to-one page mapping by allowing multiple CXL memory pages to share a single local DRAM page, while cache-line tags distinguish blocks from different pages without additional mapping metadata. Unlike PT-based DRAM caches that transfer entire pages on a cache miss, HYDRA transfers only the requested cache block, improving local DRAM utilization by allowing frequently accessed cache lines from complementary pages to reside in local DRAM.

C. Dynamic Complementary Page Mapping

To effectively identify candidate physical pages with complementary intra-page access distributions and dynamically exploit multi-to-one page mapping according to program locality, HYDRA introduces a lightweight, hardware-managed dynamic complementary page mapping mechanism. The idea is to maintain a **Miss History Buffer** that is indexed by PFN and records previously observed intra-page spatial access patterns to predict future accesses, an approach shown to be effective in guiding microarchitectural optimizations (e.g., spatial prefetching [36], [37]). The access bitmaps can be further

backed by an in-memory metadata region to enable more accurate tracking of intra-page access distributions. In addition, we introduce a Device-side DRAM Cache Management Unit (DDCMU) that periodically performs page matching to generate complementary page mapping decisions. Furthermore, HYDRA extends the Global Inverted Page Table (GIPT) into the **HYDRA Global Inverted Page Table (HGIPT)**. Unlike prior one-to-one mapping schemes, HGIPT supports multi-to-one mappings, allowing up to 16 remote pages to map to the same DRAM cache page. Each HGIPT entry is indexed by a local DRAM cache PFN and reserves contiguous storage for 16 remote PFNs. We also add an 8-bit conflict counter within each HGIPT entry to track conflict occurrences, as detailed in Section IV-D.

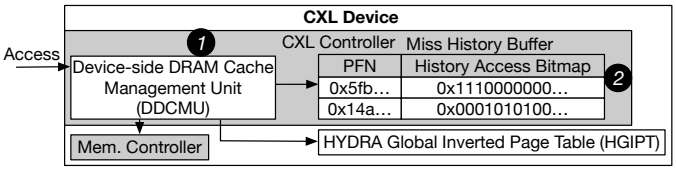

Fig. 9: Workflow of recording intra-page access pattern.

Figure 9 illustrates these components and the workflow for recording accesses to unmapped remote pages; the detailed memory access workflows for different cases are presented in Section IV-F. On a memory access to an unmapped remote page, ❶ the PFN is extracted from the request address and used for a Miss History Buffer lookup, while the memory access is served concurrently by the memory controller. ❷ If hit, the bitmap bit corresponding to the accessed cache line is set to 1. On a miss, an entry is evicted and a new entry is allocated for the accessing page, with all bits initialized to 0 except the accessed cache line, which is set to 1.

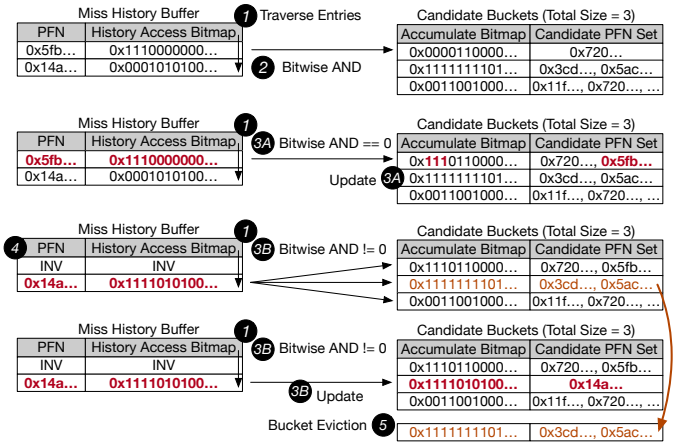

Fig. 10: Complementary page matching workflow.

Periodical Complementary Page Matching. The complementary page matching is triggered when the accumulated remote unmapped page access count since last page matching exceeds a certain threshold. The DDCMU then stops recording remote unmapped page access and starts page matching process. Figure 10 illustrates the workflow of page matching. Initially, the DDCMU allocates multiple (e.g., 8) candidate mapping buckets, each maintaining an accumulated bitmap and the PFNs of the chosen candidate pages. ❶ The DDCMU traverses all valid entries in the table. ❷ For each entry, its access bitmap is extracted to perform a bitwise `AND` operation with each candidate bucket's bitmap. ❸A When a bucket with a zero bitwise `AND` value is found, the current entry's PFN is added to the chosen candidate page set of the bucket, and the accumulated bitmap of the bucket is updated by performing a bitwise `OR` operation with the current entry's access bitmap. ❸B If the current entry cannot be added to any candidate buckets, the bucket with the largest `popcount` value is evicted. Then, another empty bucket is allocated to hold the current entry. ❹ After handling the current entry, it is then flushed from the miss history buffer. ❺ For each evicted bucket, the system allocates an available local DRAM cache slot and performs the page mapping update (illustrated in Section IV-E) to map all remote pages in the bucket to an available local DRAM cache page. After traversing all entries in the miss history buffer, the DDCMU resumes recording remote unmapped page accesses.

D. Conflict-Aware Mapping Revocation

Existing PT-based DRAM caches rely only on a system-managed FIFO-based Page Access List to select victim DRAM cache frames for new page mappings. However, FIFO-based approaches (as well as other replacement strategies such as LRU) fail to consider HYDRA 's complementary mapping design, which aims to reduce conflicts among remote pages mapped to the same local DRAM cache slot. Therefore, existing page mappings that no longer exhibit complementary intra-page access distribution due to changing access patterns might persistently degrade memory performance and never be revoked.

To address this challenge, we introduce a lightweight *conflict-aware mapping revocation*, which proactively revokes local DRAM cache pages experiencing frequent conflicts, as the remote pages currently mapped to these slots are unlikely to benefit from complementary mapping. We maintain an 8-bit conflict counter for each local DRAM cache page in the HGIPT, buffered by the **Conflict Cache**. When the counter overflows, HYDRA issues a revocation request to the DDCMU to revoke the corresponding local DRAM cache page by removing all associated remote page mappings. Otherwise, when all local DRAM cache pages are occupied and the DDCMU issues a new mapping creation request, HYDRA falls back to the default strategy of selecting the victim local DRAM cache slot from the system-managed Page Access List.

Figure 11 illustrates the workflow of conflict-aware mapping revocation. Initially, the conflict counter is set to 0 when allocating a new entry in the HGIPT during page mapping creation. For subsequent memory accesses to mapped pages that result in access conflicts (see Section IV-F for how such accesses are identified), ❶ the PFN is extracted from the request address and used for a HGIPT lookup and the memory access is served concurrently by the memory controller. ❷ The

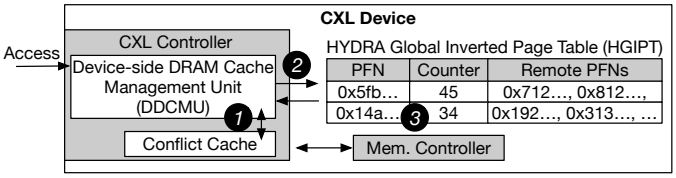

Fig. 11: Workflow of conflict-aware mapping revocation.

counter in the corresponding HGIPT entry is then incremented by one. ❸ If the increment in ❷ causes a counter overflow, a revocation request for the corresponding local DRAM cache page is issued by the DDCMU.

E. Other Details

Similar to prior PT-based DRAM caches [12]–[14], HYDRA launches an OS routine to perform page mapping updates (i.e., revocation of existing mappings, creation of new mappings). During each mapping update process, the OS routine flushes CPU caches and TLB, updates PTE entries and migrates data to ensure consistency.

The workflow of revocation request handling is as follows. The OS routine ❶ receives a mapping revocation request from the DDCMU, including the request type (i.e., revocation), multiple intermediate PFNs that share the same effective PFN (i.e., source PFNs), as well as PFNs of the corresponding remote pages retrieved from the HGIPT (i.e., destination PFNs). The OS routine ❷ updates the state field of all corresponding PTEs as `pending` to block front-end program access and performs a batched TLB shootdown. Then, the OS routine ❸ flushes CPU caches that fall within the source PFNs. All cachelines stored in the victim local DRAM cache slot are then written back to remote memory to ensure data consistency. After that, ❹ the PFN field of all related PTEs is updated to the destination PFNs, and the corresponding entries in the HGIPT are also updated. Finally, the state field of the PTEs is reset to `normal` to resume front-end memory accesses.

Mapping creation request handling shares a similar workflow with revocation request handling, except for two: (1) The request type received from the DDCMU is creation instead of revocation. Therefore, the source PFNs are the PFNs of candidate remote unmapped pages, while the destination PFNs are intermediate PFNs. (2) After flushing CPU caches that fall within the source pages, the OS routine does not fetch data into the local DRAM cache slot from remote unmapped pages. Instead, subsequent front-end program accesses trigger local DRAM cache miss and fetch the data from remote memory on demand.

F. Putting It Together: Memory Access

HYDRA's asynchronous, dynamic page mapping update allows a remote page being either mapped to a local DRAM cache slot or not. Furthermore, the cacheline granularity DRAM cache access handling allows a cacheline within mapped remote pages being placed in remote memory or in local DRAM cache. HYDRA distinguishes these different scenarios by *parsing the requesting physical address sent from the CPU* and applies different memory access handling workflows accordingly, as illustrated in Figure 12.

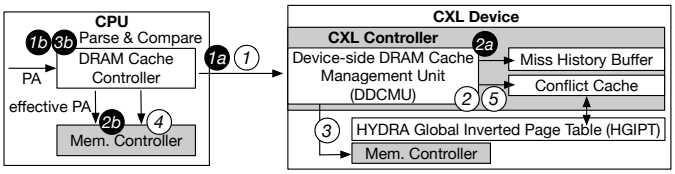

Fig. 12: HYDRA memory access handling workflow.

Unmapped Remote Page Access. Memory access to an unmapped remote page carries a PA within the remote memory's physical address space. In this case, ❶ the memory request is sent directly to the remote CXL memory after a PA range check in the DRAM cache controller. While being processed by the CXL-device-side memory controller, ❷ the request is also forwarded to the DDCMU to update the Miss History Buffer.

Mapped Remote Page Access. When a remote page is mapped to a local DRAM cache slot, memory access to this page carries an address within the intermediate physical address space. After an address range check, the DRAM cache controller ❶ starts to handle this DRAM cache access by parsing the address to get an effective mapped PA and a cacheline tag. Then, ❷ a memory access toward local memory is sent using the effective mapped PA to fetch the corresponding cacheline and tag. ❸ The DRAM cache controller then compares the fetched tag with the parsed tag to determine whether it is a cache hit or miss. ❹ Upon a cache hit, the cacheline is directly returned to the CPU.

Upon an *access conflict* (i.e., cache miss), the CPU-side DRAM cache controller ① sends a `Load` request containing the intermediate PA of the accessing cacheline, and a `Store` request containing the intermediate PA of the accessing cacheline as well as the cacheline data, to the DDCMU. ② The DDCMU then ② triggers two HGIPT lookups (buffered by the Conflict Cache) to fetch the original remote PFNs of the victim cacheline and the accessing cacheline, respectively. Meanwhile, ③ the conflict counter in the corresponding HGIPT entry is incremented. ④ The victim cacheline data is then written back to remote DRAM, while the accessing cacheline is fetched and returned to the CPU-side DRAM cache controller. Finally, the DRAM cache controller ⑤ updates the cacheline tag in local DRAM cache to the parsed tag of the accessing cacheline and returns the data to CPU.

G. Hardware Overhead and Discussion

Table II shows the hardware storage overhead comparison against prior PT- and Tag-based approaches. On the device side, HYDRA introduces a DRAM Cache Management Unit (DDCMU) that maintains two cache structures: a Miss History Buffer for identifying intra-page access sparsity, and a Conflict Cache for buffering accesses to the HYDRA Global Inverted

Page Table (HGIPT) during access conflicts. By default, HYDRA requires only an 8K-entry 8-way set-associative Miss History Buffer (∼128KB) and a 256KB 8-way set-associative Conflict Cache. The area overhead of HYDRA is estimated to be $0.1463\ \mathrm{mm}^2$ using CACTI [38] with 7nm technology [39], about 0.21% and 1.22% of the reported CXL-MHD device die area and unused die area [40], respectively.

The in-memory HGIPT is organized as a flat linear page table stored in a reserved contiguous memory region in remote CXL DRAM, with each entry storing the inverted one-to-multi page mapping (16×40 bits = 640 bits = 80 B) and an 8-bit conflict counter, totaling less than 2% ($81B/4KB \approx 1.98\%$) of the local DRAM cache size within remote memory.

Storage Overhead Comparison with Tag- and PT-Based DRAM Caches. The memory storage overhead of HYDRA includes per-DRAM-cache-line tags (4 bits per cacheline) and per-DRAM-cache-page HGIPT entries (∼80B per page). The tags are stored in local DRAM using unused ECC bits ($0.5B/64B = \sim 0.78\%$). HIGT entries are stored in remote CXL DRAM ($80B/4KB = \sim 2\%$). Notably, with a smaller multi-to-one mapping ratio, e.g., 4:1, which is one quarter of the default configuration, the storage overhead is significantly reduced to 2-bit per-line tags (∼0.4%) and 20-B per-page HIGT entries (∼0.5%).

PT-based DRAM caches incur the smallest storage overhead, as they only require maintaining an inverted table (∼8B [14] per entry for each 4KB DRAM cache slot, $8B/4KB = \sim 0.2\%$) in remote memory. Tag-based caches typically require 2B [15] to 8B [11] per DRAM cache line for tag storage, accounting for 3.125% to 12.5% of the local DRAM cache capacity.

TABLE II: Hardware Overhead Comparison

Type	Per-cacheline (in-memory)	Per-page (in-memory)	Others
Tag-based [8]	***high*** (2B-8B(3.125%-12.5%)) [7], [11]	***none***	***none*** or ***low*** [30]
PT-based [12]	***none***	***low*** (8B (∼0.2%))	***none*** or ***low*** [12]
HYDRA	***low*** (4-bit (∼0.78%))	***medium*** (∼81B (∼2%))	***low*** (∼128KB plus 256KB)

Reliability Implication of Using ECC bits. HYDRA used in-ECC tag storage as a low-overhead implementation option similar to prior in-DRAM-tag commercial processor designs (e.g., Intel Flat Mode [8], Intel Memory Mode [7], Intel ccNUMA [41], Arm MTE [42]), since system ECC often leaves tens of spare bits [16] that can be used. Compared with tag-based DRAM caches, HYDRA requires shorter per-cacheline tags, freeing space for stronger protection guarantees and helping HYDRA remain practical on DRAM modules with limited system ECC bits (e.g., HBM3 [43], [44]). HYDRA does not fundamentally depend on repurposing ECC bits. For reliability-critical systems, the 4-bit per-cacheline tags can be stored as normal data and retrieved through additional memory accesses [11], [25], preserving standard data ECC protection. This is a common design tradeoff in prior tag-based DRAM caches [11], [25], and it does not change HYDRA 's mapping mechanism or correctness.

V. EVALUATION

A. Evaluation Methodology

TABLE III: Scaled-down System Configuration.

Cores	4 OoO cores, 4GHz, 6-wide, 224-entry ROB, 72-entry LQ, 56-entry SQ
Private L1-(I/D)	32KB, 8-way, 4-cycle latency
Private L2C	1MB, 8-way, 12-cycle latency
Shared LLC	1.375MB per core, 11-way, 24-cycle latency
Local DRAM	2GB DRAM Cache, 4x DDR5-4800 channels, open row policy, tRC-tRCD-tCL-tRP: 48-15-20-15
DRAM Cache Controller	64-entry read/write buffer, FR-FCFS scheduling policy
CXL link	one-way latency: 50ns, bandwidth: 8GB/s (per direction)
CXL Memory	32GB 2x DDR5-4800 channels, tRC-tRCD-tCL-tRP: 48-15-20-15
DDCMU	2GHz; Miss History Buffer: 1024-set, 8-way, 12-cycle latency; Conflict Cache: 512-set, 8-way, 64B-per-line, 12-cycle latency

1) Benchmarks: Our evaluation targets memory-intensive datacenter applications with large memory footprint (ranging from 5.5GB to 32GB). For HPC workloads, we choose four workloads with large-scale input from NAS Parallel Benchmark Suite [34] and XSBench [35]. For graph analysis, we use six workloads from GAPBS [6] with real-world graphs (`twitter` and `web`). We further add database and LLM workloads: `TPC-C` and `YCSB` on SiloDB [45], and `llama.cpp` decoding on `Llama-3.1-8B-Instruct` with batch size 4, a 4K-token context window, and GGUF Q8_0 quantization. We distinguish different input graphs as different workloads even for the same application as their memory access characteristic are significantly different, as shown in Section III.

2) Baselines: We compare HYDRA against state-of-the-art Tag-based and PT-based DRAM cache schemes, including: (1) **Intel Flat Memory Mode (FMM)** [7], a Tag-based industry solution that using ECC-based tags to manage direct-mapped DRAM cache; (2) **Genie Cache** [12], the state-of-the-art PT-based DRAM cache, asynchronously performing DRAM cache replacement via hardware delegation; (3) **Unison Cache** [9], a Tag-based DRAM cache solution using page-granularity cache management while loading only a subset of cache lines within a page into local DRAM based on per-page access footprint information; (4) **Native DRAM Cache (NDC)** [15], the state-of-the-art 16-way set-associative Tag-based DRAM cache, applying in-memory tag check and comparison to reduce latency and bandwidth overheads from fetching tags; and (5) **Accord** [31], a 4-way set-associative Tag-based DRAM cache scheme with way predictor that can reduce extra tag accesses.

3) Simulation Methodology: We use ChampSim [19], [20], a trace-based cycle-level simulator for core simulation. We integrate Ramulator 2.0 [21], a widely-used memory simulator, for accurate memory simulation. We use an 8x scaled-down

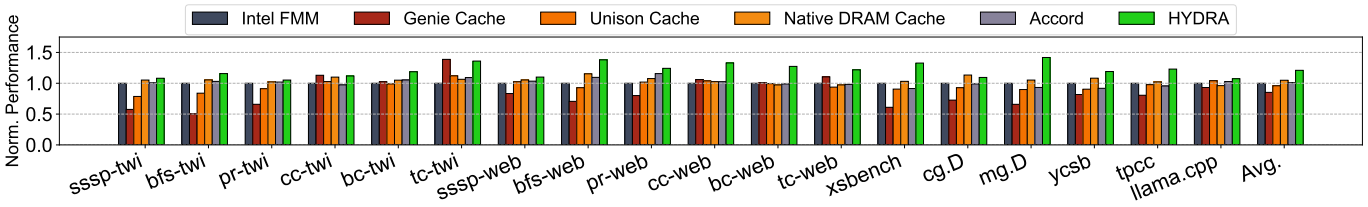

Fig. 13: End-to-end performance normalized to Intel FMM.

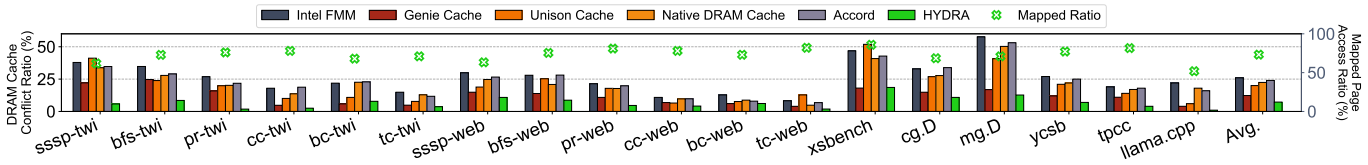

Fig. 14: DRAM cache conflict ratios across designs and HYDRA mapped-page access ratios.

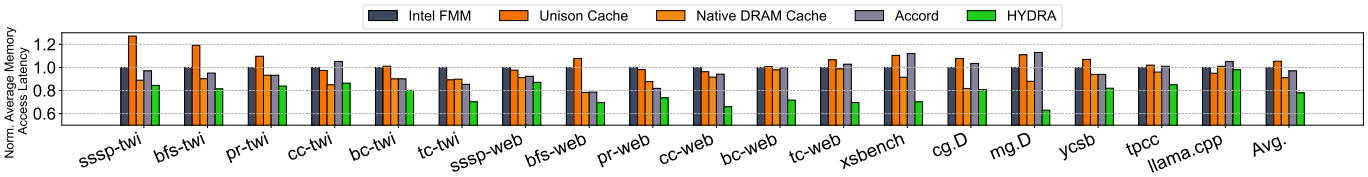

Fig. 15: Average memory access latency normalized to Intel FMM.

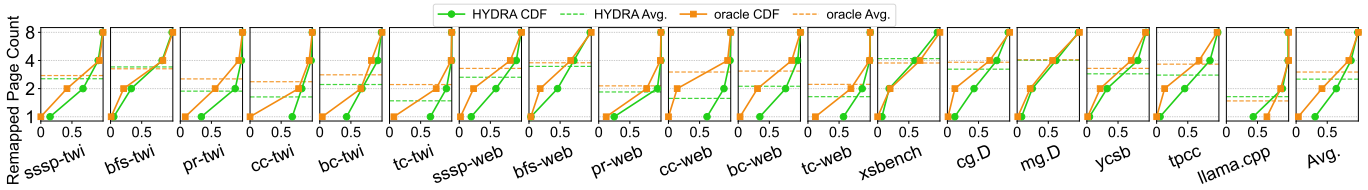

Fig. 16: Remapped page count distribution across local DRAM cache slots vs. oracle.

configuration of a target system similar to Intel Xeon CPU Max 9462 [7], as listed in Table III. Since prior DRAM-cache designs [9], [15], [18], [25] were designed and evaluated for tiered memory systems combining die-stacked memory with DDR DRAM, whereas CXL-based tiered-memory systems typically assume larger local DRAM capacities and larger workload memory footprints, we use a larger DRAM cache capacity than prior work: from around 128MB/core [9], [15] to 512MB/core by default and up to 1GB/core. We also use remote-to-local memory capacity ratios similar to those in prior work, typically ranging from 4:1 to 16:1 across workloads [9], [25], [31]. Following prior work [25], [46], We run each workload with multi-threaded setting on real machines and use Intel PinTool [47] to collect instruction and memory traces every 1-billion instruction intervals to generate 20 checkpoints (i.e., replay windows), totaling 20 billion instructions per thread. In order to (1) fully warm up the DRAM cache and (2) warm up other stateful components, such as the CPU front end, CPU caches, and DDCMU tracking components, for the k^{th} replay window, we first replay the memory traces to fast-forward to the end of the $(k-2)^{th}$ replay window and warm up the DRAM cache. We then replay the instruction traces of the $(k-1)^{th}$ replay window to warm up both the DRAM cache and the other components. Finally, we replay the instruction traces of the k^{th} replay window and collect performance metrics. For HYDRA and PT-based DRAM cache baselines, we model each OS-managed remapping with a fixed 15μs latency penalty to account for TLB shootdowns and other OS-kernel-induced overheads, following prior work [12], [48].

B. End-to-end performance

1) Overall Performance: Figure 13 presents the overall IPC performance of all evaluated schemes normalized to Intel FMM, the state-of-the-art direct-mapped tag-based DRAM cache. HYDRA achieves average speedups of 21.6%, 14.3% and 42.6% (up to 42.1%, 35.2% and 118.1%) compared to the state-of-the-art direct-mapped tag-based scheme (Intel FMM), set-associative tag-based scheme (Native DRAM Cache) and

PT-based scheme (Genie Cache), respectively, demonstrating the effectiveness of hybrid DRAM caching.

With lower intra-page sparsity, HYDRA has fewer opportunities to pack complementary sparse pages, reducing its benefit. The LLM inference workload confirms this behavior: its denser access patterns leave less sparsity for HYDRA to exploit, leading to a smaller improvement than on sparse workloads, while still outperforming prior DRAM-cache designs (4.8% over Intel FMM). For rapidly shifting complementary sparsity, stale page groupings may reduce HYDRA 's packing efficiency and cause more conflicts. However, HYDRA bounds this effect through conflict-aware mapping revocation and periodic remapping.

To further evaluate the effectiveness of HYDRA's dynamic complementary multi-to-one page mapping, we collect local DRAM access conflict ratios of different baselines across all workloads. For HYDRA, we collect access counts of mapped and unmapped pages, and compute the local DRAM cache conflict ratio by dividing the total access conflict count by the total access count toward mapped remote pages. As shown in Figure 14, due to the limited local DRAM cache capacity, and because newly accessed pages need to wait for background access tracking and periodic page-mapping creation, an average of 26.8% of memory accesses are served by unmapped remote pages. Notably, these accesses do not first probe the local DRAM cache for tag checking, and thus incur substantially lower access latency as well as lower memory bandwidth and energy overhead than DRAM cache access conflicts, which further trigger additional operations such as evictions.

Meanwhile, accesses to mapped remote pages exhibit the lowest conflict ratio (6.6% on average), showing that **HYDRA's dynamic complementary multi-to-one page mapping and conflict-aware mapping revocation significantly reduce access conflicts.** In contrast, the three line-granularity tag-based DRAM cache schemes (Intel FMM, Accord and Native DRAM Cache) exhibit higher cache conflict ratios. With higher cache set associativity (16-way), Native DRAM Cache achieves an average conflict ratio of 20.2%, while the direct-mapped Intel FMM achieves an average conflict ratio of 25.8%. The PT-based DRAM cache scheme, Genie Cache, achieves the lowest conflict ratio of 10.7% among all baselines, but shows the lowest average IPC, as the high DRAM cache miss penalty significantly degrades system performance.

To present the overall performance of the DRAM cache system, we further collect the average memory access latency for each baseline by tracking the lifetime of all memory accesses in the memory subsystem to compute the average processing time. We omit the PT-based scheme, Genie Cache, as memory accesses in Genie Cache may trigger a costly OS-routine-managed DRAM cache replacement process. Figure 15 presents the results normalized to the Intel FMM baseline. HYDRA significantly reduces the average memory access latency by 24.1% compared to Intel FMM. In contrast, all other tag-based schemes, Unison Cache, Accord and NDC, struggle to improve memory performance (105.3%, 96.3% and 89.6% of latency compared to Intel FMM). In general, HYDRA

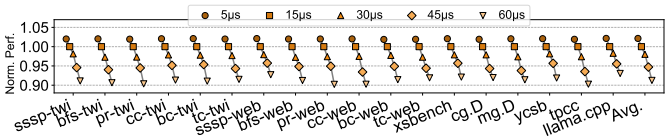

Fig. 17: Sensitivity to TLB shootdown penalties.

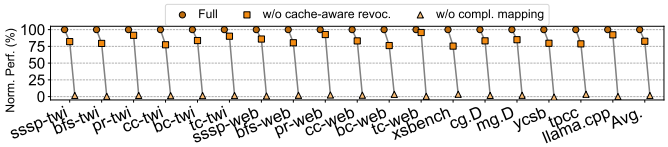

Fig. 18: Ablation study.

achieves the best memory access performance by effectively exploiting complementary intra-page access sparsity and supporting conflict-aware mapping revocation to reduce access conflicts, while maintaining a low cache miss penalty.

2) Page mapping characteristics: We collect the runtime-average distribution of remote pages that are remapped into DRAM cache slots for HYDRA and the oracle scenario (Section III-C) to further analyze the effectiveness and hardware overhead of dynamic complementary page mapping. As shown in Figure 16, the mapped page count for each DRAM cache slot differs across workloads, with an average of 2.53 pages and 3.01 pages (up to 4.52 pages and 4.07 pages, with HYDRA reaching 84.1% of oracle on average) per slot for HYDRA and oracle, respectively, suggesting that the dynamic page mapping and revocation mechanisms of HYDRA make workload-dependent mapping decisions to optimize performance. Most of the DRAM cache slots (>99%) hold less than 8 different remapped remote pages, indicating that maintaining 4-bit per-cacheline tags for up to 16 mapped remote pages is sufficient.

3) Ablation study: To isolate the benefits of dynamic complementary page mapping (Section IV-C) and conflict-aware mapping revocation (Section IV-D), we conduct an ablation study of HYDRA, as shown in Figure 18. The complementary page mapping accounts for 80.4% of the performance improvement over the random-mapping baseline, while the conflict-aware revocation accounts for 17.9% of the performance improvement over the LRU-based revocation.

4) Power consumption: To evaluate the power consumption of HYDRA, we use Ramulator's DRAM power model [21], [49] to access the power consumption metrics of local and remote memory modules, following prior work [15]. For CXL link power consumption, we use 8 pJ per bit following prior work [39]. As shown in Figure 19, HYDRA reduces total energy consumption by 24.9% compared to Intel FMM. We further break down the energy consumed by CXL link traffic and observe a 2.9% reduction over Intel FMM. Within HYDRA, CXL-link energy caused by DRAM cache access conflicts in mapped pages accounts for only 5.3% of the total energy, representing a 70.7% reduction compared to the corresponding component in Intel FMM. The remaining 12.4%

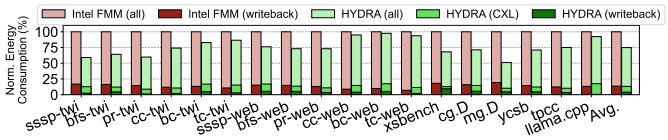

Fig. 19: Power consumption normalized to Intel FMM.

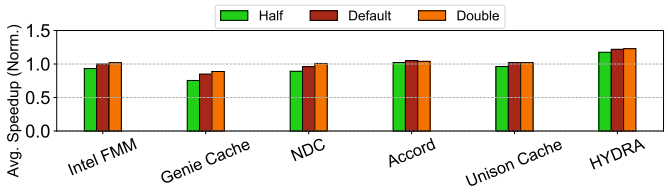

Fig. 20: Sensitivity to DRAM cache capacity.

of total energy is attributed to CXL link traffic generated by unmapped page accesses.

For the CXL-device-side DDCMU and its related operations (i.e., conflict cache accesses and miss history buffer updates), we use CACTI [38] to model the area overhead and power consumption, and scale the results to 7nm technology following prior work [39]. The area overhead of the DDCMU mainly comes from its cache structures, totaling $0.1463\ mm^2$, which is about 0.21% of the reported CXL device die area [40]. We then estimate the energy consumption by multiplying the operation counts by the average energy overhead of each operation: 0.05136 nJ per conflict cache lookup and 0.01798 nJ per miss history buffer update. The results show that the overall runtime energy consumption accounts for only 0.11% of the total memory energy consumption on average; therefore, we omit these values from Figure 19.

C. Sensitivity study

1) Sensitivity to DRAM cache capacity: Figure 20 illustrates the relative performance of all evaluated schemes over the Intel FMM baseline. With halved DRAM cache capacity, all DRAM cache schemes exhibit less performance gain. PT-based Genie Cache show the most severe performance degradation (12.7%) due to increased penalties from cache replacement with higher miss ratios, while others (including HYDRA) exhibit less performance degradation.

2) Sensitivity to CXL link latency: Figure 21 shows the performance gain of HYDRA under different CXL link latency assumptions. With the one-way link latency rising from 50ns to 100ns, HYDRA achieves an extra 16.2% performance improvement on average as it can gain more benefits from reducing high-latency CXL memory accesses.

3) Sensitivity to mapping update frequencies: The frequency of HYDRA's complementary multi-to-one page mapping is automatically adjusted based on the accumulated remote memory access count since last epoch. Figure 23 demonstrates the overall performance under different triggering thresholds. With smaller thresholds, the overall performance degrades significantly, as the aggressive mapping update triggers frequent data movement and TLB shootdowns, blocking front-end memory request handling. The overall performance maintains mostly stable with 100-200K threshold settings, and shows a slight degradation with higher threshold settings (8.3% at 800K) due to missed opportunities of converting remote unmapped page accesses into DRAM cache accesses.

4) Sensitivity to TLB shootdown penalties: As TLB shootdowns are a dominant factor of page mapping update overheads, we also measure the system performance with various TLB shootdown penalties. As shown in Figure 17, the performance of HYDRA degrades by 2.1%, 5.4% and 8.8% with 30μs, 45μs, and 60μs of shootdown penalties compared to the default setting (15μs penalty).

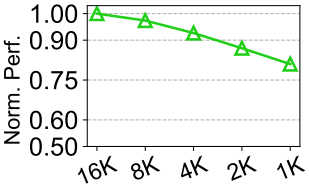

Fig. 22: Sensitivity to Miss History Buffer size.

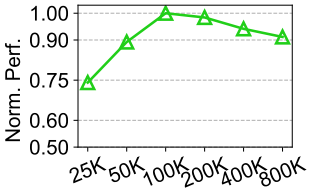

Fig. 23: Sensitivity to complementary page mapping intervals.

5) Sensitivity to hardware overheads: The major hardware overhead of HYDRA comes from the device-side Miss History Buffer and Conflict Cache. Both of them are critical to system performance: With higher Miss History Buffer capacity, HYDRA can preserve more history information to guide complementary page mapping more effectively. With higher Conflict Cache capacity, the extra latency penalty of accessing HGIPT during DRAM cache access conflict handling can be more effectively reduced.

Figure 22 shows the performance degradation when reducing capacity of Miss History Buffer from 16K to 1K entries. We observe that with less than 8K entries, the performance impact from inaccurate access history tracking becomes sig-

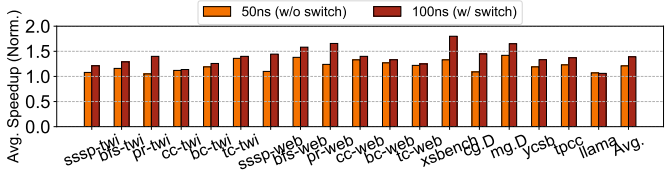

Fig. 21: Sensitivity to CXL link latency.

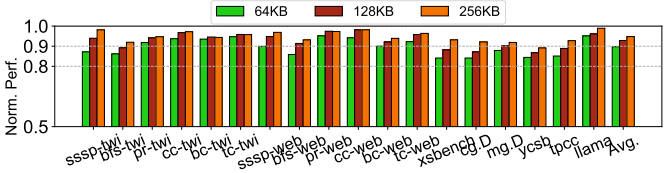

Fig. 24: Sensitivity to Conflict Cache size.

nificant. Therefore, we choose 8K entry as the default setting. Figure 24 shows the system performance under different Conflict Cache size, normalized to an ideal setting with no HGIPT access latency. HGIPT access during DRAM cache access conflict consumes 4.9% (256KB Conflict Cache) to 9.6% (64KB) of total execution time on average. With a total Conflict Cache size over 128KB, the total performance gain from adding more cache becomes marginal (1.6% from 128KB to 256KB).

VI. Related Work

In addition to the related work discussed in Section II, this section covers other related work on hardware DRAM caching, as well as OS-managed software memory tiering techniques.

Other DRAM cache optimizations. For line-granularity direct-mapped DRAM caches such as Intel FMM, prior work [50] mitigates inter-page conflicts by reducing the number of allocated pages mapped to the same cache location. However, such allocation is suboptimal because it cannot observe intra-page access distributions or exploit complementary page mappings at allocation time. We integrate this policy into our Intel FMM baseline and observe only marginal improvement. Memstrata [8] mitigates inter-VM conflicts in tiered-memory systems via software conflict detection and page migration, but it cannot identify complementary access patterns for proactive page mapping and relies on costly post-conflict adjustment.

For line-granularity set-associative DRAM caches, prior work pads tags in consecutive DRAM columns to reduce tag-access bandwidth [10], [51], or uses extra SRAM caches [30] or lightweight way predictors [9], [31] to lower tag overhead. These designs introduce additional hardware complexity and are less effective than Native DRAM Cache [15]. Other designs decouple tags from data and store tags in standalone metadata regions [25], [27], [28], enabling flexible associativity and block sizes beyond DRAM row constraints. However, they require additional hardware structures (e.g., tag MATs [15], [25]), and careful scheduling to overlap tag and data accesses [25].

Prior cache designs [52], [53] explore improving the space utilization of coarse-grained cache blocks. Decoupled Sectored Cache [53] improves sectored-cache utilization by allowing cache lines from different cache blocks to share the same physical sector. DFC [52] mitigates memory under-utilization in Footprint Cache [18] by packing the footprints of multiple blocks contiguously within the same DRAM cache set. However, within each set or sector, these designs need to maintain multiple tags as well as metadata for locating individual cache lines. This incurs even more severe storage overhead and bandwidth bloat than set-associative DRAM caches [9], [15]. Moreover, they still suffer from the conflict imbalance inherent in set-associative caches. Hydra addresses these limitations with a hybrid PT- and Tag-based indexing scheme that distributes tags and metadata across PTEs and per-line ECC bits, while enabling highly dynamic remote-to-local indexing.

Kernel-based software memory tiering. The Linux kernel supports migrating frequently access data to the local memory tier through page migration [17], [22], [23], [54]–[58]. Page migration allows migrating data based on access hotness statistics but requires costly access tracking [22] and cannot detect intra-page access spatial sparsity. By adding an indicating bit in the PTE, systems can seamlessly switch between HW-based (Hydra) and SW-based memory tiering for each page. Designing HW/SW hybrid approaches beyond Hydra is a promising future work for further performance improvement.

VII. Conclusion

We propose Hybrid DRAM Caching with Dynamic Complementary Page Mapping (Hydra), a DRAM cache for CXL tiered memory systems that monitors intra-page access patterns and performs multiple-to-one remote-to-memory page mapping based on the complementary intra-page access sparsity of remote pages to increase local memory utilization and minimize conflict accesses. We develop architectural support to enable cacheline-level fetch and replacement, as well as page-level dynamic mapping. Evaluations on large-scale datacenter workloads show that Hydra outperforms existing DRAM cache designs by 14.3%–42.6% on average.

REFERENCES

[1] CXL, "Cxl 3.1 specification," https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-3.1-Specification.pdf, 2024, online; accessed Jun, 2024.

[2] ——, "Compute express link," https://computeexpresslink.org/, 2024, online; accessed Jun, 2024.

[3] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: Cxl-based memory pooling systems for cloud platforms," in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 574–587. [Online]. Available: https://doi.org/10.1145/3575693.3578835

[4] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," ACM Trans. Intell. Syst. Technol., vol. 15, no. 3, Mar. 2024. [Online]. Available: https://doi.org/10.1145/3641289

[5] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, "Gpu cluster for high performance computing," in SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, 2004, pp. 47–47.

[6] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," arXiv preprint arXiv:1508.03619, 2015.

[7] M. D. Powell, P. Fleming, V. I. Krishna, N. Lakkakula, S. Ravisundar, P. Mosur, A. Biswas, P. Dubey, K. Sood, A. Cunningham, and S. Kumar, "Intel xeon 6 product family," IEEE Micro, vol. 45, no. 3, pp. 31–40, 2025.

[8] Y. Zhong, D. S. Berger, C. Waldspurger, I. Agarwal, R. Agarwal, F. Hady, K. Kumar, M. D. Hill, M. Chowdhury, and A. Cidon, "Managing memory tiers with cxl in virtualized environments," in Symposium on Operating Systems Design and Implementation, 2024.

[9] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014, pp. 25–37.

[10] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, 2011, p. 454–464. [Online]. Available: https://doi.org/10.1145/2155620.2155673

[11] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, 2012, pp. 235–246.

[12] Y. Kim and W. J. Song, "Genie cache: Non-blocking miss handling and replacement in page-table-based dram cache," in 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2024, pp. 983–996.

[13] Y. Kim, H. Kim, and W. J. Song, "Nomad: Enabling non-blocking os-managed dram cache via tag-data decoupling," in 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2023, pp. 193–205.

[14] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A fully associative, tagless dram cache," in Proceedings of the 42nd Annual International Symposium on Computer Architecture, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 211–222. [Online]. Available: https://doi.org/10.1145/2749469.2750383

[15] Y. Ryu, Y. Kim, G. Jung, J. H. Ahn, and J. Kim, "Native dram cache: Re-architecting dram as a large-scale cache for data centers," in 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), 2024, pp. 1144–1156.

[16] M. Hildebrand, J. T. Angeles, J. Lowe-Power, and V. Akella, "A case against hardware managed dram caches for nvram based systems," in 2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2021, pp. 194–204.

[17] Y. Sun, J. Kim, Z. Yu, J. Zhang, S. Chai, M. J. Kim, H. Nam, J. Park, E. Na, Y. Yuan, R. Wang, J. H. Ahn, T. Xu, and N. S. Kim, "M5: Mastering page migration and memory management for cxl-based tiered memory systems," in Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 604–621. [Online]. Available: https://doi.org/10.1145/3676641.3711999

[18] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache," in Proceedings of the 40th Annual International Symposium on Computer Architecture, ser. ISCA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 404–415. [Online]. Available: https://doi.org/10.1145/2485922.2485957

[19] "Champsim," https://github.com/ChampSim/ChampSim, 2025.

[20] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," arXiv preprint arXiv:2210.14324, 2022.

[21] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," IEEE Comput. Archit. Lett., vol. 15, no. 1, p. 45–49, Jan. 2016. [Online]. Available: https://doi.org/10.1109/LCA.2015.2414456

[22] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "Tpp: Transparent page placement for cxl-enabled tiered-memory," in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 742–755. [Online]. Available: https://doi.org/10.1145/3582016.3582063

[23] L. Xiang, Z. Lin, W. Deng, H. Lu, J. Rao, Y. Yuan, and R. Wang, "Nomad:{Non-Exclusive} memory tiering via transactional page migration," in 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), 2024, pp. 19–35.

[24] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong et al., "Demystifying cxl memory with genuine cxl-ready systems and devices," in Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, 2023, pp. 105–121.

[25] M. Babaie, A. Akram, W. Elsasser, B. Haukness, M. R. Miller, T. Song, T. Vogelsang, S. C. Woo, and J. Lowe-Power, "Efficient caching with a tag-enhanced dram," in 2025 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2025, pp. 745–760.

[26] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent hardware management of stacked dram as part of memory," in 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014, pp. 13–24.

[27] P. Behnam, A. Pal Chowdhury, and M. Nazm Bojnordi, "R-cache: A highly set-associative in-package cache using memristive arrays," in 2018 IEEE 36th International Conference on Computer Design (ICCD), 2018, pp. 423–430.

[28] F. Hameed, A. A. Khan, and J. Castrillon, "Improving the performance of block-based dram caches via tag-data decoupling," IEEE Transactions on Computers, vol. 70, no. 11, pp. 1914–1927, 2021.

[29] C. C. Chou, A. Jaleel, and M. K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014, pp. 1–12.

[30] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in 2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT), 2014, pp. 51–60.

[31] V. Young, C. Chou, A. Jaleel, and M. Qureshi, "Accord: Enabling associativity for gigascale dram caches by coordinating way-install and way-prediction," in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 2018, pp. 328–339.

[32] G. Huang, H. Litz, and Y. Xu, "Pipm: Partial and incremental page migration for multi-host cxl disaggregated shared memory," in Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ser. ASPLOS '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 1445–1460. [Online]. Available: https://doi.org/10.1145/3779212.3790203

[33] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J. W. Lee, "Efficient footprint caching for tagless dram caches," in 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016, pp. 237–248.

[34] D. Griebler, J. Loff, G. Mencagli, M. Danelutto, and L. G. Fernandes, "Efficient nas benchmark kernels with c++ parallel programming," in 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), 2018, pp. 733–740.

[35] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis," The Role of Reactor Physics toward a Sustainable Future (PHYSOR), 2014.

[36] S. Somogyi, T. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in 33rd International Symposium on Computer Architecture (ISCA'06), 2006, pp. 252–263.

[37] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2019, pp. 399–411.

[38] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1," Technical Report HPL-2008-20, HP Labs, Tech. Rep., 2008.

[39] H. Ham, J. Hong, G. Park, Y. Shin, O. Woo, W. Yang, J. Bae, E. Park, H. Sung, E. Lim, and G. Kim, "Low-overhead general-purpose near-data processing in cxl memory expanders," in 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2024, pp. 594–611.

[40] Y. Zhong, F. Kazhamiaka, P. Zardoshti, S. Teng, R. Fonseca, M. D. Hill, and D. S. Berger, "Octopus: Enhancing CXL memory pods via sparse topology," in 23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26). Renton, WA: USENIX Association, May 2026, pp. 1303–1322. [Online]. Available: https://www.usenix.org/conference/nsdi26/presentation/zhong

[41] K. Loughlin, S. Saroiu, A. Wolman, Y. A. Manerkar, and B. Kasikci, "Moesi-prime: preventing coherence-induced hammering in commodity workloads," in Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022, pp. 670–684.

[42] L. Lamster, M. Unterguggenberger, D. Schrammel, and S. Mangard, "HashTag: Hash-based integrity protection for tagged architectures," in 32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association, Aug. 2023, pp. 2797–2814. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/lamster

[43] J. Kim, S. Kim, Y. Ryu, S. Gorgin, and J. Kim, "Cerberus: Cross-layer ecc co-design for robust and efficient memory protection," arXiv preprint arXiv:2605.02220, 2026.

[44] B. Murdock, "What designers need to know about hbm3," https://www.synopsys.com/articles/hbm3-ip-dwtb.html, 2022, accessed 2025-09.

[45] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, "Speedy transactions in multicore in-memory databases," in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ser. SOSP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 18–32. [Online]. Available: https://doi.org/10.1145/2517349.2522713

[46] A. Cho and A. Daglis, "Starnuma: Mitigating numa challenges with memory pooling," in Proceedings of the 2024 57th IEEE/ACM International Symposium on Microarchitecture, ser. MICRO '24. IEEE Press, 2024, p. 997–1012. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00077

[47] M. Bach, M. Charney, R. Cohn, E. Demikhovsky, T. Devor, K. Hazelwood, A. Jaleel, C.-K. Luk, G. Lyons, H. Patil, and A. Tal, "Analyzing parallel programs with pin," Computer, vol. 43, no. 3, pp. 34–41, 2010.

[48] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories," in 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), 2015, pp. 126–136.

[49] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," URL: http://www. drampower. info, vol. 22, 2012.

[50] B. Lepers and W. Zwaenepoel, "Johnny cache: the end of DRAM cache conflicts (in tiered main memory systems)," in 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). Boston, MA: USENIX Association, Jul. 2023, pp. 519–534. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/lepers

[51] 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, 2012.

[52] S. Shin, S. Kim, and Y. Solihin, "Dense footprint cache: Capacity-efficient die-stacked dram last level cache," in Proceedings of the Second International Symposium on Memory Systems, ser. MEMSYS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 191–203. [Online]. Available: https://doi.org/10.1145/2989081.2989096

[53] A. Seznec, "Decoupled sectored caches: conciliating low tag implementation cost," in Proceedings of the 21st Annual International Symposium on Computer Architecture, ser. ISCA '94. Washington, DC, USA: IEEE Computer Society Press, 2026, p. 384–393. [Online]. Available: https://doi.org/10.1109/ISCA.1994.288133

[54] Y. Huang and H. A. Maruf, https://lwn.net/Articles/876993/, 2021, [PATCH 0/5] Transparent Page Placement for Tiered-Memory.

[55] T. Lee, S. K. Monga, C. Min, and Y. I. Eom, "Memtis: Efficient memory tiering with dynamic page classification and page size determination," in Proceedings of the 29th Symposium on Operating Systems Principles, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 17–34. [Online]. Available: https://doi.org/10.1145/3600006.3613167

[56] J. Liu, H. Hadian, H. Xu, and H. Li, "Tiered memory management beyond hotness," in Proceedings of the 19th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI '25. USA: USENIX Association, 2025.

[57] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter, "Hemem: Scalable tiered memory management for big data applications and real nvm," in Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 392–407. [Online]. Available: https://doi.org/10.1145/3477132.3483550

[58] Z. Zhou, Y. Chen, T. Zhang, Y. Wang, R. Shu, S. Xu, P. Cheng, L. Qu, Y. Xiong, J. Zhang, and G. Sun, "Neomem: Hardware/software co-design for cxl-native memory tiering," in 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2024, pp. 1518–1531.