Focus on What Matters: DRAM-CXL Hybrid Memory Management with PRISM

Anonymous Author(s)

Abstract

Page migration in DDR-CXL tiered memory—promoting hot pages to fast DDR and demoting cold ones to slow CXL—is the primary lever for managing performance. Nevertheless, existing policies fail on two fronts. (*i*) On the signal side, they use "access frequency" as the hotness proxy. However, out-of-order processors hide many accesses via memory-level parallelism, and hardware prefetchers issue accesses ahead of demand—neither often generates a pipeline stall, making access frequency a poor proxy for stall urgency. (*ii*) On the system side, they treat CXL as just another slow memory tier, overlooking characteristics unique to its PCIe-based interconnect: a highly non-linear latency-bandwidth curve that contaminates stall signals under congestion, and migration traffic competing with demand on the same shared link.

We present *PRISM*, which uses ROB-head stall events—loads that block the entire pipeline and *cannot* be hidden—as the hotness signal, and applies four CXL-aware mechanisms: suppressing pages whose stall scores are inflated by transient link congestion; boosting pages that simultaneously stall multiple cores; rewarding stable hot pages to reduce migration churn; and capping migration bandwidth to prevent link saturation from over-migration.

Evaluated on 13 workloads spanning graph analytics, ML inference, and GBDT training, *PRISM* achieves 16% over the state-of-the-art, requiring only modest hardware additions.

1 Introduction

CXL-attached memory is becoming standard for expanding capacity in servers handling large numbers of concurrent, memory-intensive workloads. By connecting large DRAM modules over a PCIe-based CXL link, cloud operators can provision memory pools that far exceed what DDR slots alone can provide [19, 34]. Major memory vendors now ship CXL DRAM [3, 4], and hyperscalers are deploying it at scale [38]. The canonical deployment pairs a modest fast DDR tier with a large slow CXL tier. When data exceeds DDR capacity, pages spill to CXL, incurring extra latency on every access. Page migration—promoting hot pages to DDR and demoting cold ones to CXL—is the primary mechanism to mitigate this penalty.

Access frequency is not the right signal. Existing DDR-CXL proposals [32, 38, 58] use "access frequency" as their hotness signal. Modern out-of-order (OOO) processors, however, expose significant memory-level parallelism (MLP): by issuing many memory requests simultaneously they hide individual access latencies, and hardware prefetchers inflate access counts further by issuing requests ahead of actual demand. Neither stalls the pipeline—they are not performance-critical and should not drive migration decisions.

What the processor cannot hide is a load at the head of the Reorder Buffer (ROB): when the oldest uncommitted instruction is a pending load, no instruction can retire and the pipeline stalls entirely. A page that repeatedly triggers such "ROB head-load stall events" directly costs performance regardless of its access count. We show that *stall-critical pages are not the same as the most-accessed pages*—access frequency is a poor proxy for what actually matters, i.e., processor performance. Therefore, we argue that the primary migration signal should be the ROB head-load stall event.

CXL is not just slow DRAM. Existing DDR-CXL proposals [32, 38, 53, 58] treat CXL as simply a slower memory tier, inheriting a design philosophy from DDR+HBM [39, 45] and DRAM+NVM [8, 35] where the slow tier occupies a dedicated bus with multiple independent channels and relatively predictable latency.

CXL, however, has fundamentally different hardware characteristics. First, while DDR spreads requests across multiple independent channels, CXL routes all traffic over a *single shared PCIe link*, creating serialization and head-of-line blocking. Second, CXL latency is *highly non-linear*: because all traffic serializes on one link, queuing delay grows super-linearly as load approaches link capacity—CXL latency rises up to 7.6× near saturation [36]. Third, migration traffic travels over the *same CXL link* as demand traffic, so an aggressive migration policy can push the link past the non-linear knee, raising demand latency rather than reducing it. We show that a CXL-oblivious migration policy degrades performance.

Contributions. We present *PRISM* (Page migration Robust against Interference from the Shared-link Mechanism), with the following key contributions:

(1) ROB Stall Events as The Primary Hotness Signal. *PRISM* scores pages by how often they caused ROB head-load stalls, directly measuring pipeline urgency and correctly accounting for MLP.

(2) Stall Variance Aware (SVA). CXL's non-linear latency means any access during link congestion incurs high stall latency, contaminating ROB-stall signals with *congestion victims*. SVA uses per-page stall variance to suppress such pages, reserving DDR slots for pages that are inherently hot.

(3) Cross-Core Fan-Out Weighting (CCF). A page that simultaneously stalls N cores causes $N\times$ pipeline damage on the shared CXL link. CCF boosts migration priority proportionally to the fan-out of cores stalled on the same page, particularly benefiting data-parallel workloads where multiple cores concurrently access shared model-weight pages or graph adjacency structures.

(4) Stability Bonus (SBS). A multiplicative bonus that grows with consecutive intervals of unchanged tier residency prevents thrashing and stabilizes the DDR working set between migration rounds.

(5) Migration Bandwidth Cap. *PRISM bounds* per-interval migration volume to prevent link saturation from over-migration, since migration and demand traffic share the same CXL link.

PRISM is evaluated on 13 workloads (graph analytics, ML inference, and GBDT training) on a 16-core system modeled after Intel Sapphire Rapids [13], using an in-house CXL 3.1 simulator validated against published hardware measurements (within ±8% on unloaded latency; §5.1). *PRISM* achieves 16% geomean performance improvement over the state-of-the-art, requiring negligible on-chip

storage (≈ 0.03% of the LLC) and a 13 B-per-page side table (0.32% metadata overhead).

2 Background

2.1 CXL-Tiered Memory Architecture

Compute Express Link (CXL) is a cache-coherent interconnect built on the PCIe physical/link layer that allows CPUs to address attached memory modules as regular cacheable DRAM [19]; current deployed CXL memory devices commonly use PCIe 5.0-class links. A typical deployment pairs conventional DDR5 as the fast tier with one or more CXL DRAM modules as the slow tier. From the application's perspective both tiers appear as flat physical memory.

DDR tier. The DDR tier is directly attached to the CPU's on-die memory controller bus. Multiple independent DDR channels each serve a disjoint subset of the DIMM slots, providing per-channel parallelism with no shared-bus contention between channels.

CXL tier: a single shared PCIe link. CXL memory attaches via the standard PCIe point-to-point slot. Unlike DDR, there is no separate multi-channel fabric: in typical single-device deployments, each CXL memory module occupies *one* PCIe port, and all cores funnel their CXL accesses through that single physical link. There are no per-core, per-channel, or per-bank ports on the CXL side of the memory hierarchy [19, 22]. Commercial CXL memory expansion modules (e.g., Micron CZ120 [4], SK Hynix [2]) connect via a PCIe 5.0 ×8 port (eight serial lanes forming *one* physical link), yielding a peak aggregate bandwidth of roughly 32–37 GB/s per module. By contrast, a modern server CPU provides multiple independent DDR5 channels, each delivering 38–51 GB/s to its own DIMM slots. The CXL link's single-pipe bandwidth is thus shared across all cores competing for CXL memory, while DDR channel queues remain independent of one another.

Superlinear latency-bandwidth curve. At low utilization, a CXL access incurs ∼225 ns end-to-end [51]. Past ≈70% utilization, latency rises *super-linearly*—up to 7.6× the base CXL latency on measured hardware [36], with p99.9 latencies rising even more sharply [34, 51]. A key protocol-level contributor is PCIe's credit-based flow control [19, 42]: under heavy read load, completion-buffer credits drain faster than they are replenished, stalling new requests and compounding queuing latency.

Key CXL-specific constraints. We identify three constraints that directly motivate *PRISM*'s design:

1. *Single shared link.* All cores share one PCIe link to CXL memory. On DDR, N independent channel queues distribute queuing pressure across parallel paths. On CXL, a single heavily-stalling page serializes every core's requests behind it—a compounding bottleneck when parallel threads share hot data structures.
2. *Congestion-inflated latency.* Because link latency varies dramatically with load, a page's observed performance impact is determined as much by link congestion at the moment of access as by the page's inherent hotness. A cold page touched once during a congestion episode may appear to hurt performance just as badly as a genuinely hot page touched frequently at low load. Policies that identify hot pages by measuring their *observed effect on performance* (e.g., tracking access latency or stall time) [53] will therefore confuse transient congestion victims with genuinely hot pages—a pitfall absent on dedicated-bus tiers (HBM, NVM) where link latency is relatively stable.
3. *Migration competes for the same link.* Promoting a page requires reading its contents from CXL, so migration and demand traffic share the same serialized PCIe link. An aggressive migration policy can push the link past its non-linear latency knee, raising demand latency and undermining the very benefit it is trying to deliver. DDR-based tiered systems face the same competition in principle, but DDR's aggregate multi-channel bandwidth leaves enough headroom that migration rarely causes saturation.

2.2 Identifying Hot Pages: SW vs. HW

Page migration periodically promotes hot pages from CXL to DDR and demotes cold pages back to CXL. The migration system maintains a per-page hotness *score*—most prior systems use *access frequency* as this score—and keeps it up to date as the workload evolves. Two broad design tracks have emerged for *how* to track these frequency scores, with fundamentally different cost/capability trade-offs.

Software-managed tracking. The dominant deployed approach uses the OS or a hypervisor to track page hotness via data structures already present in the virtual memory subsystem. The basic mechanism relies on page table entries (PTEs): modern ISAs expose a hardware-set Accessed bit per PTE (x86-64's `A` bit, ARM's Access Flag), which software clears periodically and re-examines to infer access frequency [17, 30, 38, 55].

Rather than inferring frequency from a single bit, the exact access count can be stored in the PTE or a side table. IBM's POWER8 [50] already does this, storing reference counts in hardware page table entries. Prior work [10, 39] generalizes the idea: TLB entries are extended to accumulate LLC miss counts for each cached PTE; on TLB eviction, the accumulated count is written back to a per-page side table—enabling fine-grained frequency tracking with negligible on-chip storage overhead. Although these systems require modest hardware support (extended TLB entries and augmented page-table-walker logic for read-modify-write on eviction), placement and migration decisions remain under software control; we therefore classify them as software-managed, consistent with prior taxonomy.

The main cost of such tracking is migration latency: software-managed page moves require a system-wide TLB shootdown to maintain coherency of virtual-to-physical mappings across all cores. The remedy is to batch migrations into long intervals (100 ms–1 s), amortizing the shootdown overhead across many moves [10, 38, 55].

Hardware-managed tracking. The alternative keeps hotness metadata entirely in hardware—typically in tables managed by the memory controller—and migrates pages autonomously without OS involvement. Representative proposals use hardware remapping tables [16] or clustered memory controllers [43]. Some industry products also move tiering decisions into hardware [6].

The primary cost of such tracking is *metadata storage*: tracking all pages in a large CXL pool requires per-page counters that must be held in on-chip SRAM, which becomes prohibitive at memory capacities of tens or hundreds of gigabytes [39].

This paper. *PRISM*'s design is agnostic to the tracking substrate—it can be implemented in either hardware or software. In this paper,

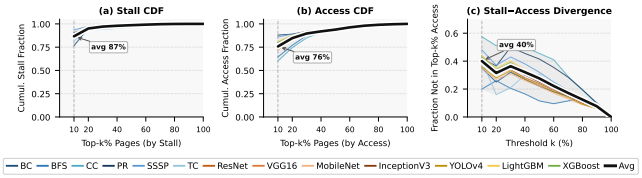


Figure 1: Stall (a) and access (b) CDFs per page; cross-rank divergence (c).

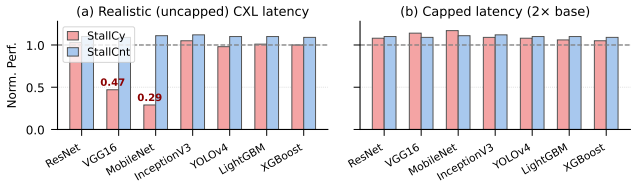


Figure 2: *StallCy* degrades under congestion (a); 2× cap removes it (b).

we adopt the software track to align with the dominant direction in recent research and production systems [10, 38, 55].

2.3 The ROB Head-Load Stall Signal

An OOO processor retires instructions strictly in order from the ROB head. When the oldest uncommitted instruction is a pending load, no instruction can retire—an *ROB head-load stall*. A page that repeatedly causes such stalls is the active bottleneck.

ROB stall signals have been exploited in prior work, but at a different granularity and for different purposes. Ghose et al. [21] predict ROB-blocking loads per load PC and feed priority hints to the memory scheduler to service critical requests sooner; CLIP [41] predicts load criticality per PC to filter prefetches on bandwidth-constrained systems. Attributing stalls to the load PC is natural: the ROB is an *instruction*-ordering structure, and the issuing PC is the most direct handle on the stall.

PRISM goes one step further: it attributes stalls to the accessed *physical page* and accumulates per-page stall counts across migration intervals as a long-term hotness score for tier placement. Pages that are structural bottlenecks—shared model-weight tensors, hot graph adjacency lists—remain so across intervals, making this a reliable long-term signal. *PRISM* also rejects stall *duration* (as in [21]) as a primary score: on CXL, the shared PCIe link inflates stall cycles for cold pages during congestion, corrupting the signal (§3.2).

3 Motivation

3.1 Stall-Critical ≠ Frequently Accessed

Figs. 1-(a) and 1-(b) characterize how ROB-head stall cycles and memory accesses are distributed across pages for workloads on a two-tier DDR/CXL system (see §5 for the experimental setup).

Stalls are highly concentrated. Fig. 1-(a) plots the cumulative distribution function (CDF) of total ROB-head stall cycles as pages are added in decreasing order of stall contribution. Across all workloads, the top 10% of pages account for 76.5%–93.5% of all processor stalls. This skew is the key premise of stall-driven migration: pages contribute very *unequally* to processor stalls, so promoting the small set of stall-critical pages to the fast tier can substantially reduce stall cycles.

Access frequency shows similar concentration. Fig. 1-(b) shows the analogous CDF for memory accesses (LLC misses), ranking pages by access frequency. A similar concentration appears: the top 10% of pages account for 60.1%–88% of all accesses. *Taken alone*, this might suggest that access frequency is a reliable proxy for performance—the implicit assumption behind frequency-based migration policies [7, 8, 10, 17, 30–32, 38, 39, 46, 55, 58].

The two rankings disagree substantially. Fig. 1-(c) tests that assumption directly. For each threshold k, it plots the fraction of the top-k% stall pages that are *absent* from the top-k% access pages. If frequency perfectly predicted stall impact, this curve would be zero everywhere. Instead, at k=10%, 20%–57.5% of the highest-stall pages do *not* appear among the highest-access pages. Even at k=50%, divergence remains substantial (up to 45.5% for CC).

Implication. A migrator selecting pages by access frequency serves *most accesses* from DDR yet misses pages that *actually stall* the processor. The disconnect arises because OOO cores overlap accesses via MLP—these raise access frequency but contribute little to ROB stalls. What matters is *non-overlappable* latency that stalls the pipeline—captured by the ROB-head stall signal. Unlike access count, which aggregates overlappable and non-overlappable events, ROB head-load stall count measures only unhideable events, isolating pages that create bottlenecks. This motivates using the signal directly, but it must be interpreted carefully on CXL, as shown next.

3.2 Link-Induced Stall Contamination

Fig. 2-(a) compares two stall-driven policies against a frequency-based baseline on a sensitive subset of our workloads: *StallCy*, which promotes pages with the highest cumulative stall cycles, and *StallCnt*, which promotes pages with the highest stall event count. On two workloads, *StallCy* drops to 0.47× on `VGG16` and 0.29× on `MobileNetV2`; whereas *StallCnt* shows no slowdown.

Root cause. The CXL link's non-linear latency-bandwidth curve means that during congestion *all* accesses—hot and cold alike—see elevated latency. A cold page touched once during peak congestion accumulates thousands of stall *cycles*. A cycles-based migrator cannot distinguish page hotness from link congestion: it promotes cold pages that triggered one long stall during a congestion event, wastes DDR slots, and creates churn that makes congestion worse. Fig. 2-(b) confirms the root cause with a controlled experiment: we artificially cap the CXL congested latency at 2× of its base latency, suppressing the non-linear blow-up. This is a causal validation: if congestion-induced latency inflation were the root cause, removing it should eliminate the slowdown—and it does. Under this synthetic model, *StallCy* not only avoids slowdowns but slightly outperforms *StallCnt*, showing that cycle-based scoring works correctly when CXL latency does not inflate unpredictably—as is the case in dedicated-link tiered memory (e.g., HBM) where latency remains relatively stable under load.

Key insight. Use stall event *count* rather than stall *cycles* as the primary hotness score. Count measures *how often* a page urgently blocked the ROB, independent of how long each event lasted under CXL congestion. Note that this does not mean stall cycles are useless: §4.3 shows that the *variance* of stall cycles across intervals is a reliable congestion-victim detector, precisely because variance captures the *pattern* of stall behavior rather than its magnitude.

3.3 Bimodal CXL Latency

Fig. 3 shows, for `VGG16`, the distribution of per-request ROB stall durations incurred while waiting for CXL to service an LLC miss.

The distribution is clearly bimodal: a low-latency mode at ~650cy (link at low load) and a high-latency mode at ~1450cy (dominated by queuing delay under link saturation).[1]

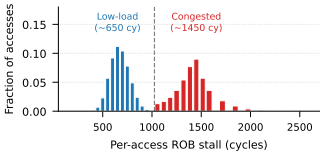


Figure 3: Bimodal latency on VGG16.

Root cause. A page whose stall latency alternates between modes (low-latency at ~650cy and congested at ~1450cy) has *high variance*: sometimes fast (low load), sometimes slow (congested). This is a *congestion victim*—slow because the link is busy, not because the page is inherently cold or hot. Migrating a congestion victim to DDR yields at most modest benefit: link congestion remains, and another page becomes the new victim.

Key insight. A page with *low variance and high mean stall* is inherently stall-inducing regardless of link load. These pages benefit from DDR residency. Tracking *per-page stall variance* allows us to discriminate true hot pages from transient congestion victims.

3.4 Multi-Core Damage Amplification

Parallel execution naturally generates multi-core memory stalls: concurrent threads share data structures that reside on the same physical pages, and when those pages sit on CXL, *every* core's request queues on the single shared link. Fig. 4-(c) quantifies how often this occurs: it shows the fraction of migration intervals in which a page stalled more than one core *simultaneously*. In ML workloads this effect is especially pronounced, as threads share model-weight pages; in graph analytics, parallel traversal of the same adjacency lists produces analogous sharing patterns.

Why the shared link multiplies damage. On DDR, multiple independent channels serve concurrent requests in parallel (Fig. 4-(a)). On CXL, there is one shared link, so *every* CXL request from *every* core queues behind others, regardless of page (Fig. 4-(b)). A page stalling N cores simultaneously causes compounding damage on the shared CXL link: each of the N requests queues behind others, so waiting time grows with fan-out. Total pipeline damage grows with N more steeply than on DDR, where independent channels absorb concurrent requests in parallel and keep latency roughly stable. Thus, two pages with identical per-core stall counts are *not* equally costly: a page stalling 1 core N times is far less costly than one stalling N cores simultaneously.[2]

Design implication. A page that simultaneously stalls many cores on the shared CXL link causes disproportionate damage. Migration priority should be boosted proportionally to the number of cores that stalled on the page in the current interval.

3.5 Migration-Induced CXL Link Load

Fig. 5 shows the breakdown of CXL link bandwidth between demand and migration traffic, using a frequency-based migration, as the interval length is swept from 10^6 to 10^{10} LLC misses (the box marks 10^8, the default used throughout this paper). Uncapped

[1]This figure shows a *representative* operating point, not peak saturation. Under extreme link load, the latency can rise even further. E.g., Intel MLC [27] saturation traffic on an FPGA-based CXL device results in a 7.6× rise in latency [36].

[2]In the worst case, all N requests serialize on the shared link, so waiting time grows roughly with fan-out N; core 1 waits L, core 2 waits $2L$, …, core N waits $N{\times}L$

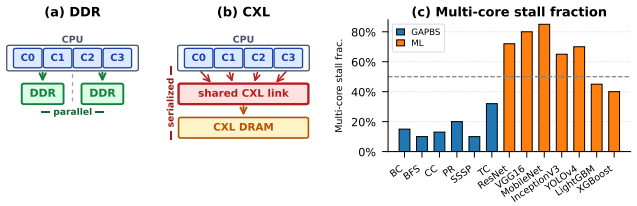


Figure 4: DDR: parallel channels (a); CXL: shared link (b); multi-core stall fraction (c).

migration—the default in most hybrid-memory systems [8, 10, 17, 30–32, 38, 39, 55]—consumes up to 20% of total CXL bandwidth.

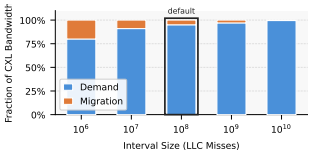


Figure 5: Demand vs. migration BW.

Why migration is expensive on CXL. Migration is bandwidth-intensive regardless of memory technology: promoting a page transfers its entire contents across the interconnect. Interaction with demand traffic depends on the migration model. Some systems perform *asynchronous* migration, copying pages in the background while other cores execute; a brief TLB shootdown is issued only at the end to atomically remap the page [17, 38, 55]. In this model, migration and demand traffic coexist on the link simultaneously, and contention is a first-order concern. Other systems [10, 31, 39] *pause execution* during the page transfer to simplify coherency. In this model there is no concurrent demand traffic, but every stalled cycle is now directly proportional to how long the page transfer takes. In this paper, including in Fig. 5, we assume the latter as the default, though we analyze the former too (see §6.3).

In both cases, CXL amplifies the cost relative to HBM+DDR. In the asynchronous case, demand and migration traffic must *share a single serialized PCIe link*, whereas HBM+DDR spreads the load across many independent channels, diluting the contention. In the synchronous case, the single CXL link has lower peak bandwidth than DDR's aggregate multi-channel bandwidth, so transferring the same number of pages takes longer, inflicting a *more severe execution pause* on every migration round. A migration policy must therefore be conservative about how many pages it promotes per interval, regardless of whether execution is halted or not.

Bandwidth cap. The migration policy must be aware of the bandwidth cost of its own actions. A per-interval *page migration cap* [46, 58] derived from a configurable bandwidth budget prevents self-defeating over-migration on the shared CXL link.

4 *PRISM* Design

PRISM addresses the CXL-specific challenges identified in §3 with a unified per-page scoring function evaluated once per migration interval. All components combine into a single score; the top-K CXL pages by score are promoted to DDR and the bottom-K DDR pages are demoted to CXL, subject to a per-interval bandwidth cap.

4.1 Overview

At each interval boundary, *PRISM* assigns every tracked page a scalar *hotness score*:

$$\text{score}(p) = \underbrace{S_{\text{FSC}}(p)}_{\text{base score}} \cdot \underbrace{f_{\text{SVA}}(p)}_{\text{variance filter}} \cdot \underbrace{w_{\text{CCF}}(p)}_{\text{fan-out weight}} \cdot \underbrace{m_{\text{SBS}}(p)}_{\text{stability bonus}} \quad (1)$$

The four components target CXL challenges (§§4.2–4.5); the bandwidth cap (§4.6) bounds total per-interval migration volume. The algorithm runs in software, either kernel or hypervisor runtime; attributing ROB head-load stall counts to individual pages requires the software or hardware instrumentation described in §4.7.

4.2 Frequency-Stall Count (FSC)

As discussed in §3.1, ROB stall events better reflect pipeline urgency than access frequency. However, scoring pages by stall event count $E(p)$ alone risks *undervaluing hot pages with lower access latency*—a DDR-resident page accumulates fewer stall events than an equally-accessed CXL page. Replacing stall events with stall *cycles* introduces congestion contamination (§3.2). FSC addresses both pitfalls by combining stall count with access frequency:

Design. Let $E(p)$ denote the stall event count and $A(p)$ the access count accumulated for page p during the current interval. The FSC base score is:

$$S_{\text{FSC}}(p) = E(p) + \alpha \cdot A(p) \quad (2)$$

Both terms apply to pages in either tier. $E(p)$ captures pipeline urgency. Any page, DDR or CXL, contributes to $E(p)$ when one of its accesses blocks the ROB head, though CXL-resident pages do so more often due to their higher access latency. The $\alpha \cdot A(p)$ term ensures that pages with high access demand but fewer stall events—e.g., DDR-resident pages whose lower access latency reduces the probability of blocking the ROB head—are not invisibly displaced by rarely-accessed pages with marginally higher stall counts.

Calibrating α. Setting $\alpha = 1/L_{\text{CXL}}$, where L_{CXL} is the CXL access latency in CPU cycles under *typical* conditions—meaning moderate utilization, between the unloaded floor and the peak-congestion ceiling—weights L_{CXL} accesses equally to one stall event. We default to $L_{\text{CXL}} = 1000$ cycles ($\alpha = 0.001$), which at 2.9 GHz is ≈345 ns—roughly 1.5× the ≈225 ns unloaded latency, representative of the moderate-load regime. For DDR-resident pages, $E(p) \approx 0$ (DDR latency is low enough that the OOO core rarely stalls at the ROB head), so $\alpha \cdot A(p)$ becomes their dominant score term, preventing them from scoring near zero and being invisibly displaced by infrequently-accessed CXL pages. For hot CXL pages ($E(p) \gg \alpha \cdot A(p)$), the score is dominated by stall count; $\alpha \cdot A(p)$ acts as a frequency tiebreaker. §6.3 confirms this default is robust over a wide range.

4.3 Stall Variance Aware (SVA)

Even with stall count, a congestion victim accessed during a congested interval accumulates stall events, inflating $E(p)$. SVA suppresses pages whose stall behavior is bimodal (fast at low load, slow under congestion)—migrating such congestion victims rarely improves performance, since link congestion persists. SVA applies to all pages but is most consequential for CXL-resident pages, where link congestion makes the bimodal signature most pronounced.

Design. SVA tracks the *variability* of a page's stall behavior over time. A *congestion victim* incurs few ROB stall cycles at low link load but many under saturation—its per-interval stall cycle total fluctuates widely (see §3.3). An *inherently hot page* incurs consistently high ROB stall cycles regardless of link load; its stall cycle is high but stable. SVA distinguishes these two cases by estimating the coefficient of variation (CV) of per-interval stall cycles: a high CV flags a congestion victim; a low CV confirms inherent hotness.

Let $C(p)$ denote the per-interval stall cycle total for page p. SVA maintains two exponential moving averages (EMAs) across intervals—a running mean μ_p and a running mean-of-squares ν_p—from which variance is estimated:

$$\mu_p \leftarrow (1-\lambda) \cdot \mu_p + \lambda \cdot C(p) \quad (3)$$

$$\nu_p \leftarrow (1-\lambda) \cdot \nu_p + \lambda \cdot C(p)^2 \quad (4)$$

where $\lambda \in (0, 1)$ (default $\lambda = 0.3$; insensitive in $[0.1, 0.5]$, see §6.3) controls how quickly the estimate adapts to recent behavior. The CV is then:

$$\text{CV}(p) = \frac{\sqrt{\max(\nu_p - \mu_p^2,\ 0)}}{\mu_p + \epsilon}, \quad \epsilon = 1 \quad (5)$$

where $\epsilon = 1$ prevents division by zero for pages with negligible stall history. A high CV means the page's stall latency swings between fast and slow—a congestion victim fingerprint. The CV is converted into a multiplicative suppression factor applied to the FSC score:

$$f_{\text{SVA}}(p) = \frac{1}{1 + \beta \cdot \text{CV}(p)}, \quad \beta = 1.0 \quad (6)$$

$\beta = 1.0$ is calibrated so that a page with CV = 1 (standard deviation equals its mean, a useful threshold for highly variable stall behavior) receives a 50% reduction in score. A congestion victim with high CV gets $f_{\text{SVA}} \ll 1$, suppressing its migration priority; a consistently-slow page with low CV passes with $f_{\text{SVA}} \approx 1$, leaving its score intact.

EMA persistence and orthogonality to FSC. μ_p and ν_p *persist* across interval resets (initialized to zero at page allocation and never reset thereafter): a congestion victim is only identifiable by contrasting stall durations across low-load and congested intervals.[3] SVA uses stall *cycles* for variance estimation while FSC uses stall *count* for the base score. Contamination arises when a single large cycle count inflates a *score*; variance captures the *pattern* across many intervals. A congestion victim shows high variance (fast at low load, slow when congested); an inherently hot page shows low variance (consistently slow regardless of link load). The two signals are complementary: FSC measures urgency; SVA checks whether that urgency is a reliable cross-interval property.

4.4 Cross-Core Fan-Out Weighting (CCF)

As discussed in §3.4, a page stalling N cores simultaneously causes $N\times$ pipeline damage on the shared PCIe link. FSC and SVA are per-page signals that do not distinguish single-core from multi-core stalls; CCF provides this distinction.

Design. Let N_c be the number of cores ($N_c = 16$ in this paper) and $M(p) \subseteq \{0, \ldots, N_c - 1\}$ be the set of core IDs that reported an ROB head-load stall on page p during the current interval, reset at each interval boundary. $M(p)$ suggests concurrent multi-core

[3] Persistence does not mean indefinite memory: if a page's stall activity ceases, μ_p and ν_p decay geometrically at rate $(1-\lambda)^k$ per idle interval, so formerly hot pages lose their signal naturally without any explicit reset.

pressure at interval granularity: a core is included if it stalled on p at any point in the interval. This approximation does not distinguish truly simultaneous stalls from sequential stalls within the interval, but it is a reliable proxy for thread-shared pages, which almost always show up in multiple cores' stall sets within the same interval.[4] Pages driving high $|M(p)|$ are shared across threads. When core i reports a stall:

$$M(p) \leftarrow M(p) \cup \{i\} \tag{7}$$

The fan-out weight is:

$$w_{\mathrm{CCF}}(p) = 1 + \gamma_{\mathrm{ccf}} \cdot |M(p)|, \quad \gamma_{\mathrm{ccf}} = 1.0 \tag{8}$$

A page stalling all 16 cores across the interval gets $w_{\mathrm{CCF}} = 17$, strongly prioritizing it over a page that stalled only one core with the same aggregate stall count.[5]

4.5 Stability Bonus (SBS)

Without hysteresis, a recently promoted DDR page can be immediately demoted to make room for a CXL page with a marginally higher score, creating thrashing that consumes CXL bandwidth without improving the steady-state hit ratio. SBS addresses this by giving incumbent pages a score bonus that grows with the number of consecutive intervals they have remained in their current tier. A page consistently hot across many intervals is more trustworthy than one that spiked in the current interval—evicting a long-stable DDR page forces a round-trip over CXL to restore it, a cost high enough to warrant protecting it against marginal challengers.

Design. Let $c(p)$ denote the number of consecutive intervals page p has remained in the same memory tier (reset to zero on any migration). The stability multiplier is:

$$m_{\mathrm{SBS}}(p) = 1 + \gamma_{\mathrm{sbs}} \cdot \min(c(p),\, I_{\max}), \quad \gamma_{\mathrm{sbs}} = 0.05,\ I_{\max} = 10 \tag{9}$$

$\gamma_{\mathrm{sbs}} = 0.05$ is selected by sensitivity sweep (§6.3): it is small enough that a single stable interval does not grant excessive protection, yet large enough to build the full bonus within a reasonable number of intervals. m_{SBS} applies symmetrically to DDR and CXL pages; a long-dormant CXL page with $c(p) = I_{\max}$ receives the same multiplier, but this is harmless: such a page has $E(p) \approx 0$ (no stalls observed), so its promotion score remains near zero regardless of the multiplier. A page seeking promotion is typically newly hot—$c(p) \approx 0$, no SBS bonus—so the effective differential is one-sided. $I_{\max}$ is set so that the maximum bonus—$1 + \gamma_{\mathrm{sbs}} \cdot I_{\max} = 1 + 0.05 \times 10 = 1.5\times$—provides substantial demotion resistance: a DDR page stably resident for $I_{\max}$ intervals can withstand a newly-hot CXL challenger with up to 1.5× higher base score before being demoted. The 1.5× cap is chosen to balance two requirements: it must be large enough to absorb inter-interval score noise (a challenger that is only marginally hotter should not trigger a round-trip migration), yet small enough that a page that has genuinely gone cold (accumulating little to no stall signal) will still be displaced despite its bonus. §6.1 and §6.3 confirm that SBS reduces migration rate without sacrificing DDR hit ratio.

4.6 Migration Bandwidth Cap

Motivated by §3.5, *PRISM* enforces a per-interval page cap from a configurable bandwidth budget ($B_{\max} = 1024$ MB). The cap stays well below the CXL device's 32GB/s peak bandwidth. Each interval spans 100M LLC misses; only misses to CXL pages generate CXL traffic, so per-interval demand is at most $100\mathrm{M} \times 64\,\mathrm{B} = 6.4$ GB reads (up to 12.8 GB including writebacks) when the DDR hit ratio is zero, and lower DDR absorbs misses. Relative to worst-case all-CXL demand, the 1024 MB cap limits migration to 8–16%; relative to actual demand (lower when DDR absorbs misses), the fraction can be higher; §6.3 evaluates sensitivity.

In the *asynchronous* model (§3.5), where migration and demand share the link, this keeps total link utilization below the 70% congestion knee (§2) in our workloads, leaving ample headroom for demand traffic. In the *synchronous* model, where execution pauses during migration, the same 1024 MB cap costs at most ≈31 ms of pause per interval—a small fraction of the intervals in our workloads, which span 600–2000 ms wall-clock time.[6]

4.7 Implementation Overhead

PRISM is implemented via the software track (see §2.2). It uses a kernel-managed per-page side table of `structs` indexed by physical frame number, similar in spirit to *MEMTIS* [32]. The table is populated via a TLB-writeback mechanism following prior HW/SW tiering designs [10, 39, 57]. *PRISM* extends this infrastructure with four additional fields per entry—stall count $E(p)$, stall cycle accumulator $C(p)$, core mask $M(p)$, and EMA state μ_p/ν_p.

TLB extension. Each L1 dTLB entry is extended with: an 8-bit stall event counter (E), a 16-bit access counter (A), a 16-bit stall cycle accumulator (C, for SVA), and a N_c-bit core mask (M, for CCF). On LLC miss, A is incremented; on ROB head-load stall, a combinational path from the load queue increments E, accumulates C, and sets bit i in M for the matching TLB entry. On eviction, all four fields are written back to the side table—same as in [10, 39, 57]. Total on-chip cost is $(8{+}16{+}16{+}N_c)$ bits per entry; with N_c=60 cores and 48 dTLB entries per core: $60 \times 48 \times (8{+}16{+}16{+}60)$ bits = $60{\times}48{\times}100$ bits ≈ 35 KB total in Intel Sapphire Rapids [13]—only 0.03% of the LLC. With N_c=16 (the scaled-down simulated system, §5), the overhead is $16 \times 48 \times (8{+}16{+}16{+}16)$ bits ≈ 5 KB.

Side-table. Packing fields tightly—$E(p)$ and $A(p)$ as 16-bit per-interval counters (wider than the TLB's 8-bit fields, since they accumulate across multiple TLB writebacks over an interval), $C(p)$, μ_p, ν_p as 16-bit saturating fixed-point EMAs (cycle counts scaled to ~1000-cycle units before storage; quantization sensitivity is negligible), $M(p)$ as a $\lceil N_c/8 \rceil$-byte core mask, and $c(p)$ as an 8-bit stability counter—each entry is $11{+}\lceil N_c/8 \rceil$ bytes: 13 B on a 16-core system, 19 B on a 60-core system. At 13 B per page, the per-page metadata overhead is at most $13\,\mathrm{B}/4\,\mathrm{KB} = 0.32\%$—on par with

[4] When a page is touched thousands of times per interval by every thread (e.g., shared model weights in ML inference), access windows inevitably overlap, making interval co-presence a reliable indicator of simultaneity. Prior work uses the same proxy: Cruz et al. [18] track a per-page thread-ID bitfield per epoch for placement; Lepers et al. [33] identify shared pages from the set of NUMA nodes accessing them per interval. Our CCF ablation (§6.3) empirically confirms $|M(p)|$ is a consistent signal.

[5] Multiple cores can simultaneously stall on the same page because a page spans many cache lines—cores may miss on *different* lines within the page.

[6] Our workloads have 17–55 LLC MPKI. At 2.9 GHz, 16 cores, per-core CPI ≈ 10–20 (CXL-bound workloads stall ~1000+ cycles per access), interval = $100{\times}10^6 / \lceil (\mathrm{MPKI}/1000){\times}(2.9{\times}10^9/\mathrm{CPI}){\times}16 \rceil$. MPKI ≈ 55, CPI ≈ 15 ⇒ $\approx 170{\times}10^6$ misses/s ⇒ interval ≈ 590 ms; MPKI ≈ 22, CPI ≈ 20 ⇒ $\approx 51{\times}10^6$ misses/s ⇒ interval ≈ 2000 ms.

MEMTIS [32] (16 B per page via `struct page` extensions) and well below *HeMem* [46] (168 B per page).

Portability. All *PRISM* instrumentation is CPU-side; the CXL device is treated as a standard `CXL.mem` endpoint with no firmware modifications or in-device counters required, making it compatible with any standards-compliant CXL memory device. *PRISM* also requires no per-workload training or tuning: its components are designed to be complementary (SVA filters congestion victims, CCF amplifies shared hot pages, SBS smooths inter-interval noise), so that pathological behaviors in one dimension are corrected by another, yielding consistent gains across *all* workloads (§6). Also, in §6.3, we show that *PRISM*'s advantage is preserved across wide sweeps of all parameters (e.g., α, CCF γ, bandwidth cap, etc.), confirming that no per-workload tuning is required.

Intervals. The side table is a dense array indexed by physical frame number; the interval-boundary sweep is a single sequential pass that also zeros per-interval fields (E, A, C_{interval}, M). At 13 B per page, the array spans $\leq 3.25F$ MB for an F GB footprint. For our largest workloads, this array reaches ≈50 MB; DDR bandwidth delivers it in under 1 ms—less than 0.1% of our intervals (§6.3).

5 Methodology

5.1 Simulation Infrastructure

For evaluations, we extend ZSim [47] with our in-house CXL simulator. We find existing open-source CXL simulators inadequate for evaluating tiering policies under realistic link congestion. We briefly overview their limitations and then explain our approach.

CXLMemSim [56] injects coarse per-epoch latency penalties derived from PEBS LLC-miss counts, with no protocol state machine, flit serialization, or credit management, and an incomplete bandwidth model. CXL-DMSim [54] is built on gem5 [12] and validated against real CXL 1.1 silicon, but does not model CXL 3.0/3.1-specific protocol features [19]: the 256-byte flit format, the latency-optimized (LO) sub-flit pipeline that eliminates store-and-forward delay, piggybacked link-layer credit return (LLCRD) in data flits, and the revised link-level retry (LLR) mechanism. CXL-DMSim also models congestion as simple FIFO queue saturation rather than NP/Cpl credit-pool exhaustion [19] (detailed below) that drives superlinear latency on real hardware, and inherits gem5's LLC model, which the authors themselves report has up to 56.4% error in certain working-set ranges. MESS [20] fits bandwidth–latency curves from Micron's proprietary SystemC model into ZSim/gem5 as static lookups; but the model has no protocol internals whatsoever, and its extrapolation above the measured knee is linear—precisely where CXL behavior becomes superlinear and most consequential for tiering. DRackSim [44] and ESF [9] extend gem5 for disaggregated topologies but replace CXL protocol details with generic parameterized interconnects. To our knowledge, none of the simulators we surveyed fully models: *(i)* NP/Cpl/P credit-pool dynamics [19], the primary driver of the superlinear latency–bandwidth degradation measured on real hardware [36, 51]; *(ii)* flit-level serialization in any format [19]; or *(iii)* CXL 3.0/3.1 protocol features that govern congestion behavior in CXL 3.0/3.1 devices [19].

Our approach. We develop an in-house CXL 3.1 simulator integrated into ZSim [47], focusing on the protocol features as described next.

Flit format and base latency. CXL 3.0/3.1 introduces a latency-optimized (LO) 256B flit [19] enabling cut-through forwarding, reducing per-hop serialization delay vs. prior CXL generations. We model CXL 3.1 protocol-layer features (256B LO flits, NP/Cpl credit pools, LLCRD piggybacking) on top of a PCIe 5.0 ×8 link rate; modeling PCIe 6.0 rates is left to future work as no PCIe 6.0 CXL silicon is yet shipping. This combination matches deployed and near-term CXL memory devices [3, 4] and yields unloaded CXL read latency ≈215–240 ns—consistent with 214–271 ns measured on CXL 2.0 devices [36, 51]; the protocol additions below affect load behavior, not base latency. We also model link-level retry (LLR) [19]; without it, tail latency under stress is optimistic. The simulator supports configurable lane widths; default is PCIe 5.0 ×8 (≈32 GB/s), matching the PCIe 5.0 ×8 CXL port of SPR [13] and the lane width of mainstream CXL memory expansion devices [3, 4]; lane-width sensitivity is evaluated in §6.3.

Credit-based flow control. The key CXL 3.1 mechanism is credit-based flow control [19, 42]: the host maintains separate Non-Posted (NP, reads) and Completion (Cpl, responses) credit pools; once either is exhausted under heavy read pressure, requests stall regardless of available bandwidth—a protocol-layer bottleneck our simulator attributes to NP/Cpl credit exhaustion, consistent with the superlinear curve described in §2. Prior simulators model congestion as bandwidth saturation or FIFO queuing, producing a linear knee; only credit exhaustion reproduces the abrupt upturn seen on real hardware [36, 51]. CXL 3.1 also changes how link-layer credits (LLCRD) are returned [19]: prior versions sent them in standalone flits; CXL 3.1 piggybacks them in every outgoing data flit, coupling credit-return rate to throughput, affecting the congestion knee. We model both credit pools and LLCRD piggybacking.

Model validation. Our simulated unloaded latency (≈215–240 ns) matches CXL 2.0 measurements [36, 51] to within ±8%; consistent with CXL 3.1, since unloaded latency is governed by the PCIe physical layer, with protocol additions affecting only the loaded regime. Under load, we validate against CXL 2.0, with up to ∼15% lower p99 tail latency at peak bandwidth. Per the CXL 3.1 specification [19], the larger credit pool and revised arbitration shift the congestion knee rightward relative to CXL 2.0, reducing tail-latency inflation at a given load; our CXL 2.0-validated model is therefore conservative. Above the congestion knee, our credit model reproduces the nonlinear upturn while MESS [20] extrapolates linearly—confirming that credit-pool dynamics, not just bandwidth curve shape, are necessary to evaluate tiering policies in the congested regime.

Table 1 lists system parameters. We model an SPR-like processor [13] paired with a CXL 3.1 memory-link model over PCIe 5.0 ×8. The core/cache/DDR subsystem is scaled down to 16 cores for simulation tractability; the core model follows Intel's Performance-core (Golden Cove) microarchitecture used in SPR—Intel reports a 512-entry ROB for this core [25]—and all aggregate parameters (LLC, DDR bandwidth) are scaled proportionally to 16 cores, as detailed below.

Table 1: Simulated system parameters.

Parameter	Value
Cores	16 2-threaded OOO, 2.9 GHz, 512-entry ROB
L1/L2/Shared L3	48KB/2MB/30MB
Memory tiers	4GB fast tier, 77GB/s peak DDR and 32GB/s peak CXL
DDR tier	DDR5-4800 [1, 28], CL40-tRCD39-tRP39-tRAS76

DDR bandwidth. A full SPR socket provides 8-channel DDR5-4800, totalling ≈307 GB/s. Scaling to 16 cores gives ≈82 GB/s; we

model 2-channel DDR5-4800 (77GB/s), which is the closest standard configuration [28].

CXL bandwidth. SPR exposes one CXL port over a PCIe 5.0 link [13], with per-port bandwidth depending on the lane configuration. We configure our CXL tier at 32GB/s (PCIe 5.0 ×8), matching the bandwidth of mainstream CXL DRAM expansion devices [19].

DDR and CXL capacity. DDR capacity is 4GB, small enough that all workloads spill into CXL, exercising the tiering policy under realistic pressure. The CXL pool is uncapped to model a large expansion pool, consistent with production CXL deployments where the slow tier is several times larger than local DRAM [38].

Hardware prefetcher. We model the two dominant prefetchers in SPR [5, 26]: a *stride prefetcher* (DCU) that detects strided patterns, and a *stream prefetcher* (MLC Streamer) that tracks 32 independent access streams per core—one forward and one backward per page.

We use a 4KB page size and Table 2 lists our workloads.

Table 2: Evaluated workloads with memory footprint.

Workload	Description	Footprint
GAPBS Graph Analytics (Twitter: 41.7M nodes, 1.47B edges, 64 trials)		
BC	Betweenness centrality, direction-optimizing BFS	12.2 GB
BFS	Breadth-first search, direction-optimizing	11.7 GB
CC	Connected components (Afforest)	12.2 GB
PR	PageRank, 20 iterations	12.0 GB
SSSP	Single-source shortest paths, delta-stepping, weighted graph	11.4 GB
TC	Triangle counting, intersection-based, undirected graph	5.8 GB
ML (ONNX inference: ResNet–YOLOv4; GBDT training: LightGBM, XGBoost)		
ResNet50	Image classification, ImageNet, 224 × 224, batch = 2048, 5000 iters	5.1 GB
VGG16	Image classification, ImageNet, 224 × 224, batch = 1024, 5000 iters	14.3 GB
MobileNetV2	Image classification, ImageNet, 224 × 224, batch = 4096, 10000 iters	10.3 GB
InceptionV3	Image classification, ImageNet, 299 × 299, batch = 2048, 5000 iters	4.5 GB
YOLOv4	Object detection, COCO (80 classes), 608 × 608, batch = 512, 5000 iters	15.2 GB
LightGBM	GBDT training, HIGGS (11M rows, 28 features), leaf-wise growth	4.7 GB
XGBoost	GBDT training, HIGGS (11M rows, 28 features), depth=25, 2000 rounds	10.9 GB

We evaluate three workload classes: graph analytics, ML inference, and GBDT training. For graph analytics, we run all six GAPBS [11] kernels on the Twitter dataset. For ML inference, we run five vision models—ResNet50 [24], VGG16 [49], MobileNetV2 [48], InceptionV3 [52], and YOLOv4 [14]. For GBDT training, we run LightGBM [29] and XGBoost [15]. To make these workloads compatible with ZSim, we use a C++ simulation harness, including a custom multi-threaded ONNX [40] inference engine and C++ implementations of the GBDT workloads.

Simulation length. All workloads run for 200 B instructions; the first 20% is warm-up and excluded.

Code Repository. The implementation of *PRISM*, all evaluated policies, the ML workload suite, and the CXL 3.1 simulation model are in: https://anonymous.4open.science/r/prism_anonymous_repo-F20E

5.2 Compared Policies

We compare two groups of policies: an *ablation chain* that isolates each *PRISM* contribution, and *state-of-the-art* systems representative of deployed and published CXL tiering approaches. To isolate signal quality from migration frequency, all methods except *TPP* use a fixed interval of 100M LLC misses. *TPP* retains its published 100 ms time-based interval because its PTE access-bit scanning and exponential decay factor are inherently calibrated to a fixed time window. §6.3 evaluates different interval lengths.

Ablation chain.

- ***FreqBased*** (baseline, 1.0×): Access-count-only migration. The *de-facto* approach in prior work [10, 30, 32, 38, 39, 55]; our normalization point. *FreqBased* is a *strong* baseline: recent hardware-assisted tiering systems [10] implement precisely this signal at hardware access-count granularity, and software-based systems [30, 38] approximate it via PTE access-bit scanning. Any improvement over *FreqBased* therefore reflects improvement over the broadest deployed class of tiering policies.
- ***StallCy*** (Stall Cycles): The raw stall-cycle policy from §3.2, here enriched with the FSC frequency tiebreaker ($\alpha \cdot A(p)$) and stability bonus (SBS); all other *PRISM* components disabled. Represents a naive application of the ROB stall signal to CXL without addressing congestion-cycle contamination.
- ***StallCnt*** (Stall Count): As *StallCy* but scores by stall event *count* rather than cycles—the same enrichment (FSC + SBS), with the cycles-vs-count distinction as the only variable; SVA and CCF remain disabled.
- ***VarAware*** (Variance-Aware): *StallCnt* + SVA. Adds per-page stall variance suppression on top of count-based scoring.
- ***PRISM***: The full system: all four components combined.

State-of-the-art comparisons.

- ***TPP*** [38]: CXL tiering deployed at Meta. Promotes hot pages identified via access-bit scanning (PTE A bit); demotes cold pages using an aging pass at 100 ms OS intervals. *TPP* applies an exponential decay so pages from prior phases lose score and become demotion candidates. *Simulation:* Per-page access count over a 100 ms simulated interval with the published decay factor, mapped to the equivalent instruction count.
- ***AutoTiering*** [30]: Access-frequency histogram approach that bins pages into hot/warm/cold tiers and promotes/demotes based on histogram thresholds. Like *TPP*, it uses PTE-bit scanning and is frequency-only. *Simulation:* We use the published histogram parameters (three tiers, promotion threshold at the top decile of per-page access counts) and promote the hot tier to DDR each interval. The interval length matches our standard 100M trigger.
- ***MEMTIS*** [32]: Uses PEBS-based LLC-miss sampling to build per-page access-count histograms and classify pages into hot/warm/cold tiers, with dynamic 4 KB/2 MB migration granularity. *Simulation:* Per-page access counts accumulated at each 100M-miss boundary; 2 MB promotion modeled by promoting all 512 sub-pages when the top-density threshold is met.
- ***NeoMem*** [58]: Embeds NeoProf, a hardware profiler in the CXL device controller, to track per-page LLC misses device-side; the OS daemon reads these counters for migration decisions. *Simulation:* We model *NeoMem* using per-page LLC misses for CXL-resident pages only; NeoProf counters are read at each 100M-miss boundary, matching the PRISM interval trigger. A key behavioral difference from *FreqBased* is that *NeoMem* scores only CXL-resident pages—it has no visibility into DDR accesses.

6 Evaluation

6.1 Main Performance Results

Fig. 6 shows instructions per clock (IPC) normalized to *FreqBased* (bars) and DDR hit ratio (fraction of memory accesses served from

the fast tier, dots) for all 13 workloads; the ablation chain appears left of *PRISM*, state-of-the-art systems appear right.

***FreqBased* misidentifies hot pages on ML workloads.** *FreqBased* represents the signal used by most deployed tiering systems [38] and is our (strong) baseline. On ML workloads, it promotes activation buffers—whose accesses are hidden by prefetching and OOO—while model-weight pages that block the ROB remain in CXL (§3.1). *FreqBased* achieves a *higher* DDR hit ratio than *PRISM* (68.4% vs. 61.4% on ML), yet performance is lower: DDR holds accesses the processor can hide, not ones that stall it. On graph workloads, skewed degree distribution lets frequency partially capture the right pages, but phase-driven instability leaves performance on the table: as the active frontier shifts between intervals, *FreqBased* demotes recently-active hub pages and must re-promote them next interval, consuming CXL bandwidth at the worst moment.

***StallCy* fails on ML workloads.** *StallCy* collapses to 0.47× on `VGG16` and 0.29× on `MobileNetV2` (§3.2). CXL congestion inflates stall-cycle totals for cold pages, causing oscillation and churn that lowers DDR hit ratio below *FreqBased* while wasting CXL bandwidth re-promoting evicted weight pages.

Ablation chain. *StallCnt* eliminates the slowdown and gains 8.3% geomean over *FreqBased*: stall event count is less sensitive to congestion inflation than stall cycles—a single congested access can add many cycles but only one event—with residual contamination handled by SVA. Adding SVA (*VarAware*) improves to 14.5% by suppressing pages whose stalls are inflated by transient link congestion—congestion victims with high stall variance across intervals (§3.3)—freeing DDR for stably-hot weight pages; with CCF, *PRISM* reaches 17.7% geomean over *FreqBased*. On ML, DDR hit ratio is 61.4% (7% below *FreqBased*), yet performance is far higher: DDR holds weight pages that block the ROB rather than activation pages hidden by OOO. CCF identifies pages stalling *multiple cores simultaneously*: shared model weights stall all 16 cores, while a single-core signal captures only a fraction of this damage (§3.4); *PRISM* improves ML performance by 23.9%.

In `GAPBS`, hub pages (high-degree vertices touched by every thread each iteration) are the primary migration target. CCF amplifies their scores since they stall all 16 cores simultaneously; SBS retains them across phase transitions, preventing the churn a frequency-only policy exhibits as the graph frontier shifts. *PRISM* improves graph performance by 10.9%; hit ratios run 13–31% vs. *FreqBased*'s 15–34%, with higher IPC because DDR holds the hub pages that dominate ROB stalls.

Overhead-wise, *PRISM*'s metadata updates are triggered by TLB evictions, which are infrequent: dTLB MPKI is 0.06 on ML and 0.8 on `GAPBS`, keeping metadata writeback *bandwidth* overhead negligible.

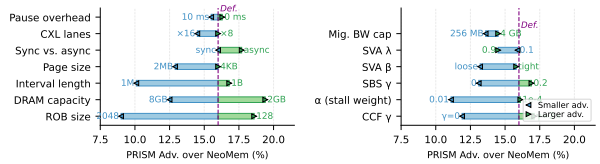


Figure 7: *PRISM* performance advantage over *NeoMem* per parameter variant.

6.2 Comparison Against State-of-the-Art

Fig. 6 also shows four state-of-the-art systems alongside *PRISM*; the first three use commodity hardware, *NeoMem* adds NeoProf in the CXL controller. None exposes per-page ROB stall counts, stall variance, or fan-out, so frequency-based scoring is the strongest signal each can compute; hit-ratio dots on ML often match *PRISM*'s, yet performance lags—the same decoupling seen in *FreqBased*.

***TPP*: recency decay, stall blindness.** *TPP* [38] achieves +0.9% geomean. Exponential aging frees DDR from stale-hot pages, but PTE `A`-bit scanning cannot distinguish model-weight pages from prefetcher-covered activation buffers (§3.1), dropping to 0.92× and 0.90× on `VGG16` and `MobileNetV2`.

***AutoTiering*: frequency histograms, congestion-blind.** *AutoTiering* underperforms *FreqBased* by 5.5%: its coarse bins and fixed hot/warm/cold thresholds are vulnerable to phase shifts and access skew, causing spurious promotions that displace genuinely stall-critical pages.

***MEMTIS*: PEBS sampling, frequency-limited.** *MEMTIS* underperforms *FreqBased* by 7.2% geomean (*PRISM* leads by 26.8%). 2 MB promotion granularity is the cause: up to 71.4% of promoted capacity is cold, evicting hot 4 KB pages; hub pages span a small fraction of each 2 MB block, and on ML the frequency-contaminated signal promotes wrong regions. Its scan window, calibrated for ~2× DDR-NVM latency, is too coarse for CXL's 5–6× congested regime.

***NeoMem*: hardware profiling, frequency-limited.** *NeoMem* [58] achieves 1.5% geomean—best among compared systems—with *PRISM* leading by 16%. NeoProf's device-side counts filter cache-hit inflation at 4 KB granularity, but the signal remains LLC miss count: it cannot determine whether a miss blocked the ROB or was covered by OOO, and is blind to DDR-resident pages. On phase-shifting and ML workloads, stale DDR pages block new promotions; on ML, activation buffers vanish from NeoProf's view, eroding performance.

Summary. *PRISM*'s 16% advantage reflects the limits of observable signals: all four systems rely on access frequency (PTE bits, LLC miss counts, or device-side miss counters), none of which expose per-page ROB stall counts, stall variance, or cross-core fan-out. On ML workloads, the hit-ratio decoupling is especially stark: SotA systems frequently match or exceed *PRISM*'s DDR hit ratio yet lag in performance, confirming that wrong pages—not too few pages—are filling DDR. Promoting more pages at a lower threshold

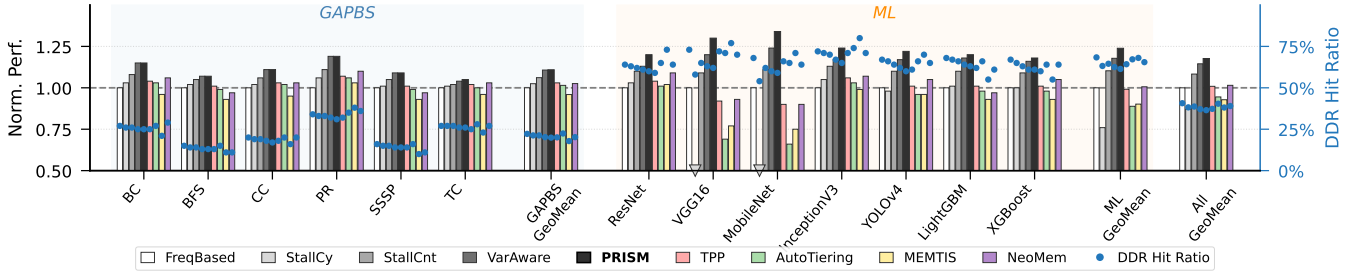


Figure 6: Norm. performance (bars) and DDR hit ratio (dots) for all workloads; geomean at right.

would raise the hit ratio further but still not place the right pages in DDR; closing this gap requires identifying which misses block the ROB, not which pages are accessed most.

6.3 Sensitivity Analysis

Fig. 7 sweeps each parameter and hardware knob; bars show *PRISM*'s geomean advantage over *NeoMem* (best prior), with the default config at the dotted line (16%); parameters are ordered by sensitivity range. The advantage is robust across the sweep: it narrows when hardware hides CXL latency (deep ROB, wide CXL link, ample DDR) or when algorithmic parameters reach their extremes, but these reflect changes in the CXL operating point, not fragile tuning.

ROB size. A smaller ROB (128 entries) sharpens the hotness signal (18.6%); a deeper ROB (2048 entries) hides more CXL latency and degrades it to 9%.

DRAM capacity. Tightening DDR to 2 GB raises the advantage to 19.4% (fewer slots sharpen the stall signal); expanding to 8 GB drops it to 12.5% as ML working sets increasingly fit in DDR.

Interval length. Shorter intervals (1M) reduce the advantage to 10.1% because SVA's EMA and SBS need several intervals to converge; longer intervals (1B) improve to 16.8%.

Page size. 2 MB huge pages reduce the advantage to 12.9%: a 2 MB migration bundles 512 sub-pages (most cold), wasting DDR on non-stall pages and conflating hotness in stall counts.

Migration execution model. Switching to asynchronous migration [38, 55] improves geomean by 1.7% percentage points (17.7% advantage over *NeoMem*) by eliminating stalls during page transfer, at the cost of coherency complications (§3.5) such as dirty-page tracking [55].

CXL lane width. Widening to ×16 (≈64 GB/s) shifts the congestion knee rightward, weakening the stall signal to 14.5%: credits exhaust less often and the congestion regime is entered later.

OS pause overhead. Eliminating the 1 ms default pause improves marginally to 16.3%; inflating it to 10 ms drops the advantage to 15.5%. *PRISM* is insensitive across the practical 0–10 ms range.

CCF γ_{ccf}. Disabling fan-out amplification (γ_{ccf}=0) reduces advantage to 11.9%; γ_{ccf}=2× yields 17.1%, confirming moderate amplification is near-optimal (§4.4).

α (stall-to-freq. weight). High α reverts to *FreqBased* (11.1%); low α raises unnecessary demotions (16.1%); the range $[0.0001, 0.005]$ is stable. $\alpha = 0.001 = 1/L_{\text{CXL}}$ needs no per-workload tuning.

SBS γ_{sbs}. Disabling SBS ($\gamma_{\text{sbs}} = 0$) reduces to 13.1% due to churn; γ_{sbs}=0.2 improves to 16.9% with no penalty at higher values.

SVA λ. Swept over $[0.1, 0.9]$: stable within $[0.1, 0.5]$ but degrades to 14.5% at λ=0.9, where fast adaptation reduces smoothing and noisifies CV estimates.

Migration BW cap. Both extremes hurt: a tight cap (256 MB) starves migration (13.6%); a loose cap (4 GB) floods the CXL link with migration (14.5%). The default 1 GB is the sweet spot (§4.6).

SVA β. Loose (β=0.5) under-suppresses congestion victims (13.2%); tight (β=2.0) over-suppresses hot pages (15.8%). β=1.0 gives a CV = 1 page exactly a 50% score reduction (§4.3).

7 Related Work

Frequency-based OS tiering. *AutoNUMA* [17] migrates to the NUMA node of highest page-fault frequency; *Nimble* [55] improves migration mechanics but leaves the hotness signal unchanged. Both share the signal limitation of §6.2: access frequency cannot distinguish pipeline-critical pages from those whose latency is hidden by prefetching or OOO execution.

Hardware-counter-based tiering. *HeMem* [46] uses Intel PEBS for higher-precision LLC miss tracking; the signal remains access frequency, and it assumes stable interconnect latency that CXL's non-linear bandwidth curve invalidates.

Hardware-managed tiering. Proposals using remapping tables [16] or clustered memory controllers [43] keep migration decisions in hardware. They are orthogonal to *PRISM*: ROB stall counts could serve as the scoring criterion within such a framework.

Beyond-hotness. Several recent systems share *PRISM*'s premise that access frequency is an incomplete signal. SoarAlto [37] formalizes *amortized offcore latency* (AOL) with strong results, but requires offline profiling and heap-object-granularity instrumentation that demands application or toolchain changes. *PACT* [23] samples LLC misses online via Intel PEBS or AMD IBS to find critical pages. However, its MLP estimate is per-tier (TOR-based) rather than per-page, so it cannot distinguish pages whose misses serially block the ROB from those whose latency is covered by MLP within the same tier. Also, its PEBS/IBS dependence limits portability across host CPU platforms (unavailable on ARM). Colloid [53] diagnoses loaded-latency inversion by equalizing per-tier latency, but relies on Intel CHA counters, limiting portability to Intel host CPUs. Tier-level operation cannot identify pages driving stalls (§3.1), and average tier latency lacks an equivalent of *PRISM*'s congestion-victim filter. *PRISM* addresses the full problem: ROB-head stall counts measure non-overlappable latency at page granularity with no offline profiling or device instrumentation. Unlike *NeoMem*, *PRISM* requires no CXL-controller support. Unlike Colloid and *PACT*, *PRISM* does not depend on Intel CHA counters or PEBS/AMD IBS availability; it requires only the proposed CPU-side stall-attribution support, compatible with any standards-compliant `CXL.mem` device.

8 Conclusion

CXL's single shared PCIe link breaks an assumption underlying existing tiering systems: the pages most often accessed are not always the pages most limiting performance. Across ML and graph workloads, the highest-stall and highest-access page rankings diverge substantially—20%–57.5% of the top-stall pages do not appear among the top-access pages—and frequency-based policies fill DDR with the latter, holding pages the OOO core can hide while stall-critical pages remain on the slow tier. Migrating on stall counts rather than access counts is non-trivial on CXL: congestion inflation, bimodal latency, and multi-core amplification each distort a naive stall signal, requiring the CXL-aware mechanisms in *PRISM*. The result is 17.7% geomean improvement over *FreqBased* and 16% over the best prior system across all 13 workloads, with no regression on any workload.

References

[1] 2022. Micron DDR5 SDRAM Product Core Data Sheet. https://static6.arrow.com/aropdfconversion/fc2b144ccf061160504edd742d01cb58fdda91bb/ddr5_sdram_core.pdf.

[2] 2022. SK Hynix Introduces Industry's First CXL-Based Computational Memory Solution (CMS) at the OCP Global Summit. https://news.skhynix.com/sk-hynix-introduces-industrys-first-cxl-based-cms-at-the-ocp-global-summit/.

[3] 2023. CXL Memory Module - Box (CMM-B). https://semiconductor.samsung.com/news-events/tech-blog/cxl-memory-module-box-cmm-b/.

[4] 2023. Micron Memory Expansion Using CXL. https://www.micron.com/products/memory/cxl-memory.

[5] 2023. Tuning Guide for AI on the 4th Generation Intel Xeon Scalable Processors. https://www.intel.com/content/www/us/en/developer/articles/technical/tuning-guide-for-ai-on-the-4th-generation.html.

[6] 2026. The Advantages of Managing the CXL Memory Tier in Hardware. https://www.intel.com/content/www/us/en/content-details/886601/the-advantages-of-managing-the-cxl-memory-tier-in-hardware.html.

[7] Neha Agarwal, David Nellans, Mike O'Connor, Stephen W Keckler, and Thomas F Wenisch. 2015. Unlocking Bandwidth for GPUs in CC-NUMA Systems. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.

[8] Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[9] Yuda An, Shushu Yi, Bo Mao, Qiao Li, Mingzhe Zhang, Ke Zhou, Nong Xiao, Guangyu Sun, Xiaolin Wang, Yingwei Luo, et al. 2024. A Novel Extensible Simulation Framework for Cxl-Enabled Systems. *arXiv preprint arXiv:2411.08312* (2024).

[10] Mohammad Bakhshalipour, Hamidreza Zare, Farid Samandi, Fatemeh Golshan, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. 2024. Blenda: Dynamically-Reconfigurable Stacked DRAM. In *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[11] Scott Beamer, Krste Asanović, and David Patterson. 2015. The GAP Benchmark Suite. *arXiv preprint arXiv:1508.03619* (2015).

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *ACM SIGARCH computer architecture news* (2011).

[13] Arijit Biswas. 2021. Sapphire Rapids. https://hc33.hotchips.org/assets/program/conference/day1/HC2021.C1.4%20Intel%20Arijit.pdf. In *Hot Chips Symposium (HCS)*.

[14] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934* (2020).

[15] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[16] Chia Chen Chou, Aamer Jaleel, and Moinuddin K Qureshi. 2014. CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[17] Jonathan Corbet. 2012. AutoNUMA: The Other Approach to NUMA Scheduling. https://lwn.net/Articles/488709/.

[18] Eduardo HM Cruz, Matthias Diener, Laércio L Pilla, and Philippe OA Navaux. 2016. A Sharing-Aware Memory Management Unit for Online Mapping in Multi-Core Architectures. In *European Conference on Parallel Processing*.

[19] CXL Consortium. 2023. Compute Express Link (CXL) Specification, Revision 3.1. https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-3.1-Specification.pdf. © 2019–2023 Compute Express Link Consortium, Inc. All rights reserved..

[20] Pouya Esmaili-Dokht, Francesco Sgherzi, Valéria Soldera Girelli, Isaac Boixaderas, Mariana Carmin, Alireza Monemi, Adria Armejach, Estanislao Mercadal, Germán Llort, Petar Radojković, et al. 2024. A Mess of Memory System Benchmarking, Simulation and Application Profiling. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[21] Saugata Ghose, Hyodong Lee, and José F. Martínez. 2013. Improving Memory Scheduling Via Processor-Side Load Criticality Information. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*.

[22] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct Access, High-Performance Memory Disaggregation with DirectCXL. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*.

[23] Hamid Hadian, Jinshu Liu, Hanchen Xu, Hansen Idden, and Huaicheng Li. 2026. PACT: A Criticality-First Design for Tiered Memory. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[25] Intel Corporation. 2021. Fact Sheet: Intel Unveils Biggest Architectural Shifts in a Generation for CPUs, GPUs and IPUs. https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-fact-sheet.pdf.

[26] Intel Corporation. 2023. Hardware LLC Prefetch Feature on 4th Gen Intel Xeon Scalable Processor (Sapphire Rapids). https://cdrdv2-public.intel.com/780991/780991-Hardware_LLC_prefetch__4th_Gen_Intel_Xeon_Scalable_Processor-R1_0.pdf.

[27] Intel Corporation. 2024. Intel Memory Latency Checker (MLC). https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html.

[28] JEDEC Solid State Technology Association. 2022. *DDR5 SDRAM Standard, JESD79-5B.* Technical Report. JEDEC.

[29] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[30] Jonghyeon Kim, Wonkyo Choe, and Jeongseob Ahn. 2021. Exploring the Design Space of Page Management for Multi-Tiered Memory Systems. In *Proceedings of the 2021 USENIX Annual Technical Conference (ATC)*.

[31] Min Lee, Vishal Gupta, and Karsten Schwan. 2013. Software-Controlled Transparent Management of Heterogeneous Memory Resources in Virtualized Systems. In *Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*.

[32] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. MEMTIS: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*.

[33] Baptiste Lepers, Vivien Quéma, and Alexandra Fedorova. 2015. Thread and Memory Placement on NUMA Systems: Asymmetry Matters. In *Proceedings of the USENIX Annual Technical Conference (ATC)*.

[34] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[35] Yang Li, Saugata Ghose, Jongmoo Choi, Jin Sun, Hui Wang, and Onur Mutlu. 2017. Utility-Based Hybrid Memory Management. In *Proceedings of the International Conference on Cluster Computing (CLUSTER)*.

[36] Jinshu Liu, Hamid Hadian, Yuyue Wang, Daniel S. Berger, Marie Nguyen, Xun Jian, Sam H. Noh, and Huaicheng Li. 2025. Systematic CXL Memory Characterization and Performance Analysis at Scale. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[37] Jinshu Liu, Hamid Hadian, Hanchen Xu, and Huaicheng Li. 2025. Tiered Memory Management Beyond Hotness. In *Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[38] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[39] Mitesh R. Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H. Loh. 2015. Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-Stacked and Off-Package Memories. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.

[40] ONNX Community. 2023. ONNX Model Zoo. https://github.com/onnx/models.

[41] Biswabandan Panda. 2023. CLIP: Load Criticality Based Data Prefetching for Bandwidth-Constrained Many-Core Systems. In *Proceedings of the 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[42] PCI-SIG. 2019. PCI Express Base Specification, Revision 5.0. PCI-SIG.

[43] Andreas Prodromou, Mitesh Meswani, Nuwan Jayasena, Gabriel Loh, and Dean M Tullsen. 2017. MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-Level Memories. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*.

[44] Amit Puri, Kartheek Bellamkonda, Kailash Narreddy, John Jose, Venkatesh Tamarapalli, and Vijaykrishnan Narayanan. 2024. DRackSim: Simulating CXL-Enabled Large-Scale Disaggregated Memory Systems. In *Proceedings of the ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS)*.

[45] Moinuddin K. Qureshi and Gabe H. Loh. 2012. Fundamental Latency Trade-Off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[46] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. 2021. HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*.

[47] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.

[48] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[49] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations (ICLR)* (2015).

[50] Balaram Sinharoy, JA Van Norstrand, Richard J Eickemeyer, Hung Q Le, Jens Leenstra, Dung Q Nguyen, B Konigsburg, K Ward, MD Brown, José E Moreira, et al. 2015. IBM POWER8 Processor Core Microarchitecture. *IBM Journal of Research and Development* (2015).

[51] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. In *Proceedings of the 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[52] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[53] Midhul Vuppalapati and Rachit Agarwal. 2024. Tiered Memory Management: Access Latency Is the Key!. In *Proceedings of the 30th ACM Symposium on Operating Systems Principles (SOSP)*.

[54] Yanjing Wang, Lizhou Wu, Wentao Hong, Yang Ou, Zicong Wang, Sunfeng Gao, Jie Zhang, Sheng Ma, Dezun Dong, Xingyun Qi, et al. 2025. CXL-DMSim: A Full-System CXL Disaggregated Memory Simulator with Comprehensive Silicon Validation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025).

[55] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble Page Management for Tiered Memory Systems. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[56] Yiwei Yang, Brian Zhao, Yusheng Zheng, Pooneh Safayenikoo, Tanvir Ahmed Khan, and Andi Quinn. 2023. CXLMemSim: A Pure Software Simulated CXL.mem for Performance Characterization. *arXiv preprint arXiv:2303.06153* (2023).

[57] Xiangyao Yu, Christopher J Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas. 2017. Banshee: Bandwidth-Efficient DRAM Caching Via Software/Hardware Cooperation. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*.

[58] Zhe Zhou, Yiqi Chen, Tao Zhang, Yang Wang, Ran Shu, Shuotao Xu, Peng Cheng, Lei Qu, Yongqiang Xiong, Jie Zhang, and Guangyu Sun. 2024. NeoMem: Hardware/Software Co-Design for CXL-Native Memory Tiering. In *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*.