M5: Mastering Page Migration and Memory Management for CXL-based Tiered Memory Systems

Yan Sun
University of Illinois
Urbana, U.S.A.
yans3@illinois.edu

Jongyul Kim
University of Illinois
Urbana, U.S.A.
jyk@illinois.edu

Zeduo Yu
University of Illinois
Urbana, U.S.A.
zeduoyu2@illinois.edu

Jiyuan Zhang
University of Illinois
Urbana, U.S.A.
jiyuanz3@illinois.edu

Siyuan Chai
University of Illinois
Urbana, U.S.A.
siyuanc3@illinois.edu

Michael Jaemin Kim
Seoul National University
Seoul, Republic of Korea
michael604@scale.snu.ac.kr

Hwayong Nam
Seoul National University
Seoul, Republic of Korea
hwayong.nam@scale.snu.ac.kr

Jaehyun Park
Seoul National University
Seoul, Republic of Korea
jhpark@scale.snu.ac.kr

Eojin Na
Seoul National University
Seoul, Republic of Korea
eojin.na@scale.snu.ac.kr

Yifan Yuan
Intel Labs
Hillsboro, U.S.A.
yifan.yuan@intel.com

Ren Wang
Intel Labs
Hillsboro, U.S.A.
ren.wang@intel.com

Jung Ho Ahn
Seoul National University
Seoul, Republic of Korea
gajh@snu.ac.kr

Tianyin Xu
University of Illinois
Urbana, U.S.A.
tyxu@illinois.edu

Nam Sung Kim
University of Illinois
Urbana, U.S.A.
nskim@illinois.edu

Abstract

CXL has emerged as a promising memory interface that can cost-effectively expand the capacity and bandwidth of a memory system, complementing the traditional DDR interface. However, CXL DRAM presents 2–3× longer access latency than DDR DRAM, forming a tiered-memory system that demands an effective and efficient page-migration solution. Although many page-migration solutions have been proposed for past tiered-memory systems, they have achieved limited success. To tackle the challenge of managing tiered-memory systems, this work first presents a CXL-driven profiling solution to precisely and transparently count the number of accesses to every 4KB page and 64B word in CXL DRAM. Second, using the profiling solution, this work uncovers that (1) widely used CPU-driven page-migration solutions often identify warm pages as hot pages, and (2) certain applications have sparse hot pages, where only a small percentage of words in each of these pages are frequently accessed. Besides, this work demonstrates that the performance overhead of identifying hot pages is sometimes high enough to degrade application performance. Lastly, this work presents M5, a platform designed to facilitate the development of effective CXL-driven page-migration solutions, providing hardware-based hot-page and hot-word trackers in the CXL controller. On average, M5 can identify 47% hotter pages and offer 14% higher performance than the best CPU-driven page-migration solution, even with a simple policy.

CCS Concepts: • **Hardware → Memory and dense storage**; **Hardware accelerators**; • **Computer systems organization → Architectures**.

Keywords: Compute Express Link, DRAM, Tiered Memory, Near Memory Processing, Page Migration

ACM Reference Format:
Yan Sun, Jongyul Kim, Zeduo Yu, Jiyuan Zhang, Siyuan Chai, Michael Jaemin Kim, Hwayong Nam, Jaehyun Park, Eojin Na, Yifan Yuan, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2025. 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 (ASPLOS '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3676641.3711999

ASPLOS '25, March 30–April 3, 2025, Rotterdam, Netherlands.

ACM ISBN 979-8-4007-1079-7/25/03
https://doi.org/10.1145/3676641.3711999

1 Introduction

As datacenter applications continue to evolve, they demand DRAM with larger capacity and higher bandwidth. However, it has become more challenging to meet these demands

cost-efficiently, as the current DRAM and its DDR interface technologies have almost reached their scaling limits. Consequently, it has taken 4 and 7 years to double the per-chip capacity of mainstream DRAM (from 8Gb to 16Gb [70]) and the bandwidth of the DDR interface (from 19.2GB/s to 38.4GB/s [64]), respectively. In such a case, the number of memory channels determines both the capacity and bandwidth of a memory system. Yet, a large number of pins is required for each memory channel (*e.g.*, 288 pins for the DDR4/5 interfaces), limiting the number of memory channels in a system under various physical constraints imposed by the CPU package and the PCB board.

Compute Express Link (CXL) [58] built on PCIe has emerged as a promising alternative memory interface, addressing the aforementioned challenges. For example, CXL built on PCIe 5.0 can offer the same bandwidth as DDR5 with 3× fewer pins [62, 66]. Furthermore, the host CPU can access CXL memory with load/store instructions, as the CXL protocol can expose the CXL memory as memory in a remote NUMA node. Nonetheless, a serial interface and the CXL protocol together increase the latency of accessing CXL DRAM by 140–170*ns* compared to DDR DRAM [62]. Therefore, CXL DRAM is considered slow memory, forming a tiered-memory system when used alongside fast memory (*e.g.*, DDR DRAM). To minimize the performance penalty of frequently accessing CXL DRAM, a page-migration solution is essential.

A page-migration solution periodically determines frequently accessed (hot) pages in slow memory and then migrates them to fast memory. It has been extensively studied whenever a new memory architecture and/or technology presenting non-uniform memory access latency has emerged. For instance, a NUMA system experiences a performance penalty whenever a CPU accesses DRAM in a remote NUMA node. To reduce the performance penalty, a page-migration solution, *i.e.*, Automatic NUMA Balancing (ANB), was introduced, exploiting hinting page faults to identify hot pages [5]. Later, other page-migration solutions have been proposed to efficiently use emerging memory, such as Intel 3D XPoint, which presents 3× longer access latency than DDR4 DRAM [31, 48, 53, 68, 69].

These page-migration solutions are based on causing hinting page faults (*e.g.*, TPP [42]), scanning Page-Table Entries (PTEs) (*e.g.*, DAMON [48]), or sampling accessed memory addresses (*e.g.*, Memtis [31]). However, it has been technically challenging to precisely determine whether pages identified by them are truly hot. Furthermore, a large body of work has suggested that a significant percentage of 64B words in each 4KB page are not accessed at all for a certain class of applications (*e.g.*, [2, 8, 52]). We refer to such a page as a sparse page in this work. When sparse pages are migrated, they are first brought into the cache hierarchy, which causes cache pollution. Therefore, it may be less desirable to migrate sparse pages, depending on the characteristics of applications. Nonetheless, the CPU-driven page-migration solutions lack the capability of distinguishing between sparse and dense pages.

The rise of CXL memory has brought renewed attention to the importance of effective page-migration solutions. Unlike the previous memory technologies, including DDR DRAM and 3D XPoint, CXL employs a third-party controller. This offers unprecedented flexibility, facilitating easy integration of hardware functions between the host CPU and CXL memory. Exploiting such flexibility of CXL memory, this work makes the following contributions.

Contribution 1: A CXL-driven hot-page and hot-word profiling solution (§3). To assist with the development of page-migration solutions for CXL-based tiered-memory systems, we present Page Access Counter (PAC) and Word Access Counter (WAC), implemented in an FPGA-based CXL device. They exploit the near-memory processing capability of the CXL controller to count the number of accesses to every 4KB page and 64B word, respectively, in CXL DRAM. As such, they can provide a more precise and transparent profiling capability than other methodologies, such as dynamic binary instrumentation, simulation/emulation, and address sampling.

Contribution 2: CPU-driven page-migration solutions considered harmful (§4). Using PAC, we demonstrate that two representative CPU-driven page-migration solutions (ANB and DAMON) often identify warm pages in popular memory-intensive applications. Specifically, the number of accesses to hot pages identified by ANB and DAMON is only 21% and 29%, respectively, of the number of accesses to the same number of hot pages determined by PAC. Besides, using WAC, we confirm that these page-migration solutions often migrate sparse pages. For instance, we discover that only 16 or fewer words out of 64 words in a page are accessed in 86% of the pages allocated by `Redis` [54]. Meanwhile, we observe that the performance overhead of identifying hot pages and migrating them is high. That is, unless true hot, dense pages are identified and migrated, application performance may degrade instead. This necessitates a page-migration solution that can identify true hot pages and distinguish between dense and sparse hot pages, while minimizing the performance overhead of identifying them.

Contribution 3: A platform for developing CXL-driven page-migration solutions (§5). To address the shortcomings of the CPU-driven page-migration solutions, we advocate for a CXL-driven page-migration solution with the following capabilities. First, it should delegate CPU-intensive operations for identifying hot pages to the CXL controller. Second, it should eschew the migration of sparse pages if necessary. In this work, we provide such capabilities with M5 (Mastering Page Migration and Memory Management for CXL-based Tiered Memory Systems), a platform designed to facilitate the development of such a CXL-driven page-migration solution. M5 consists of the following two key components: (1) Hot-Page Tracker (HPT) and Hot-Word

Tracker (HWT) and (2) M5-manager. HPT and HWT cost-effectively track the top-K hot pages and hot words, respectively. M5-manager provides users with software interfaces for the synergistic use of HPT and HWT, empowering them to develop diverse policies for intelligently identifying pages to migrate. On average, M5 can identify 47% hotter pages and offer 20% and 14% higher performance than ANB and DAMON, respectively, for memory-intensive applications, even with a simple policy.

2 Background

2.1 CPU-driven Page Migration Solutions

To identify hot pages, past page-migration solutions rely on (1) causing hinting page faults [5, 26, 42, 65], (2) scanning PTEs [4, 7, 15, 18, 41, 46, 48, 55, 68], or (3) sampling accessed memory addresses [15, 31, 53]. They are effective for certain applications but are also known to incur a notable performance overhead [32, 42, 46].

Solution 1: Causing hinting page faults. It periodically samples a certain number of pages (*e.g.*, 64K pages) in slow memory, resets the `present` bit in the PTE of each sampled page, and invalidates the corresponding TLB entries across all CPU cores to unmap the corresponding pages. If such pages are accessed later, soft page faults will occur, and the OS will migrate them to fast memory. TPP is the latest page-migration solution that determines hot pages based on such a mechanism. However, it consumes a significant number of CPU cycles for accessing the PTEs of sampled pages, invalidating TLB entries, and handling page faults, especially degrading the performance of latency-sensitive applications such as `Redis` [62].

Solution 2: Scanning PTEs. It periodically scans every valid PTE associated with pages in slow memory. At a given epoch, it checks whether the `access` bit of each PTE is set or not. If it detects that the access bit was set, it increments the access count associated with the page and resets the access bit. Note that the access bit can be set again only if an access to the page incurs a TLB miss later, necessitating the invalidation of the corresponding TLB entry at some point. Unlike Solution 1, it passively invalidates TLB entries, depending on architectural events, such as TLB conflict misses and context switching. To determine whether the page is hot or not, it needs to accumulate the access count over multiple epochs, each consuming a large number of CPU cycles to scan all PTEs. Lastly, since the access bit is a Boolean value, capturing only one access in a given epoch, regardless of the number of accesses to the page, it cannot determine the precise number of accesses to the page.

Solution 3: Sampling accessed memory addresses. It harnesses the modern CPU's capability to sample architectural events, *e.g.*, Processor Event-Based Sampling (PEBS) in Intel CPUs [20] to determine hot pages [15, 31, 53]. Specifically, it samples LLC miss addresses at a specified frequency (*e.g.*, once every 1,000 LLC misses) and stores them in the PEBS buffer. When the PEBS buffer is full, an interrupt is triggered, and the sampled addresses in the PEBS buffer are processed and analyzed by the CPU to identify hot pages. It may determine the number of accesses to pages more precisely than Solutions 1 and 2. Yet, it may lack preciseness in identifying hot pages because it does not capture all the addresses. Furthermore, it incurs a higher performance overhead to more precisely identify hot pages, as a higher sampling frequency interrupts the CPU more frequently to process the sampled addresses in the PEBS buffer. Lastly, at the moment, the PEBS on the Intel CPU does not support sampling LLC misses to CXL memory [67].

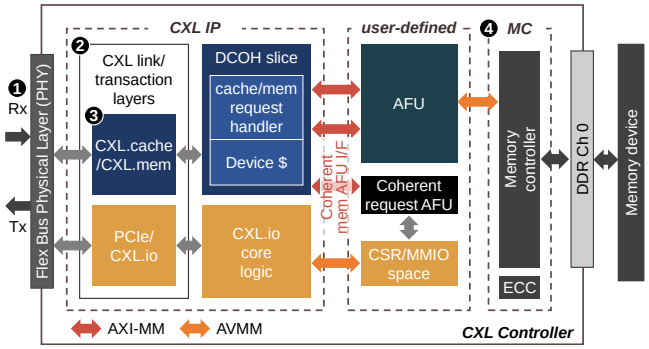

Figure 1. Overview of a CXL controller architecture implemented in Intel Agilex FPGA devices.

2.2 CXL Memory

CXL device types. Built on the physical layer of PCIe, CXL defines three device types: CXL Types 1, 2, and 3, each with a distinct composition of three protocols: CXL.io, CXL.cache, and CXL.mem. CXL.io is the common protocol for all three types of CXL devices to initialize the interface between a host CPU and a CXL device [14]. CXL.cache facilitates device cache and cache-coherent Device-to-Host (D2H) memory accesses for CXL Type-1 and Type-2 devices. The device can access the entire host memory space in a cache-coherent manner. CXL.mem supports device memory and Host-to-Device (H2D) and Device-to-Device (D2D) memory accesses for CXL Type-2 and Type-3 devices. Since the host CPU can access the entire device memory space, we can use both CXL Type-2 and Type-3 devices as memory capacity/bandwidth expanders that serve as slow memory in a tiered-memory system.

CXL controller architecture. Figure 1 depicts a CXL controller architecture implemented in Intel Agilex FPGA devices. It consists of ❶ PCIe physical layer, ❷ CXL link layer, ❸ CXL transaction layer, and ❹ memory controller (MC) IPs. ❷ and ❸ implement the CXL.cache and/or CXL.mem protocol functions, depending on a desired device type. Between ❸ and ❹, we may implement diverse near-memory hardware functions in the Accelerated Function Unit (AFU) and

the Coherent request AFU (CAFU). The AFU can snoop and manipulate not only addresses from the host CPU to device memory but also data from the host CPU or the MC. Additionally, it can independently issue non-cache-coherent D2D memory access requests to the device memory. CXL.cache accompanies one or more instances of the Device Coherence Engine (DCOH), each including device cache and serving D2H and H2D memory accesses from the CAFU and the host CPU, respectively, in a cache-coherent manner. ❷ and ❸ are collectively referred to as CXL IP in this work.

3 CXL-driven Page and Word Access Counting

Limitations of current approaches. It has been challenging to precisely determine the number of accesses to every page and word in DRAM with commodity systems. For example, Intel PEBS can sample only one address out of a hundred at most. A dynamic binary instrumentation using Intel Pin [24] can capture every memory access address, but it requires notable effort to precisely determine DRAM access addresses, especially when other applications co-run and cause interference at L2 cache and LLC. As such, it has been elusive to assess whether pages identified by past page-migration solutions are truly hot pages and how many words in such pages are accessed or not in commodity systems. Meanwhile, the emergence of a commodity FPGA-based CXL device has opened up an unprecedented opportunity to tackle such a challenge. In this section, we first describe page and word access counting functions (PAC and WAC) in the CXL controller of such a device. PAC and WAC exploit the inherent capability of near-memory processing in a CXL controller to precisely count the number of accesses to every page and word in CXL memory. Subsequently, we describe software support to use PAC and WAC as a complete profiling solution.

Hardware. In this section, we assume that a system supports a 48-bit Physical Address (PA) space, and the OS manages memory in 4KB page granularity, except when specifically allocating 2MB or 1GB huge pages. Then DRAM is accessed with PA[47:6] because the access granularity of DRAM is 64B (*i.e.*, the size of a cache line), and the Page Frame Number (PFN) of a 4KB page is identified by PA[47:12]. Figure 2 depicts the hardware architecture of PAC and WAC based on these assumptions. The PAC hardware consists of three key components. ❶ An address-to-PFN converter snoops every memory access address (PA[47:6]) from the CXL IP (Figure 1-❸) to the MCs (Figure 1-❹), and right-shifts the address by 6 bits to obtain the PFN. ❷ An SRAM unit provides a capacity of 4MB, with each entry storing an L-bit access count for a 4KB page. ❸ Miscellaneous logic comprises an L-bit adder, a controller, and configuration/control registers. Lastly, WAC has the same hardware architecture as PAC, but it does not convert a given address to a PFN. Note that PAC and WAC

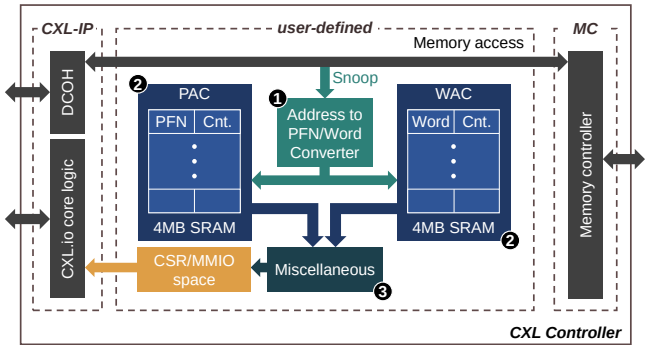

Figure 2. Page Access Counter (PAC) and Word Access Counter (WAC) in a CXL controller.

fundamentally differ from event sampling-based approaches such as PEBS since PAC and WAC track every DRAM access address.

The SRAM unit is indexed by the PFN from the address-to-PFN converter. For a given PFN, the controller gets the corresponding access count from the SRAM unit, increments it using the adder, and then stores it back to the SRAM unit. An overflow may happen when L is small. However, for example, a 16-bit access count saturates only after ~20s, even for memory-intensive applications in our experiments. Nonetheless, PAC may reset saturated counters after accumulating them into the corresponding 64-bit counters stored in the `access-count table` allocated in a host or device memory region using D2H or D2D memory accesses. After completing the execution of a given workload, the host CPU can obtain the precise number of accesses to each page from the access-count table. The SRAM unit is exposed to the host CPU as an MMIO region, allowing PAC software to directly access the access counts.

Software. The software interface for PAC and WAC first maps the SRAM units and configuration/control registers to an MMIO region. Then, it can access the access counts through the MMIO interface over CXL.io. However, since the size of an MMIO region is limited to 2MB, we use a 1MB region to access the access counts in the SRAM unit and the remaining 1MB region to access configuration and control registers. To access 4MB of the access counts with a 1MB region, we design PAC and WAC hardware functions to use a configuration register to store a base address for each 1MB region of the SRAM unit and access the access counts with the base address plus an offset given by PAC and WAC software interfaces. Therefore, by changing the base address in the configuration register through the MMIO interface, it can access all access counts.

Scalability. When the SRAM unit in the CXL controller is not large enough to store all access counts for a given CXL DRAM capacity, we may take one of the following approaches. First, we adapt the SRAM unit as a cache to store a subset of counters. When a cache miss occurs, the

controller chooses a counter to evict, writes the count to the corresponding counter in the access-count table using a D2H or D2D memory access, and then writes 1 to the counter in the SRAM unit. Second, we may limit the size of the CXL memory region that PAC or WAC monitors at a time by setting a configuration register that specifies the memory address range. Then, we monitor either (1) all CXL memory regions over multiple intervals during a single run or (2) only one CXL memory region during a single run and repeat it for different CXL memory regions over multiple runs. In this work, WAC monitor a 128MB memory region at a time, where each word address maps to a 4-bit counter in the SRAM array.

4 Usefulness and Cost of CPU-driven Page Migration Solutions

In this section, we first use PAC to evaluate how precisely CPU-driven page-migration solutions, ANB and DAMON determine true hot pages. We choose the latest version of ANB over TPP because TPP has some known problems [63] that we have also experienced, whereas ANB has improved over the years. Also, we do not evaluate Memtis, a representative sampling-based page-migration solution, because the Intel CPU lacks PEBS support for CXL devices and, consequently, cannot correctly run Memtis at the moment [67]. Second, we assess how many words in those pages are accessed, using WAC. Lastly, we evaluate the performance cost of identifying hot pages. Note that these analyses are not intended to provide comprehensive evaluations of the effectiveness of these solutions across a wide range of memory-intensive applications. Instead, they are meant to demonstrate the utility of PAC and WAC as a profiling solution for developing and optimizing page-migration solutions.

4.1 Usefulness of Identified Hot Pages

In this section, we obtain the access counts of K hot pages identified by ANB and DAMON by looking up PAC's access-count table with the PFNs of these pages. We then compare these access counts to those of the top-K hot pages determined by PAC. Lastly, we evaluate the access sparsity of pages using WAC.

Access count. Using PAC, we first evaluate the hotness of pages determined by ANB and DAMON with the following steps. (S1) We modify ANB and DAMON to store the PFNs of identified hot pages into a *hot-page list* but not to migrate these pages. (S2) We execute a given benchmark with a page-migration solution after using Linux `cgroup` to allocate all the pages of the benchmark to CXL DRAM. This makes every DRAM access from the benchmark served by CXL DRAM. (S3) During the execution, the page-migration solution stores the PFNs of identified hot pages in the hot-page list, while PAC records the number of accesses to every page in the access-count table, respectively. (S4) After the execution, we take a PFN from the hot-page list, determine the number of accesses to the page corresponding to the PFN by looking up the access-count table with the PFN, accumulate it into `k_access_count`, and repeat this for every PFN in the hot-page list. (S5) We also find the top-K hot pages from the access-count table, determine the accumulated number of accesses to these pages, and store it in `top_k_access_count`. Then, we divide `k_access_count` by `top_k_access_count` to get an *average access-count ratio*, which serves as a metric to evaluate how precisely a given page-migration solution identifies true hot pages.

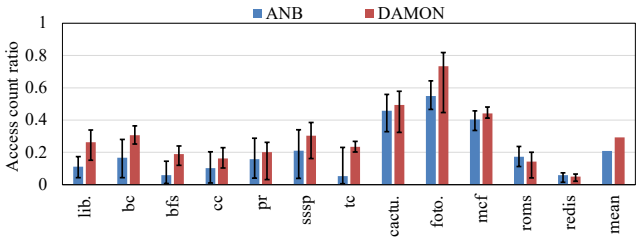

Figure 3. Average access-count ratio of hot pages identified by ANB and DAMON.

Figure 3 plots the average access-count ratio of hot pages identified by ANB and DAMON. For this experiment, we run twelve popular memory-intensive benchmarks. See §6 for detailed information on the setup and these benchmarks. We set K to be up to 128K pages (*i.e.*, 512MB) corresponding to roughly 1/16 of the memory footprint of all benchmarks above. We collect up to 128K hot pages identified by these solutions and record the number of accesses to every page. We repeat the measurement above at 10 different random execution points. A vertical line across a bar in Figure 3 represents the minimum and maximum values of the average access-count ratio across 10 execution points. This is to evaluate the preciseness of the hot pages determined by these solutions at various execution phases. Except for `cactuBSSN_r`, `fotonik3d_r`, and `mcf_r`, we find that both solutions give the access-count ratio below 0.4 on average. That is, the hot pages identified by ANB and DAMON are not as hot as the top-K hot pages. Overall, DAMON offers higher average access-count ratios than ANB.

Observation 1: CPU-driven page-migration solutions often identify warm pages as hot pages.

Access sparsity. The page-migration solutions migrate an entire page from slow memory to fast memory, regardless of whether only a few or all words in the page are hot. Meanwhile, only a small number of words in a page can be frequently accessed. In other words, such a sparse page can be identified as a hot page because of a few very hot words. However, when the host CPU migrates such sparse pages, it not only wastes the capacity of fast memory but also pollutes the cache hierarchy. Therefore, if there coexist sparse and

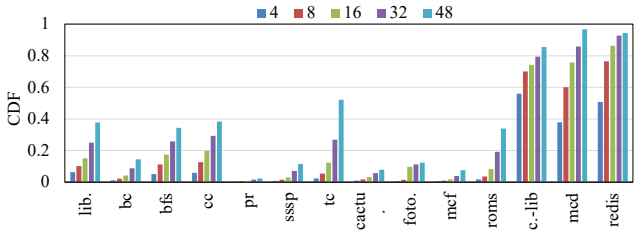

Figure 4. Probability of a 4K page where at most N unique 64B words are accessed. `mcd` and `c.-lib` are abbreviations for `Memcached` and `CacheLib`, respectively.

dense hot pages, migrating dense hot pages can be more beneficial than migrating sparse hot pages. However, the current page-migration solutions lack the ability to distinguish between sparse and dense hot pages. This inspires us to leverage WAC and investigate the access sparsity of pages in memory-intensive applications.

Figure 4 plots the probability of a 4K page where at most 4, 8, 16, 32, and 48 unique 64B words (6.25%, 12.5%, 25%, 50%, and 75% of words in a page) are accessed. This shows that the likelihood of a page having 25% or fewer of its unique words accessed is 86%, 76%, and 74% for `Redis`, `Memcached`, and `CacheLib`, respectively. That is, most pages in these benchmarks are sparsely accessed. In contrast, the pages in the SPEC CPU 2017 benchmarks, except for `roms_r`, are densely accessed, as the probability of a page with at least 75% of its words accessed is 87% to 92%. The `GAP` benchmark suite exhibits greater variance across benchmarks in the probability of sparse pages. The pages in `PageRank` and `SSSP` are mostly densely accessed, with the probability of a page having at least 75% of its words accessed being 98% and 89%, respectively. In contrast, `Liblinear`, `BC`, `BFS`, `CC`, and `TC` present notable access sparsity, with the probability of a page with at most 25% of words accessed being 15%, 4%, 17%, 20%, and 12%, respectively.

Observation 2: A large percentage of pages in certain applications can be sparsely accessed.

4.2 Performance Cost of Identifying Hot Pages

To identify hot pages, CPU-driven page-migration solutions perform expensive operations (§2.1). When such expensive operations are performed periodically and frequently, they can interfere with application execution, incurring a notable performance penalty, especially for latency-sensitive applications. In this section, to quantify the performance overhead of identifying hot pages, we pin all Linux kernel processes, including the page-migration process, to a CPU core. Next, we measure the CPU cycles consumed by the Linux kernel processes after disabling only `migrate_pages()`, to evaluate the performance overhead of identifying hot pages alone.

Our experiment, based on the methodology described above, shows that ANB and DAMON increase the number of CPU cycles consumed by the Linux kernel by up to 487% and 733% (with an average of 159% and 277% across the benchmarks), respectively. This is because scanning a large number of PTEs (DAMON), invalidating TLBs (ANB), and handling soft page faults (ANB) consumes a large number of CPU cycles (§2.1). This significant increase in CPU cycles consumed by ANB and DAMON can disturb the execution of applications running on the same CPU core. For example, ANB and DAMON increase the 99th-percentile (p99) latency of `Redis`, a representative latency-sensitive application, by 34% and 39%, respectively. The increase in execution time of best-effort applications is also notable. For instance, ANB and DAMON increase the execution time by up to 4.6% (`SSSP`) and 8.6% (`Liblinear`), respectively. Note that recent work has reported that even sampling-based page migration solutions can also considerably degrade application performance when the sampling rate is high to achieve high precision in identifying hot pages (*e.g.*, more than 15% when sampling one every 100 LLC miss addresses [75]).

Observation 3: The performance overhead of identifying hot pages is notable enough to potentially degrade application performance if the benefit do not compensate for the overhead.

5 M5: Track, Filter, and Migrate

In this section, to facilitate the development of effective CXL-driven page-migration solutions, we present M5, consisting of two components: (1) Hot Page Tracker (HPT) and Hot Word Tracker (HWT) and (2) M5-manager. HPT and HWT, inspired by hardware-based Row-Hammer (RH) defense solutions, cost-efficiently track the top-K hot pages and words, respectively. HPT and HWT can eliminate the high performance overhead of identifying hot pages and introduce the ability to determine sparse hot words, respectively, which the CPU-driven page-migration solutions lack. M5-manager provides software interfaces to HPT and HWT, allowing users to explore diverse policies for their CXL-driven page-migration solutions.

5.1 HPT and HWT

Need for cost-efficient top-K hot page and word trackers. Page Access Count (PAC) and Word Access Count (WAC) are useful offline profiling mechanisms, but they have two limitations when used as online mechanisms to determine hot pages and words, respectively. First, they impose a high storage cost. For example, PAC and WAC demand 128MB and 8GB of memory space, respectively, to store the number of accesses to every page and word for 256GB of CXL DRAM. Second, they require a significant amount of time to determine the top-K hot pages and words because either software or hardware must fetch all access counts and then sort them. For instance, it takes hundreds of milliseconds for a CPU core to access 2M 16-bit access counts for 8GB of

memory. This prevents us from making agile decisions on which pages to migrate.

Algorithm and requirement. To minimize the storage and performance overheads of determining hot pages and words without significantly compromising preciseness, we may exploit one of the top-K estimation algorithms for HPT and HWT. Among such algorithms, streaming algorithms [45] have been widely studied and adopted in various fields, from networking applications [16] to Row-Hammer defense solutions [27, 40, 49]. Specifically, the streaming algorithms can be categorized into counter-based, sketch-based, and sampling-based algorithms, represented by Space-Saving [43], CountMin-Sketch (CM-Sketch) [12], and Sticky-Sampling [38], respectively. After analyzing the preciseness, scalability, and hardware cost of these algorithms, we choose CountMin-Sketch (CM-Sketch) and compare it with a variant of Space-Saving designed for a Row-Hammer defense solution, Mithril [27], in this work.

To make HPT and HWT useful and practical, we determine the following requirements. First, they should be able to handle each memory access every 2.5ns (*i.e.*, tCCD or the highest memory access rate of DDR4 3200 DRAM). That is, they must operate at least at 400MHz. Second, when the host CPU queries the top-K hot addresses, it should be able to return them quickly. Lastly, the hardware cost should be reasonable, although it is a secondary constraint.

Architecture and operations. Both HPT and HWT share the same architecture and operations, except that they use page and word addresses, similar to PAC and WAC, respectively. In this section, we refer to both HPT and HWT as the top-K tracker. A top-K tracker comprises (1) an SRAM-based CM-Sketch unit and (2) a sorted Content Addressable Memory (CAM) unit, which keep track of the (estimated) access count of each accessed address and the top-K hot addresses, respectively. Figure 5 illustrates the architecture and operations of the top-K tracker. First, CM-Sketch comprises an SRAM array with H rows and W columns of entries, designed to estimate the number of accesses to a given address. Each entry stores an access count. For a given memory access, the address is applied to H hash functions in parallel (❶). Each hash function is associated with a row, and one of the W entries in the row, indexed by the hash function's output for the address, is incremented by one (❷). Among the H incremented access counts across the rows, the minimum value is determined by a comparator tree (❸) and becomes the estimated access count of the address.

Second, the sorted CAM unit comprises K entries, each storing a pair of an address (tag) and an access count (value), sorted based on the access counts. When a memory address is provided, it is checked against the sorted CAM unit. For a hit, the `access count` field of the matched CAM entry is updated with the access count from the CM-Sketch unit (❹). For a miss, the access count is compared with the minimum access count from the sorted CAM unit (❺). If the access count is

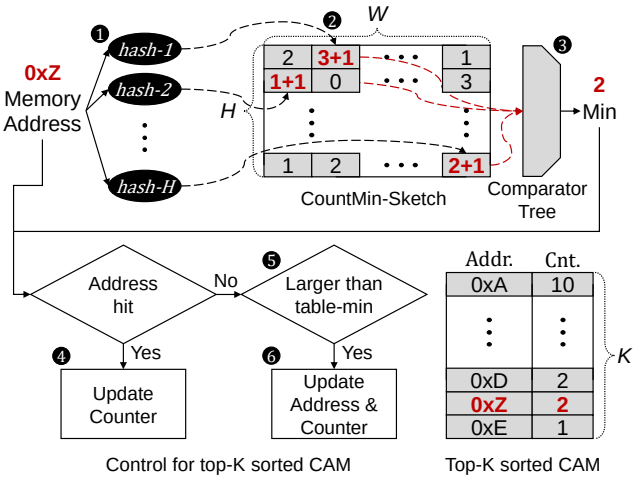

Figure 5. A CM-Sketch top-K tracker.

larger than the minimum access count, the entry storing the minimum access count is updated with the address and the access count from the CM-Sketch unit (❻). In this way, the sorted CAM unit can keep track of the top-K hot addresses and quickly report them back to M5-manager (§5.2), which runs on the host CPU and places a query. Both the CM-Sketch unit and the sorted CAM unit can be reset immediately after the query is served to track a fresh set of the top-K hot addresses for the next epoch.

5.2 Manager

M5-manager consists of four components: ❶ `Monitor`, ❷ `Nominator`, ❸ `Elector`, and ❹ `Promoter`, providing a collection of interfaces and a template implemented in software (Figure 6). `Monitor` interfaces with performance counters in the host CPU, which provide useful statistics for users to determine the frequency of page migration and assess whether the previous page migration was effective or not. `Nominator` not only interfaces with HPT and HWT to collect hot page and word addresses, respectively, but also processes these addresses to help users identify more useful hot pages. `Elector` offers a template that incorporates `Monitor` and `Nominator`, allowing users to easily design their hot-page migration policies. `Promoter` provides in-kernel components that interface between `Elector` and the Linux kernel to launch the migration of pages identified by `Nominator` to DDR DRAM. Lastly, M5 depends on a capability of the latest Linux kernel, Multi-Generation LRU (MGLRU) [72], to choose pages to demote to DDR DRAM. It has been proven to be robust, precise, and cost-effective in past work [25, 31, 44, 71].

The underlying implementation of M5-manager resembles ANB, which periodically migrates pages. However, M5-manger differs from ANB in how migrations are triggered and how responsive the migration is. Besides, the components of M5-manager are all user-space programs except

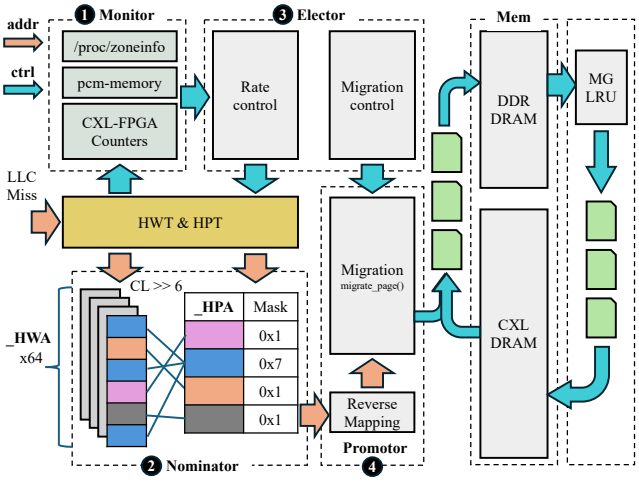

Figure 6. An overview of the M5 manager.

for `Promoter`, whereas those of ANB are all kernel-space programs. Such a design choice facilitates better software compatibility with the fast-evolving kernel, as it minimizes and isolates the code that depends on a specific kernel. It also helps M5-manager seamlessly integrate with other kernel memory management functions, such as `cgroup`. The remainder of this section describes the four M5-manager components, along with guidelines derived from our extensive profiling of various benchmarks to help users design effective page-migration policies.

❶ **Monitor.** HPT can precisely and transparently identify hot pages in CXL DRAM. Nonetheless, such a capability alone is not sufficient to design an effective page-migration solution, especially when the capacity of DDR DRAM allocated to a given application is constrained. Specifically, since HPT cannot discern whether hot pages in CXL DRAM are hotter than those in DDR DRAM, blindly migrating hot pages identified by HPT to DDR DRAM may end up demoting even hotter pages to CXL DRAM, thereby degrading application performance. This demands metrics that help a page-migration policy determine whether to migrate pages and how aggressively they should be migrated to DDR DRAM. To address such needs, M5-manager provides `Monitor`, which captures various utilization statistics of a given tiered-memory system through three functions described in Table 1: `nr_pages()`, `bw()`, and `bw_den()`. Note that `Monitor` reports only read bandwidth because write accesses to both DDR DRAM and CXL DRAM occur only when they miss the LLC, which first incurs read accesses to them under the write-allocate policy. Below we discuss why these statistics can serve as useful metrics.

Suppose that `nr_pages(DDR)` and `nr_pages(CXL)` pages are randomly allocated to DDR DRAM and CXL DRAM, respectively. When `nr_pages(DDR)` + `nr_pages(CXL)` is large enough, we hypothesize that `bw(DDR)` and `bw(CXL)` should be proportional to `nr_pages(DDR)` and `nr_pages(CXL)`, respectively. To validate this hypothesis, we run `mcf_r` with the `nr_pages(DDR)/nr_pages(CXL)` ratios of 2, 1, and 1/2 and observe that the `bw(DDR)/bw(CXL)` ratios are 2.02, 0.919, and 0.571, respectively. From this, we can first infer that `bw(DDR)` should increase and `bw(CXL)` should decrease, as hotter pages are migrated to DDR DRAM. Second, we recognize that the bandwidth density, `bw_den(node)` (= `bw(node)` / `nr_pages(node)`), is also a useful metric to assess the density of hot pages. For instance, if `bw_den(CXL)` is higher than `bw_den(DDR)`, CXL DRAM may store more hot pages per allocated capacity than DDR DRAM. Lastly, we note that application performance is proportional to the total consumed bandwidth (= `bw(DDR)` + `bw(CXL)`) for a given execution phase.

> **Guideline 1:** If `bw_den(CXL)` is higher than `bw_den(DDR)`, hot pages should be migrated from CXL DRAM to DDR DRAM as soon and aggressively as possible.
> **Guideline 2:** Hot pages can continue to be migrated to DDR DRAM as long as `bw(DDR)` keeps increasing, even if `bw_den(DDR)` exceeds `bw_den(CXL)`.

❷ **Nominator.** Based on the statistics from `Monitor`, M5-manager may decide to migrate hot pages to DDR DRAM. It may simply access HPT to get hot-page addresses and then migrate these hot pages, also referred to as HPT-only `Nominator`. However, as we discussed earlier (§4.1), we might prefer to migrate dense hot pages over sparse ones, especially when the hotness (*i.e.*, access count) of pages is similar. To identify denser hot pages, M5-manager provides `Nominator`, which can also leverage hot-word addresses from HWT. It comprises two data structures, `_HPA` and `_HWA` (Figure 6), which store hot-page addresses and hot-word addresses from HPT and HWT, respectively. Each entry of `_HPA` also stores a 64-bit mask, where each bit is set by the hot-word addresses from `_HWA` that map to the PFN stored in the entry. Whenever invoked, `Nominator` accesses `_HPA` and `_HWA`, which are periodically updated by HPT and HWT using the D2H memory access capability of CXL Type-2 devices (§2.2), to get hot-page and hot-word addresses, respectively.

With `_HPA` and `_HWA`, it offers two mechanisms that facilitate a given hot-page migration policy to identify denser hot pages. The first mechanism, dubbed HPT-driven `Nominator`, searches `_HPA` updated by HPT, using the PFNs derived from the hot-word addresses from `_HWA`. Upon finding a matching PFN in `_HPA`, it sets one bit of the 64-bit mask, indexed by the relative word address in the page. Depending on how many

Table 1. Monitor functions.

Function	Description	Tool
`nr_pages(node)`	Number of pages allocated to node	`pcp-zoneinfo`
`bw(node)`	Consumed read bandwidth of node	`pcm`
`bw_den(node)`	`bw(node)` per allocated capacity	

mask bits are set in a hot page, the page-migration policy can determine whether the hot page is sparse or dense. The second mechanism, named HWT-driven `Nominator`, starts with an empty list of `_HPA` and uses only hot-word addresses from `_HWA` to construct `_HPA`. As HWT-driven `Nominator` does, it searches `_HPA` using the page addresses converted from hot-word addresses. If it does not find any match, it adds the page address to the PFN field of an `_HPA` entry and sets the 64-bit mask, which serves as an access count, to one. If it finds a match, it increments the access count stored in the associated 64-bit mask.

Guideline 3: HPT-driven `Nominator` can be useful for applications with a mix of dense and sparse hot pages, such as `roms` and `liblinear`.
Guideline 4: HWT-driven `Nominator` can be useful for applications with only sparse hot pages, such as `Redis` and `Cachelib`.

❸ **Elector.** Based on the statistics obtained by `Monitor` and the hot pages identified by `Nominator`, `Elector` determines the frequency of page migration and whether to migrate hot pages to DDR DRAM. Algorithm 1 provides a sample `Elector` implementation where users can simply statically or dynamically adapt tunable parameters. First, following Guideline 1, we introduce a function, `fscale()`, which scales the default frequency, `f_default`, for page migration proportional to `bw_den(CXL)/bw_den(DDR)` (Algorithm 1:line-2). `f_default` is a tunable parameter, and `fscale()` can employ a monotonically increasing linear or non-linear function, such as $y = x^n$ or $y = n \times exp^x$, where n is a tunable parameter. Next, we present two more functions, `bw_tot` and `rel_bw_den()`, *i.e.*, `bw(DDR) + bw(CXL)` and `bw_den()/bw_tot`, respectively. If `rel_bw_den(DDR)` increases compared to the previous period, the previously migrated pages might have contributed to increasing the bandwidth consumption of DDR DRAM, *i.e.*, application performance, based on Guideline 2 (Algorithm 1:line-4—6). Thus, `Elector` invokes `Promoter` to migrate hot pages nominated by `Nominator` to DDR DRAM. Otherwise, `Elector` stops migrating pages at the current period, as the migrated pages have decreased the bandwidth consumption of DDR DRAM. We normalize `bw_den()` to `bw_tot` to account for changes in `bw_den()` caused by the execution-phase changes that increase or decrease the intensity of memory access over time.

❹ **Promoter.** It provides an interface between `Elector` and the Linux kernel to migrate hot pages decided by `Nominator`. Specifically, upon receiving the hot-page addresses from `Elector`, it updates a Linux kernel `proc` file, an in-kernel component of `Promoter`, with these addresses. When the `proc` file is updated, another in-kernel component first checks whether the given pages can be safely migrated to DDR DRAM, as certain pages may be pinned to specific nodes or physical addresses for various reasons. For example, `Promoter` will reject the migration of a page if the user explicitly requests the page to be allocated on a CXL device or if the page is pinned for Direct Memory Access (DMA). Subsequently, when it determines that migrating the pages to DDR DRAM is safe, it invokes `migrate_pages()`.

Algorithm 1: `Elector`

```
1  while (true) {
2      T = 1 / (fscale(bw_den(CXL) / bw_den(DDR)) * f_default);
3
4      bw_tot = bw(DDR) + bw(CXL);
5      rel_bw_den(DDR) = bw_den(DDR) / bw_tot;
6      if (rel_bw_den(DDR) - prev_rel_bw_den(DDR) > 0) {
7          Promoter(Nominator());
8      }
9      prev_bw_tot = bw_tot;
10     prev_rel_bw_den(DDR) = rel_bw_den(DDR);
11
12     sleep(T);
13 }
```

6 Evaluation Setup

System. We use a dual-socket server based on the Intel 4th-generation Xeon Scalable Processor. We connect the server to the latest Intel Agilex-7 I-Series development kit [22] (or simply Agilex7 henceforth), which serves as a CXL development platform in this work. Agilex7 is integrated with a hardened CXL ×16 endpoint IP and a DDR4-2666 DRAM controller connected to 8GB of on-board DRAM. We use the same number of CPU cores as the number of benchmark instances or threads and disable the remaining CPU cores using the Linux CPU hotplug feature [59]. This is to capture the interference incurred by the page-migration processes by running the page-migration processes and one of the instances or threads of a given benchmark on the same CPU core. Since we use only a subset of CPU cores, we scale the LLC capacity proportional to the number of used CPU cores using Intel Cache Allocation Technology (CAT) [23]. See Table 2 for a detailed description of our system.
Benchmark. We evaluate twelve memory-intensive benchmarks. Specifically, we choose the four most memory-intensive benchmarks from the SPECrate CPU 2017 benchmark suite [61]: `mcf_r`, `cactuBSSN_r`, `fotonik3d_r`, and `roms_r`, after running all the benchmarks on our server and measuring LLC Misses Per Kilo Instructions (MPKI). We run eight instances of each SPECrate benchmark, which collectively have a memory footprint close to the capacity of our 8GB CXL DRAM,

Table 2. System configuration of a dual-socket server.

CPU	2× Intel® Xeon 6430 CPUs @2.1 GHz [21], 32 cores and 60 MB LLC per CPU, Hyper-Threading disabled
Memory	Socket 0: 4× DDR5-4800 channels Socket 1: 4× DDR5-4800 channels

Table 3. Evaluated benchmarks

Benchmark	Description	Footprint	# cores/ways
Liblinear	Linear classification, tested with KDD2012 dataset	6.0GB	20/10
BC	Betweenness Centrality	6.9GB	20/10
BFS	Breadth-First Search	6.9GB	20/10
CC	Connected Components	6.9GB	20/10
PR	PageRank	6.9GB	20/10
SSSP	Single-Source Shortest Paths	6.9GB	20/10
TC	Triangle Counting	5.0GB	20/10
cactuBSSN_r	Einstein's equations simulation	6.3GB	8/4
fotonik3d_r	Photonic waveguide simulation	6.8GB	8/4
mcf_r	Single-depot vehicle scheduling	4.9GB	8/4
roms_r	Free-surface ocean model simulation	6.7GB	8/4
Redis	In-memory KVS with YCSB-A	6.0GB	1/1

except for mcf_r. We run one instance of each of the following benchmarks: an in-memory database (Redis [54] version 6.0.16), a machine learning application (Liblinear [36] version 2.47), and six graph processing benchmarks (GAP [6]): BFS, SSSP, PR, CC, BC, and TC, all using their original configurations. As inputs, we use YCSB-A [11] (Redis), KDD 2012 [35] (Liblinear), Twitter [30] (BFS, CC, TC, and PR with undirected graph), and Google [1] (BC and SSSP with directed graph). We adjust the size of these input datasets to have a memory footprint close to 8GB. We limit the capacity of DDR DRAM that a given benchmark can use to 3GB, ensuring that roughly 50% of the pages can be migrated to DDR DRAM.

Page-migration solutions. We use ANB and DAMON as CPU-driven page migration solutions. For ANB, we use Linux version 5.19, as we assess it provides the most stable page-migration performance; in some other Linux versions, ANB fails to migrate any pages for certain benchmarks we evaluate. For DAMON we use Linux kernel version 6.11, which has officially integrated it as one of the page-migration solutions. M5 is implemented in Linux kernel version 6.5. A specific kernel version may affect application performance. However, we decided not to port DAMON and ANB to the same kernel version as M5 because of the following reasons. First, we do not observe an obvious application performance trend across different Linux kernel versions. Second, one of the key evaluation points is the access-count ratios, which are not affected by a specific kernel version.

7 Evaluation

In this section, we first explore the design space of CM-Sketch and Space-Saving top-K trackers, each implemented with both Agilex7 and a 7nm logic technology [10]. Second, we compare the average access-count ratio and application performance among ANB, DAMON, and M5.

7.1 Design Space Exploration of Top-K Trackers

To explore the design space of HPT and HWT, we assess the average access-count ratio, size, and power of a CM-Sketch top-K tracker while sweeping its key design parameter, *i.e.*, the number of access counts (= $H \times W$). We also evaluate those of a Space-Saving top-K tracker while varying N, which is equivalent to $H \times W$, as both represent the number of access counts. We use N for both hereafter. The Space-Saving top-K tracker may comprise a sorted CAM unit with N entries, each storing an address (tag) and an access count (value). The CAM unit is the same as the one used for the CM-Sketch top-K tracker, but it requires N CAM entries to store and track the access counts of N addresses, where N should be much larger than K. In contrast, the CM-Sketch top-K tracker stores the access counts of N addresses in the SRAM unit, while tracking the top-K access counts in the CAM unit, decoupling the access counts from the sorted CAM unit. The preciseness of both the Space-Saving and CM-Sketch top-K trackers increases with N, but the CAM complexity of the Space-Saving top-K tracker limits N.

Size and power under timing constraint. Before assessing the average access-count ratio of top-K trackers, we synthesize both the Space-Saving and CM-Sketch top-K trackers for various N and fixed K (= 5) with both Agilex7 and the 7nm logic technology. This is done to determine the maximum N that meets the 400MHz timing constraint for both the top-K trackers. Although our platform is based on FPGA, we also evaluate ASIC-based top-K trackers, because they will be used in industry CXL DRAM products. The Space-Saving and CM-Sketch top-K trackers employ M parallel CAM and SRAM blocks (or banks), respectively, as design optimization techniques to maximize N while meeting the 400MHz timing constraint. For each memory address, the Space-Saving top-K tracker must access all the CAM blocks in parallel, whereas the CM-Sketch top-K tracking accesses only one SRAM block, which allows pipelined accesses to different SRAM blocks for consecutive memory addresses.

The FPGA synthesis report shows that the Space-Saving top-K tracker can have only up to 50 CAM entries, whereas the CM-Sketch top-K tracker can have up to 128K SRAM entries. This large difference in N between the two top-K trackers arises from the fact that CM-Sketch uses a K-entry CAM unit regardless of N, whereas Space-Saving uses an N-entry CAM unit. Table 4 lists the size and power consumption of the ASIC-based Space-Saving and CM-Sketch top-K trackers, with H fixed at 4. In our evaluation, we sweep H from 2 to 16 and see only a secondary effect on both the average access-count ratio and the timing. Even the ASIC-based Space-Saving top-K tracker can have at most N = 2K, providing almost an order of magnitude fewer entries than the FPGA-based CM-Sketch top-K tracker. For the same number of entries (*e.g.*, N = 2K), the Space-Saving top-K tracker consumes 33.6× and 7.6× more chip space and power than the CM-Sketch top-K tracker, although it can offer better preciseness.

Average access-count ratio. To evaluate the average access-count ratio of HPT and HWT, based on Space-Saving and CM-Sketch for diverse Ns, we use an in-house simulator, as the FPGA-based Space-Saving top-K tracker limits the

Table 4. Size and power consumption of top-5 trackers.

Number of Entries (N)	Size (um^2)		Power (mW)	
	Space-Saving based (CAM)	CM-Sketch based (SRAM)	Space-Saving based (CAM)	CM-Sketch based (SRAM)
50	3,649	1,899	0.7	2.0
100	7,323	2,134	1.3	2.2
512	36,374	2,878	6.4	2.7
1K	89,369	3,714	15.0	3.2
2K	179,625	5,346	29.9	3.9
8K	-	13,509	-	7.9
32K	-	46,930	-	23.2
128K	-	180,530	-	83.8

evaluation only up to N = 50 under the 400MHz constraint. Specifically, we first use Intel Pin [24] and `Ramulator` [28] to collect traces of cache-filtered and time-stamped addresses to DRAM from our server running the four most memory-intensive SPECrate CPU 2017 benchmarks, `Liblinear`, and `PageRank`. Then, we feed the traces into our simulator to evaluate the preciseness of HPT and HWT, each implemented with both Space-Saving and CM-Sketch, while sweeping N. Later, we will use PAC (§7.2) and FPGA-based implementations of HPT, based on Space-Saving and CM-Sketch with 50 and 32K for N, to evaluate the average access-count ratio in the same way as we evaluate the CPU-driven page-migration solutions (§7.2).

Figure 7 plots the average access-count ratio of (a) HPT and (b) HWT, relative to the precise counting of accesses to every page and word by PAC and WAC, respectively. In this evaluation, we first vary N while fixing K to 5 and the query period to $1ms$ and $100\mu s$ for HPT and HWT, respectively. The average access-count ratio of both the Space-Saving and CM-Sketch top-K trackers strongly depends on N. However, the Space-Saving top-K tracker based is more precise than the CM-Sketch top-K tracker for the same N because CM-Sketch severely suffers from hash collisions when N is small. For example, HPT, based on the CM-Sketch top-5 tracker with N = 2K, achieves an average access-count ratio similar to HPT based on the Space-Saving top-5 hot-page tracker with

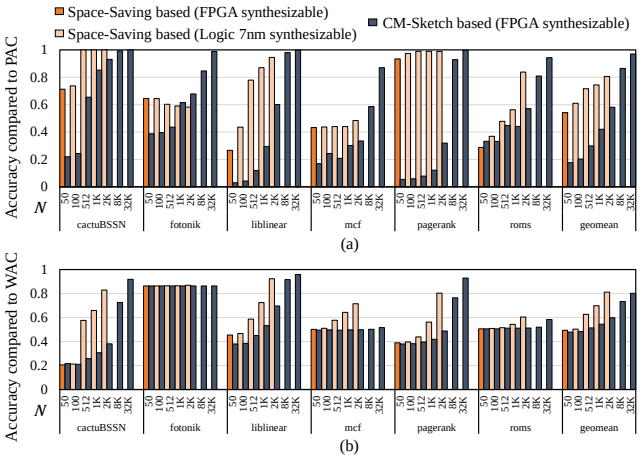

Figure 7. Simulation-based average access-count ratio of (a) HPT and (b) HWT.

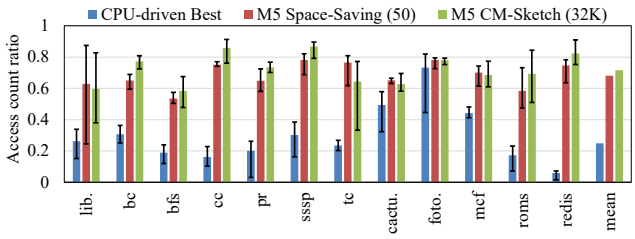

Figure 8. Agilex7-based average access-count ratios of HPT.

N = 50. Nonetheless, under the timing constraint of FPGA-based implementations, we demonstrate that the CM-Sketch top-K tracker can provide a higher average access-count ratio than the Space-Saving top-K tracker with larger N. For instance, HPT, based on the CM-Sketch top-K tracker with 32K entries, which we will use for the evaluation of M5 (§7), offers the average access-count ratio of 0.97 on average across the benchmarks. In contrast, HPT, based on the Space-Saving top-K tracker with 50 CAM entries, offers the average access-count ratio of 0.49 across the benchmarks. Lastly, the average access-count ratio depends on how frequently M5-manager checks with the top-K trackers, and we observe that it increases the preciseness as the interval decreases.

7.2 Full-System Comparison

Average access-count ratio. Figure 8 plots the average access-count ratio of the best CPU-driven page-migration solution between ANB and DAMON, along with M5, using HPT based on both the Space-Saving and CM-Sketch top-K trackers. These trackers are queried at rates determined by `Elector` (Algorithm 1), which will be used for the end-to-end performance evaluation later. This shows that HPT, based on the CM-Sketch top-K tracker with N = 32K, offers 3.5% and 47% higher average access-count ratios than HPT, based on the Space-Saving top-K tracker with N = 50, and the best CPU-driven page-migration solution, respectively, on average.

The average access-count from HPT, based on the CM-Sketch top-K tracker with N = 32K, is 0.72. That is, HPT cannot always identify the true top-K hot pages because M5 determines these hot pages based on the access counts of pages within a limited time window, whereas PAC reports the top-K hot pages based on the entire execution time. As such, HPT can hastily determine less hot pages in a given interval because these pages were accessed intensively during the interval but not so much later. Meanwhile, some pages are not accessed intensively but are constantly accessed throughout the execution, eventually becoming the top-K hot pages at the end. This does not mean that M5 is not as effective as we hope, since it is likely that intensively accessed pages at the moment will remain hot pages in the near future, although this is not always the case for every application. Therefore, to use M5 more effectively, it is essential to develop a sophisticated policy for `Elector`. Lastly, the same argument

can be made for the CPU-driven page-migration solutions, but M5, even with a simple policy, still offers notably higher average access-count ratios for all the benchmarks, proving its effectiveness.

End-to-end performance. Figure 9 plots the end-to-end performance of the benchmarks when ANB, DAMON, and M5 are deployed, normalized to that of ANB. For `Redis`, we use the p99 latency as a performance metric and plot the inverse of the normalized p99 latency values in Figure 9. When starting to evaluate a given benchmark, we allocate all pages of the benchmark to CXL DRAM and then let a given page-migration solution migrate identified hot pages to DDR DRAM. After the given DDR DRAM capacity (3GB) is used up, whenever the page-migration solution migrates a certain number of pages to DDR DRAM, it demotes the same number of pages to CXL DRAM.

Between ANB and DAMON, DAMON provides the highest performance improvement. DAMON offers 6% and 81% higher performance than ANB and no page migration, respectively. Since we provide M5-manager as a sample policy to demonstrate the efficacy of HPT and HWT, we choose $y = x^n$ for `fscale()` and do not use any adaptive algorithm to determine `f_default` for a given benchmark (*i.e.*, out of our intended scope). For this evaluation, we simply try a few reasonable values of n (*e.g.*, 3 to 6) and `f_default` (*e.g.*, 1), and then choose the best performance, which shows 14% and 106% higher performance than DAMON and no page migration, respectively. The performance improvement of M5 over ANB and DAMON comes from two aspects of M5: (1) it incurs a much lower cost for identifying hot pages and (2) it identifies hotter pages than ANB and DAMON. Especially, the performance improvement of `Redis` is 10% and 43% compared to ANB and DAMON, respectively, when we detect hot pages based on HWT-driven `Nominator` (M5 (HWT)), as M5 chooses more useful hot pages than ANB and DAMON with virtually no performance cost.

Note that ANB and M5 improve the performance of `Redis` by 8% and 18-19%, respectively, whereas DAMON degrades the performance by 16%. This is because DAMON continues to scan PTEs even after page migration reaches an equilibrium state (*i.e.*, when the given DDR DRAM capacity is fully occupied), despite further migration providing little performance improvement due to `Redis`' uniform random memory accesses. Meanwhile, we observe that ANB rarely unmaps pages at this state (*i.e.*, incurring little performance overhead for identifying hot pages). This also explains why M5 outperforms DAMON, reemphasizing the importance of minimizing performance overhead when identifying hot pages.

Figure 10 plots the distribution of access counts of all pages in each benchmark, collected with PAC. This partially explains why M5 provides 24% and 14% higher performance than ANB and DAMON, respectively, for `Liblinear`, with no improvement in `PageRank` compared to ANB and DAMON. The access is more skewed in the case of `Liblinear`, which rewards M5 for migrating hot pages more precisely. This also explains why M5 delivers 96% and 30% higher performance than ANB and DAMON, respectively, for `roms_r`, showing superior improvement over the other SPECrate benchmarks. As shown, the p90, p95, and p99 pages of `roms_r` are 2×, 8×, 17× more frequently accessed than the p50 page, respectively.

In contrast, other benchmarks exhibit similar levels of hotness across pages. For instance, once approximately the top p50 hot page in `TC` are migrated to DDR DRAM, the bottom p50 page has only 288 more accesses than that of the bottom p10 page on average. This is not enough of the number of accesses to amortize the cost of page migration (~54μs in our setup), which requires more than 318 accesses (= 54μs/(270ns - 100ns)) on average. For such applications, we should conservatively migrate pages to DDR DRAM after enough pages are migrated to occupy a given capacity of DDR DRAM, as the cost of page migration may outweigh the benefit.

8 Discussion

Scalability of the top-K tracker based on CM-Sketch. As the capacity of CXL memory and the overall memory footprint of applications increase, the address cardinality (*i.e.*, the number of unique addresses) increases. This may increase collisions in the CM-Sketch unit's entries, each estimating (or tracking) the number of accesses to a given address, and thus decrease the accuracy of the top-K tracker. Figure 11 shows the preciseness of the top-K trackers with 32K CM-Sketch entries as the memory footprint increases.

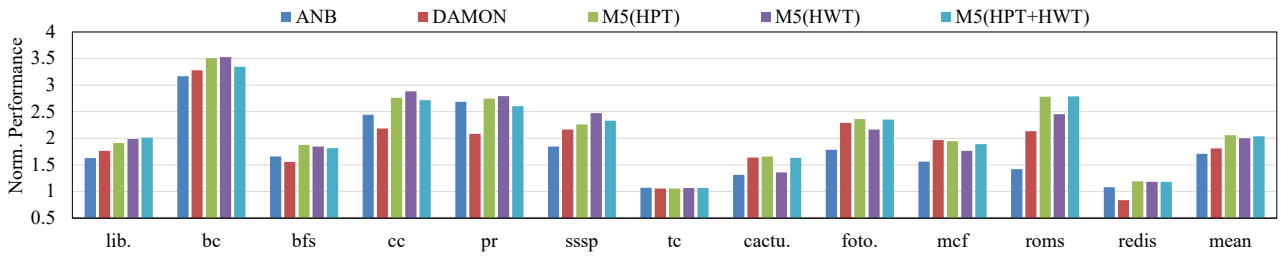

Figure 9. Performance of DAMON and M5 normalized to that of no page migration. 'HPT,' 'HWT,' and 'HPT+HWT' denote HPT-only, HWT-driven, and HPT-driven `Nominator` configurations.

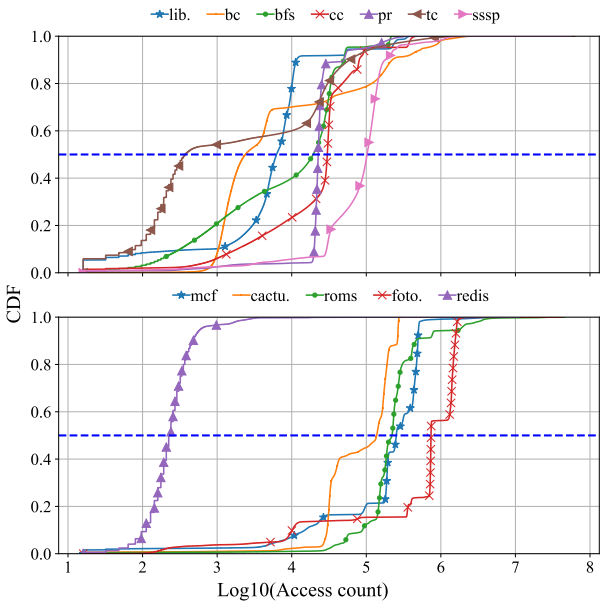

Figure 10. Distribution of access counts per 4KB page.

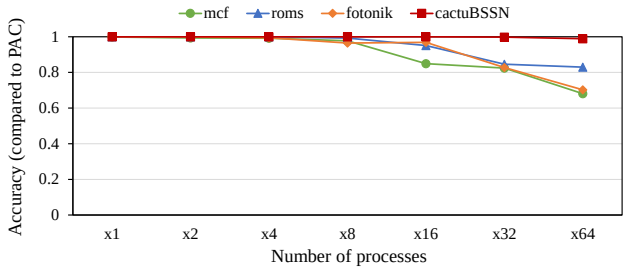

Figure 11. Accuracy of CM-Sketch with 32K entries as the working set size increases.

We increase the memory footprint of `mcf_r`, `cactuBSSN_r`, `fotonik3d_r`, and `roms_r` by increasing the number of co-running instances (or processes), each using a unique range of physical addresses. The 32 processes of these benchmarks demand 20GB to 27GB of memory capacity. This shows that the preciseness of the top-K tracker decreases gracefully as the memory footprint increases. The top-K tracker based on CM-Sketch consumes a small amount of chip space, even for 32K entries, accounting for only 0.01% of the total chip space of all DRAM dies in an 8GB DRAM module. Therefore, for a larger DRAM capacity, we may increase the number of CM-Sketch entries to 128K (Table 4). If we need more than 128K CM-Sketch entries, we may increase the number of top-K tracker instances, each paired with a DRAM rank or module [57]. Nonetheless, we conservatively expect that the chip sapce per GB for the top-K tracker remains mostly constant.

Hot huge pages. The evaluated benchmarks in this work do not allocate huge pages. For applications allocating 2MB huge pages, we can extend M5 to track hot 2M page addresses using hot 4KB page addresses from HPT, as we identify hot 4KB page addresses from the hot 64B word addresses (§5.2). Alternatively, we can deploy another HPT modified to track hot page addresses in 2MB granularity. In both approaches, M5 needs to consult with the OS to check whether these page addresses belong to allocated huge pages.

9 Related Work

Hardware-assisted hot-page detection. A large body of prior work has proposed hardware-assisted mechanisms for identifying hot pages (*e.g.*, MemPod [51], PoM [60], SILC-FM [56], and HoPP [33]). However, these mechanisms are often tightly integrated with the CPU architecture, demanding changes in the CPU's memory subsystem, such as DRAM controllers. Notably, PoM and SILC-FM specifically target the CPU with 3D-stacked DRAM. In contrast, M5 exploits the unique properties of commodity CXL devices built on an industry-standard interface to the CPU. As such, M5 is decoupled from the specific generations of CPU architectures, facilitating the evaluation of page-migration solutions with commodity servers, making it more practical and adoptable.

A recent study [74] explores Intel Flat Memory Mode (IFMM) [39], where DDR memory acts as an exclusive cache for CXL memory. When a CXL memory address is accessed, the memory controller swaps a 64B word at the CXL memory address with a 64B word at the one-to-one mapped DDR memory address. This eliminates the expensive operations needed for page migration, shooting down TLB entries, updating page table entries, and copying entire 4KB pages, especially for sparse hot pages. However, IFMM is limited to the case where local DDR memory and remote CXL memory have the same capacity because of the one-to-one mapping between CXL and DDR memory addresses for swapping. Therefore, M5 can be synergistically used with IFMM when remote CXL DRAM is larger than local DDR DRAM. For example, IFMM can migrate hot words in sparse pages to DDR DRAM while M5 can migrate hot dense pages to DDR DRAM.

We recognize that the latest work, NeoMem, on tracking hot pages in the CXL controller [75] as concurrent work. It shares one of the observations we have made in this work—high cost of identifying hot pages—and then proposes a CM-Sketch-based top-K tracker to tackle this challenge. However, in contrast to NeoMem, we also present PAC and WAC, which have enabled us to precisely evaluate the access-count ratio of identified hot pages, thereby assessing the usefulness of hot pages identified not only by the latest CPU-driven page migration solutions but also by M5. Lastly, NeoMem does not track hot-word addresses, while M5 does.

Hot-value tracking algorithms. Identifying frequently accessed (hot) values in a time- and space-efficient way is a problem that has been extensively studied. Sketch [50], Bloom filter [37], cuckoo filter [17], and HyperLogLog [19]

have been applied in multiple applications, including heavy-hitter detection and packet filtering in networking flows [3, 47], data analytics [9, 13], key-value data store [34, 73], and genomic analysis in computational biology [29]. In this work, we repurposed CM-Sketch, a streaming algorithm, to track hot page and word addresses for a tiered-memory system.

10 Conclusion

To tackle the challenges of effectively managing tiered-memory systems, first, this work proposed a CXL-driven, hardware-assisted profiling solution that leverages the unique capabilities of an FPGA-based CXL device. Second, this work showed that CPU-driven page-migration solutions often not only classify warm pages as hot pages—many of which may also be sparse—but also incur notable performance overhead. Lastly, this work presented M5, a platform for developing CXL-driven page-migration solutions. M5 provides hardware-based hot page and word trackers in the CXL controller, along with software interfaces and guidelines for designing effective migration policies. Our evaluation has demonstrated that M5 can identify useful hot pages more precisely than the CPU-driven page migration solutions, with virtually no performance overhead, leading to higher application performance.

Acknowledgments

This work was supported in part by a grant from PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00405857), by the MSIT, Korea, under the Global Scholars Invitation Program (RS-2024-00456287) supervised by the IITP, by the Samsung Electronics, and by a generous gift from Intel Corp.

A Artifact Appendix

A.1 Abstract

The artifact includes the source code of M5 which is composed of PAC (Page Access Counter), HPT (Hot Page Tracker), HWT (Hot Word Tracker), the userspace M5-manager, and finally the kernel changes involved to support M5. Its goal is to reproduce the results of Figure 8, 9, and 10 in the paper comparing with Automatic NUMA Balancing (ANB) and DAMON. The artifact also contains instructions and scripts for setting up a system and conducting experiments.

A.2 Artifact check-list (meta-information)

- **Program:** Four benchmarks from SPECrate CPU 2017 (505.mcf_r, 507.cactuBSSN_r, 549.roms_r, and 554.roms_r), Redis version 6.0.16, `Liblinear` version 2.47, and GAP Benchmark Suite (`BFS`, `SSSP`, `PR`, `CC`, `BC`, and `TC`). The details of benchmarks are described in Table 3 in the paper. All benchmarks are public.
- **Compilation:** `gcc-11.4` for M5 and ANB, and DAMON. The FPGA project is compiled with Intel Quartus Prime v23.2.
- **Data set:** We use the YCSB-A [11] workload and the KDD 2012 [35] for `Redis` and `Liblinear`, respectively, while Twitter [30] and Google [1] as inputs for `GAP`.
- **Run-time environment:** M5 with Linux kernel version 6.5, ANB with Linux kernel version 5.19, and DAMON runs with the Linux kernel version 6.11.
- **Hardware:** Intel Xeon Scalable processor supporting CXL and an Intel Agilex 7 I-series FPGA, version R1BES, listed in Table 2.
- **Run-time state:** An idle system.
- **Metrics:** The access count ratio is reported from the experiment of Figure 8. YCSB running with Redis reports the 99th percentile latency in nanoseconds. All the other benchmarks in Figure 9 reports the execution time in seconds.
- **Output:** The experiment scripts output the result to text files. Figures 8, 9, and 10 show the expected results.
- **Experiments Figure 8:** The experiment compares the tracking accuracy of various tracking methods. We have each scheme 1) detect hot pages, 2) retrieve access count for the access to these hot pages, and 3) analyze how many times they are accessed.
- **Experiments Figure 9:** The end-to-end performance evaluation for application in each scheme.
- **Experiments Figure 10:** We plot the access count for each application in Figure 8, to draw a CDF plot of the access count.
- **How much disk space required (approximately)?:** Around 20 GB for GAPBS graph data sets and 50 GB for SPEC 2017. Finally, about 300 GB is required for storing the generated results.
- **How much time is needed to prepare workflow (approximately)?:** About 3 hours.
- **How much time is needed to complete experiments (approximately)?:** About 48 hours.
- **Publicly available?:** All software and hardware RTL are available in https://github.com/ece-fast-lab/ASPLOS-25-M5.
- **Code licenses (if publicly available)?:** GPL 2.0
- **Archived:** https://zenodo.org/records/14303160

A.3 Description

A.3.1 How to access. The source code is provided at https://github.com/ece-fast-lab/ASPLOS-2025-M5. In the rest of the section, `repo` refers to directory that holds the aforementioned GitHub repository. All benchmarks, except for SPEC CPU2017, are publicly available:

- GAPBs: https://github.com/sbeamer/gapbs
- YCSB: https://github.com/brianfrankcooper/YCSB
- Liblinear: https://www.csie.ntu.edu.tw/~cjlin/liblinear/

A.4 Installation

A.4.1 Hardware. The following steps describe how to configure the FPGA.

```
$ cd repo/hw
```

```
$ bash program_fpga.sh bitstreams/m5_cm_banking.cdf
$ sudo poweroff
$ <Power on with BMC>
$ cd repo/setup
$ bash setup_m5.sh
```

A.4.2 Software. The `repo` provides one folder for each tested kernel in `repo/kernels`. The file paths in `setup_m5.sh` needs to be updated for the benchmark and binary paths. Please refer to `repo/README.md` for compiling and switching the kernel between tests.

A.5 Experiment workflow

A.5.1 Setup. This setup will setup the platform for testing.

- **Kernel switch:** Edit `/etc/default/grub`, and apply the changes with `update-grub`.
- **Program FPGA / Setup:** Instructions listed in the Section A.4.

A.5.2 Experiments. The experiments are embedded in the provided scripts, and the input arguments depend on the test case. Please refer to the `testing_scripts/README.md` for more details. The data of Figure 10 is part of Figure 8.

```
$ cd repo/testing_scripts
$ bash fig8_eval_all.sh <arg>
$ bash fig9_eval_all.sh <arg>
```

The following steps describe how to parse and visualize the results. All steps are performed in the directory `repo/results`. The results will be stored in generated PDF files.

```
$ bash organize_results.sh
$ bash parse_all_figs.sh
$ python3 plot_all_figs.py
```

A.6 Evaluation and expected results

The experiment in Figure 8 shows a high access count ratio for the CM-sketch algorithm compared to the software schemes.The end-to-end performance evaluation in Figure 9 shows M5 outperforms the software baselines by 14%. Finally, Figure 10 shows the access distribution of the tested benchmarks.

References

[1] Abdullah T. Mughrabi. accessed in 2024. GraphBrew GAPBS. https://github.com/UVA-LavaLab/GraphBrew.

[2] Ahmed Abulila, Vikram Sharma Mailthody, Zaid Qureshi, Jian Huang, Nam Sung Kim, Jinjun Xiong, and Wen-mei Hwu. 2019. FlatFlash: Exploiting the Byte-Accessibility of SSDs within a Unified Memory-Storage Hierarchy. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems.*

[3] Anup Agarwal, Zaoxing Liu, and Srinivasan Seshan. 2022. HeteroSketch: Coordinating Network-wide Monitoring in Heterogeneous and Dynamic Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation.*

[4] 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.*

[5] Andrea Arcangeli. accessed in 2024. AutoNUMA. https://blog.linuxplumbersconf.org/2012/wp-content/uploads/2012/09/2012-lpc-virt-autonuma-arcangeli.pdf.

[6] Scott Beamer, Krste Asanovic, and David A. Patterson. 2015. The GAP Benchmark Suite. *CoRR* abs/1508.03619 (2015). arXiv:1508.03619 http://arxiv.org/abs/1508.03619

[7] Shai Bergman, Priyank Faldu, Boris Grot, Lluís Vilanova, and Mark Silberstein. 2022. Reconsidering OS Memory Optimizations in the Presence of Disaggregated Memory. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on Memory Management.* ACM, New York, NY, USA, 1–14.

[8] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. 2021. Rethinking Software Runtimes for Disaggregated Memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems.*

[9] Monica Chiosa, Thomas B. Preußer, and Gustavo Alonso. 2021. SKT: A One-Pass Multi-Sketch Data Analytics Accelerator. *Proceedings of the VLDB Endowment* 14, 11 (2021).

[10] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *Microelectronics Journal* 53 (2016), 105–115.

[11] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, Indiana, USA).

[12] Graham Cormode and Shan Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[13] Michael Correll and Michael Gleicher. 2016. The Semantics of Sketch: Flexibility In Visual Query Systems For Time Series Data. In *2016 IEEE Conference on Visual Analytics Science and Technology.*

[14] CXL Consortium. accessed in 2025. Compute Express Link (CXL). https://www.computeexpresslink.org.

[15] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, Brian Morris, Chiranjit Mukherjee, Jingliang Ren, Greg Thelen, Paul Turner, Carlos Villavieja, Parthasarathy Ranganathan, and Amin Vahdat. 2023. Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems.*

[16] Cristian Estan and George Varghese. 2002. New Directions in Traffic Measurement and Accounting. In *Proceedings of the*

2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. 323–336.

[17] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 75–88.

[18] Vishal Gupta, Min Lee, and Karsten Schwan. 2015. HeteroVisor: Exploiting Resource Heterogeneity to Enhance the Elasticity of Cloud Platforms. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*.

[19] Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*.

[20] Intel. accessed in 2024. Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B. https://cdrdv2.intel.com/v1/dl/getContent/671427.

[21] Intel Corporation. accessed in 2023. Intel Xeon Gold 6430 Processor. https://ark.intel.com/content/www/us/en/ark/products/231737/intel-xeon-gold-6430-processor-60m-cache-2-10-ghz.html.

[22] Intel Corporation. accessed in 2023. Intel® Agilex™ 7 FPGA I-Series Development Kit. https://www.intel.com/content/www/us/en/products/details/fpga/development-kits/agilex/i-series/dev-agi027.html.

[23] Intel Corporation. accessed in 2024. Improving Real-Time Performance by Utilizing Cache Allocation Technology. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf.

[24] Intel Corporation. accessed in 2024. Pin - A Dynamic Binary Instrumentation Tool. https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html.

[25] Jonathan Corbet. accessed in 2024. Merging the multi-generational LRU. https://lwn.net/Articles/894859/.

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

[27] Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, Jae W. Lee, and Jung Ho Ahn. 2022. Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh. In *2022 IEEE International Symposium on High-Performance Computer Architecture*. 1156–1169. https://doi.org/10.1109/HPCA53966.2022.00088

[28] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2015), 45–49.

[29] Can Kockan, Kaiyuan Zhu, Natnatee Dokmai, Nikolai Karpov, M Oguzhan Kulekci, David P Woodruff, and S Cenk Sahinalp. 2020. Sketching Algorithms for Genomic Data Analysis and Querying in a Secure Enclave. *Nature methods* 17, 3 (2020).

[30] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a Social Network or a News Media?. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA). Association for Computing Machinery, New York, NY, USA, 591–600. https://doi.org/10.1145/1772690.1772751

[31] 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 Symposium on Operating Systems Principles*.

[32] Yunjae Lee, Yoonhee Kim, and Heon Y. Yeom. 2020. Lightweight Memory Tracing for Hot Data Identification. *Cluster Computing* 23 (09 2020). https://doi.org/10.1007/s10586-020-03130-1

[33] Haifeng Li, Ke Liu, Ting Liang, Zuojun Li, Tianyue Lu, Hui Yuan, Yinben Xia, Yungang Bao, Mingyu Chen, and Yizhou Shan. 2023. HoPP: Hardware-Software Co-Designed Page Prefetching for Disaggregated Memory. In *IEEE International Symposium on High-Performance Computer Architecture*.

[34] Yongkun Li, Chengjin Tian, Fan Guo, Cheng Li, and Yinlong Xu. 2019. ElasticBF: Elastic Bloom Filter with Hotness Awareness for Boosting Read Performance in Large Key-Value Stores. In *2019 USENIX Annual Technical Conference*.

[35] Chih-Jen Lin. accessed in 2024. KDD CUP 2012 Dataset in LIBSVM Format. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#kdd2012.

[36] Chih-Jen Lin. accessed in 2024. LIBLINEAR – A Library for Large Linear Classification. https://www.csie.ntu.edu.tw/~cjlin/liblinear/.

[37] Lailong Luo, Deke Guo, Richard T. B. Ma, Ori Rottenstreich, and Xueshan Luo. 2019. Optimizing Bloom Filter: Challenges, Solutions, and Comparisons. *IEEE Communications Surveys & Tutorials* 21, 2 (2019).

[38] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate Frequency Counts over Data Streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 346–357.

[39] Manoj Sukumaran. accessed in 2024. Orchestrating Memory Disaggregation with Compute Express Link. https://www.intel.com/content/www/us/en/content-details/817889/orchestrating-memory-disaggregation-with-compute-express-link.html.

[40] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. 2022. PROTRR: Principled yet Optimal In-DRAM Target Row Refresh. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[41] Adnan Maruf, Ashikee Ghosh, Janki Bhimani, Daniel Campello, Andy Rudoff, and Raju Rangaswami. 2022. MULTI-CLOCK: Dynamic Tiering for Hybrid Memory Systems. In *IEEE International Symposium on High-Performance Computer Architecture*.

[42] 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 ACM International Conference on Architectural Support for Programming*

Languages and Operating Systems.

[43] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proceedings of the 10th International Conference on Database Theory*. Springer, 398–412.

[44] Michael Larabel. accessed in 2024. MGLRU Continues To Look Very Promising For Linux Kernel Performance. https://www.phoronix.com/news/Linux-MGLRU-v9-Promising.

[45] Shanmugavelayutham Muthukrishnan. 2005. Data Streams: Algorithms and Applications. *Foundations and Trends® in Theoretical Computer Science* (2005).

[46] Alan Nair, Sandeep Kumar, Aravinda Prasad, Ying Huang, Andy Rudoff, and Sreenivas Subramoney. 2024. Telescope: Telemetry for Gargantuan Memory Footprint Applications. In *Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference*.

[47] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. 2023. Sketchovsky: Enabling Ensembles of Sketches on Programmable Switches. In *20th USENIX Symposium on Networked Systems Design and Implementation*.

[48] SeongJae Park. accessed in 2024. DAMON: Data Access Monitor. https://sjp38.github.io/post/damon/.

[49] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W. Lee. 2020. Graphene: Strong yet Lightweight Row Hammer Protection. In *Proceedings of the 53rd IEEE/ACM International Symposium on Microarchitecture*.

[50] Mert Pilanci and Martin J. Wainwright. 2017. Newton Sketch: A Near Linear-Time Optimization Algorithm with Linear-Quadratic Convergence. *SIAM Journal on Optimization* 27, 1 (2017).

[51] 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 *2017 IEEE International Symposium on High Performance Computer Architecture*.

[52] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, C. J. Newburn, Dmitri Vainbrand, I-Hsin Chung, Michael Garland, William Dally, and Wen-mei Hwu. 2023. GPU-Initiated On-Demand High-Throughput Storage Access in the BaM System Architecture. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*.

[53] 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 ACM SIGOPS 28th Symposium on Operating Systems Principles*.

[54] Redis Ltd. accessed in 2023. Redis. https://redis.io/.

[55] Jie Ren, Dong Xu, Junhee Ryu, Kwangsik Shin, Daewoo Kim, and Dong Li. 2024. MTM: Rethinking Memory Profiling and Migration for Multi-Tiered Large Memory. In *Proceedings of the 19th European Conference on Computer Systems*.

[56] Jee Ho Ryoo, Mitesh R. Meswani, Andreas Prodromou, and Lizy K. John. 2017. SILC-FM: Subblocked InterLeaved Cache-Like Flat Memory Organization. In *2017 IEEE International Symposium on High Performance Computer Architecture*.

[57] Samir Rajadnya, Durgesh Srivastava. accessed in 2024. CMS: Hotness Tracking Requirements. https://www.opencompute.org/documents/ocp-cms-hotness-tracking-requirements-white-paper-pdf-1.

[58] Debendra Das Sharma. 2022. Compute Express Link (CXL): Enabling Heterogeneous Data-Centric Computing With Heterogeneous Memory Hierarchy. *IEEE Micro* (2022). https://doi.org/10.1109/MM.2022.3228561

[59] Sebastian Andrzej Siewior, Rusty Russell, Srivatsa Vaddagiri, Ashok Raj, Joel Schopp, and Thomas Gleixner. accessed in 2024. CPU hotplug in the Kernel. https://docs.kernel.org/core-api/cpu_hotplug.html.

[60] Jaewoong Sim, Alaa R. Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. 2014. Transparent Hardware Management of Stacked DRAM as Part of Memory. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*.

[61] Standard Performance Evaluation Corporation. accessed in 2024. SPEC CPU2017 Benchmark Suites. https://www.spec.org/cpu2017/.

[62] 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 Annual IEEE/ACM International Symposium on Microarchitecture*.

[63] Yupeng Tang, Ping Zhou, Wenhui Zhang, Henry Hu, Qirui Yang, Hao Xiang, Tongping Liu, Jiaxin Shan, Ruoyun Huang, Cheng Zhao, Cheng Chen, Hui Zhang, Fei Liu, Shuai Zhang, Xiaoning Ding, and Jianjun Chen. 2024. Exploring Performance and Cost Optimization with ASIC-Based CXL Memory. In *Proceedings of the Nineteenth European Conference on Computer Systems*. New York, NY, USA.

[64] Vinicius Petrucci, Eishan Mirakhur, Rita Gupta, Mahesh Wagh, Nikesh Agarwal, Su Wei Lim, Vishal Tanna. accessed in 2025. CXL Memory Expansion: A Closer Look on Actual Platform. https://www.micron.com/content/dam/micron/global/public/products/white-paper/cxl-memory-expansion-a-close-look-on-actual-platform.pdf.

[65] Vishal Verma. accessed in 2024. Tiering-0.8. https://git.kernel.org/pub/scm/linux/kernel/git/vishal/tiering.git/log/?h=tiering-0.8.

[66] Hao Wang, Chang-Jae Park, Gyung-su Byun, Jung Ho Ahn, and Nam Sung Kim. 2015. Alloy: Parallel-Serial Memory Channel Architecture for Single-Chip Heterogeneous Processor Systems. In *IEEE International Symposium on High Performance Computer Architecture*.

[67] Lingfeng Xiang, Zhen Lin, Weishu Deng, Hui Lu, Jia Rao, Yifan Yuan, and Ren Wang. 2024. NOMAD: Non-Exclusive Memory Tiering via Transactional Page Migration. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*.

[68] 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*.

[69] Ying Huang. accessed in 2024. Memory Tiering: Hot Page Selection with Hint Page Fault Latency. https://patchwork.kernel.org/project/linux-mm/patch/20210722031819.3446711-5-ying.huang@intel.com/.

[70] Jung. H. Yoon. 2015. 3D NAND Technology: Implications to Enterprise Storage Applications. In *Proceedings of the Flash Memory Summit (FMS'15)*.

[71] Yu Zhao. accessed in 2024. Java / POWER9 Benchmark with MGLRU. https://lore.kernel.org/lkml/20221221000748.1374772-1-yuzhao@google.com/.

[72] Yu Zhao. accessed in 2024. Multi-Gen LRU Framework. https://lore.kernel.org/lkml/20220614071650.206064-1-yuzhao@google.com/l.

[73] Weitao Zhang, Yinlong Xu, Yongkun Li, Yueming Zhang, and Dinglong Li. 2018. FlameDB: A Key-Value Store With Grouped Level Structure and Heterogeneous Bloom Filter. *IEEE Access* (2018).

[74] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. 2024. Managing Memory Tiers with CXL in Virtualized Environments. In *18th USENIX Symposium on Operating Systems Design and Implementation*.

[75] 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 *57th IEEE/ACM International Symposium on Microarchitecture*.