Can Hardware Outsmart Software in Tiered Memory Management? A CMM-H Case Study

Zhen Lin
zxl5722@mavs.uta.edu
The University of Texas at Arlington
Arlington, Texas, USA

Yujie Yang
yxy1115@mavs.uta.edu
The University of Texas at Arlington
Arlington, Texas, USA

Lingfeng Xiang
lingfeng.xiang@mavs.uta.edu
The University of Texas at Arlington
Arlington, Texas, USA

Lianjie Cao
lianjie.cao@hpe.com
Hewlett Packard Labs
Milpitas, California, USA

Faraz Ahmed
faraz.ahmed@hpe.com
Hewlett Packard Labs
Milpitas, California, USA

Jia Rao
jia.rao@uta.edu
The University of Texas at Arlington
Arlington, Texas, USA

Hui Lu
hui.lu@uta.edu
The University of Texas at Arlington
Arlington, Texas, USA

Puneet Sharma
puneet.sharma@hpe.com
Hewlett Packard Labs
Milpitas, California, USA

Abstract

With the advent of Compute Express Link (CXL), hardware-managed memory tiering has become a reality. In this paper, we investigate Samsung's CXL Memory Module–Hybrid (CMM-H), a CXL Type 3 device integrating DRAM and NAND flash managed by an FPGA-based controller and providing byte-addressable memory interface via the `cxl.mem` protocol. We perform a detailed evaluation of CMM-H and compare its performance with OS-level and block-level tiering solutions. Our results highlight the performance benefits of CMM-H for cache-hit scenarios and identify key limitations for cache-miss situations, offering insights into the trade-offs involved in adopting hardware-managed memory tiering in emerging CXL-based systems.

CCS Concepts: • **Hardware**; • **Computer systems organization** → **Architectures**;

Keywords: CXL, CMMH, Memory Tiering, Caching

ACM Reference Format:
Zhen Lin, Yujie Yang, Lingfeng Xiang, Lianjie Cao, Faraz Ahmed, Jia Rao, Hui Lu, and Puneet Sharma. 2025. Can Hardware Outsmart Software in Tiered Memory Management? A CMM-H Case Study. In *The 18th ACM International Systems and Storage Conference (SYSTOR '25), September 8–9, 2025, Virtual, Israel.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3757347.3759140

SYSTOR '25, Virtual

ACM ISBN 979-8-4007-2119-9/2025/09
https://doi.org/10.1145/3757347.3759140

1 Introduction

Modern computer systems increasingly rely on tiered memory and storage hierarchies to balance performance, capacity, and cost by combining a fast but expensive performance tier such as DRAM with a slow but large capacity tier such as non-volatile memory (NVM), flash-based SSDs, or hard disk drives (HDD). Extensive research has investigated a wide range of tiered storage and memory systems, including combinations of SSDs and HDDs [11, 14, 18, 21, 25, 33, 41, 45, 49], DRAM and disks [17, 20, 23, 38], High-Bandwidth Memory (HBM) and DRAM [19, 37, 40], NUMA-based memory architectures [1, 5], persistent memory (PM) and DRAM [12, 13, 30, 31, 35], local and remote (far) memory tiers [16, 22, 26, 27, 39, 46], as well as systems integrating multiple hierarchical tiers [24, 28, 31, 43, 48].

As we enter the era of Compute Express Link (CXL) [4], CXL-enabled devices are becoming an increasingly important new tier in memory and storage hierarchies. CXL is an open, industry-standard interconnect built on PCI Express (PCIe) that offers a memory-like, byte-addressable, and cache-coherent interface, enabling the disaggregation of diverse memory and storage devices such as DRAM, persistent memory (PM), and SSDs. Samsung recently unveiled the CXL Memory Module–Hybrid (CMM-H) [8], a CXL Type 3 device that integrates DRAM and NAND flash within a single module, featuring an internally hardware-managed tiered memory architecture.

CMM-H presents appealing and unique features compared to the existing tiered memory/storage solutions: (1) Unlike traditional software tiering that relies heavily on OS- [34, 36, 46] or application-level management [15], CMM-H uses a *hardware-managed*, FPGA-based controller to enable memory tiering between an internal DRAM cache and a

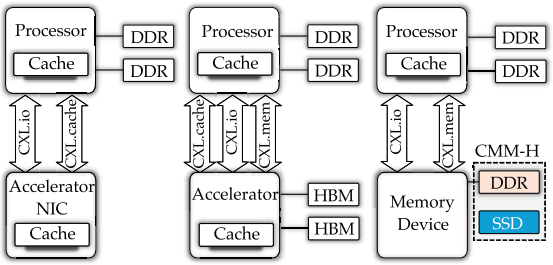


Figure 1. CXL protocols and devices.

NAND SSD. (2) CMM-H provides a byte-addressable memory programming interface (via the `CXL.mem` protocol), allowing the CPU to directly access data in the SSD via ordinary `load` and `store` instructions. (3) CMM-H can operate in dual modes – tiered memory mode, using DRAM as a performance cache for frequently accessed data, and persistent memory mode, ensuring durability using an internal power source and supporting CXL Global Persistent Flush (GPF) to ensure that the cached data is written to the SSD upon a crash.

In this paper, we seek to answer the following questions – (1) *Can hardware-managed memory tiering within the CMM-H hybrid device achieve performance close to traditional DRAM memory while benefiting from NAND SSD's large capacity?* (2) *How does hardware-managed tiering compare to software-managed tiering across different workloads?* To this end, we conduct a comprehensive evaluation of CMM-H, measuring its latency, throughput, and scalability under diverse data-access scenarios. Unlike prior work [51] that primarily evaluates CMM-H's cache-hit performance, our study explicitly analyzes both cache-hit and cache-miss behaviors, exposing practical bottlenecks that arise under real-world usage. We compare CMM-H against two popular software-based memory tiering schemes: an OS-based approach that maps (`mmap`) storage into user-level memory space and relies on page replacement to manage data across memory tiers, and a block-level caching system, Open CAS [9]. Our analysis not only reveals the trade-offs between hardware and software tiering approaches, but also uncovers critical performance characteristics (§3), identifies the key limitations of CMM-H compared to software-based tiered memory approaches (§4), and offers valuable insights for future innovations in tiered memory and storage systems.(§5).

2 Background and Related Work

CXL-based memory expansion: Compute Express Link (CXL) [4] is an open, high-performance interconnect standard designed to enable low-latency, memory-coherent communication between CPUs, accelerators (e.g., GPUs, DPUs, FPGAs), and memory devices over the PCIe physical interface. Its primary goal is to enable coherent memory sharing across connected devices, allowing them to access each other's memory, thus eliminating costly data movement.

Figure 1 shows CXL protocols and various types of CXL devices and their relationship to the host processor. First, CXL supports three key protocols: `CXL.io` for device discovery, configuration, and control; `CXL.cache` for maintaining cache coherence between the host processor and attached devices; and `CXL.mem` for enabling direct, byte-addressable memory access between the CPU and device memory. Further, CXL devices are categorized into three types based on their memory and caching capabilities. Type 1 devices use the `CXL.cache` protocol to provide cache-coherent access to host memory but do not include device-managed memory. Type 2 devices also utilize `CXL.cache` for cache coherence with the host but include their own device-managed memory, enabling coherent memory sharing between the device and the host CPU. They require dedicated hardware coherence mechanisms, and currently, no fully compliant commercial Type 2 devices exist. In contrast, Type 3 devices use the `CXL.mem` protocol, allowing the host processor direct, cacheable access (via `load/store` instructions) to memory in CPU-less devices, which typically serve as memory expanders.

Samsung CMM-H: Although CXL Type 3 devices, initially designed for DRAM-based memory expansion, have recently evolved to incorporate NAND flash as a backing store for a more cost-effective solution, such as Samsung's CXL Memory Module-Hybrid (CMM-H) [8].

CMM-H combines a hardware-managed DRAM cache and NAND flash SSD within a single CXL Type 3 device. A hardware controller implemented on an FPGA manages data placement and movement at a 4 KB page granularity, buffering frequently accessed data ("hot data") in DRAM to reduce the latency associated with NAND flash. This enables CMM-H to (potentially) deliver near-DRAM performance for cache-hit accesses, while providing the scalability and cost-efficiency associated with NAND flash storage. The CMM-H interfaces with the host system via a PCIe Gen5 x8 connection, complying with the CXL 1.1 specification and supporting both the `CXL.mem` and `CXL.io` protocols. Additionally, the device includes a hardware-supported persistence mechanism: using an internal backup power source, CMM-H flushes data from DRAM to NAND flash upon power loss, supporting the CXL Global Persistent Flush (GPF) protocol. As a result, CMM-H can function either as volatile memory expansion or as persistent memory.

Differences from traditional hardware: Compared to conventional block devices such as SSDs, CMM-H presents a fundamentally different design and interface. Traditional SSDs are accessed via block device drivers, using block-addressable interfaces (e.g., NVMe), while CMM-H exposes a byte-addressable, memory-semantic interface via `cxl.mem`, enabling direct load/store access. However, internally, CMM-H still relies on NAND flash, which operates on 4 KB block granularity. This mismatch between fine-grained memory access and coarse-grained flash operations introduces inefficiencies such as costly read-modify-write (RMW) on small,

random writes. Moreover, unlike SSDs that are loosely coupled with CPUs through the I/O subsystem, CMM-H is tightly integrated into the memory hierarchy via *cxl.mem, making its performance characteristics more sensitive to internal DRAM cache-hit rate.*

Related work: Recent studies [32, 42] have investigated the performance characteristics of CXL devices, using both FPGA-based CXL prototypes and commercial ASIC-based CXL products. A key finding from [42] reveals that true CXL memory achieves lower latency and higher bandwidth efficiency compared to emulated CXL memory, showing promise as memory bandwidth expanders. [32] provides a more detailed analysis of CXL-induced slowdowns, linking them to increased stall cycles in the CPU backend and reduced cache prefetch efficiency. [50] developed a more comprehensive understanding of the architectural interactions between CXL memory, the host processor, and the memory hierarchy. However, they all focus on DRAM-based memory expansion. The work most closely related to ours is by Zeng et al. [51], which presents a comprehensive performance characterization of the Samsung CMM-H prototype, providing practical guidelines for its usage. Our study differs from it by directly comparing hardware-managed (CMM-H) and software-managed solutions, explicitly evaluating cache-hit and cache-miss scenarios across diverse data access patterns and concurrency levels, and revealing both strengths and weaknesses of CMM-H in practical tiered memory scenarios. In addition, our study evaluates the most recent version of the CMM-H device, which differs significantly from the earlier prototype in [51], featuring a larger DRAM cache (48 vs. 16 GB) and a more advanced FPGA-based controller.

3 Performance Characterization

This section presents a performance characterization of the CMM-H device, measuring its latency, bandwidth, and scalability with parallel threads. We also identify key performance bottlenecks and discuss potential mitigation strategies.

Experimental configurations: Our experiments were conducted on a server equipped with one Intel Xeon Platinum 8556 processor, eight 32 GB DDR5-4800 MHz DRAM modules, and one Samsung CXL CMM-H device. The CMM-H device includes a 48 GB DRAM cache and a 1 TB Samsung PM9A3 NAND SSD [10], managed internally by an FPGA-based hardware controller (Intel Altera Agilex 7). Detailed hardware configurations are provided in [3]. For comparison, the server was also equipped with one 256 GB Micron CZ120 CXL memory module and one 960 GB Samsung PM9A3 NVMe SSD. The operating system used for testing was Ubuntu Server 24.04.2 LTS, running Linux kernel version 6.8.0-53-generic.

To evaluate the latency and bandwidth performance of the CMM-H device accessed via its `CXL.mem` interface, we implemented a dedicated micro-benchmark. Before each measurement, we pre-warmed the CPU cache and the internal DRAM and NAND SSD of the CMM-H device to ensure the initial conditions were consistent across experiments.

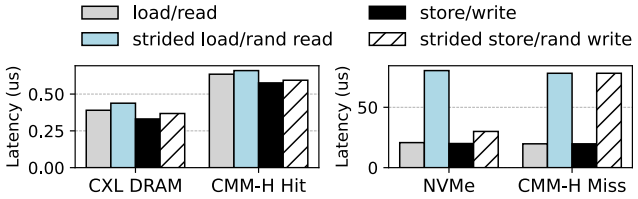


Figure 2. Single-thread idle latency.

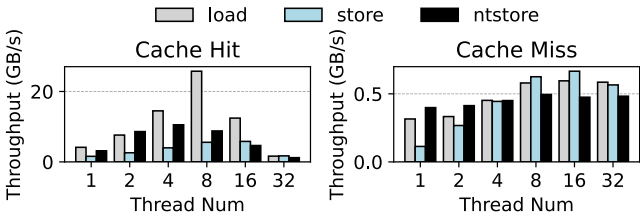


Figure 3. Bandwidth comparison.

Latency tests: We measured the *idle latency* of single-threaded memory access on CMM-H under *no-load* conditions. First, we evaluated latency for accesses hitting the performance tier (i.e., DRAM cache in CMM-H) and compared it with the Micron CZ120 CXL memory module; both devices are Type 3 CXL memory. Next, we measured latency for accesses reaching the capacity tier (i.e., DRAM cache miss), backed by NAND SSD storage within CMM-H. For comparison, we measured the latency of a standalone NVMe SSD connected directly via PCIe using FIO [7]. Figure 2 illustrates the idle latency for both the performance and capacity tiers, measured across different memory access patterns – sequential load/store and strided (or random) load/store with a 16 KB stride, where each access touches 64 B of data for DRAM access. FIO uses 4 KB I/Os to test the standalone SSD.

First, accesses to Micron CXL memory demonstrate consistently low latency (~0.4 µs), whereas the cache-hit latency in CMM-H is uniformly higher (~0.6 µs), irrespective of access pattern. The increased latency observed with CMM-H arises primarily due to two factors: (1) additional overhead from internal tiered memory management operations (e.g., cache-line lookups), and (2) the FPGA-based implementation, which is less efficient than the ASIC-based implementation used in the Micron CXL device.

Interestingly, the cache-miss latency of CMM-H closely matches that of the NVMe SSD (except for strided writes), with both exhibiting significantly higher latencies (20–80 µs), despite differences in (1) their access interfaces (byte-addressable memory interface in CMM-H versus block -addressable I/O interface in the NVMe SSD) and (2) their access data sizes (64 bytes for CMM-H and 4 KB for the NVMe SSD). This similarity arises from the *granularity mismatch* between the byte-addressable memory interface of CMM-H and its internal NVMe block interface. Specifically, on store cache misses, CMM-H must perform a Read-Modify-Write

(RMW) operation, first reading an entire 4 KB page from the internal SSD before updating the smaller requested data portion (e.g., 64 B). Consequently, even small (64 B) accesses incur latency comparable to the 4 KB I/O operations of the NVMe SSD, except for strided writes.

For strided writes, the NVMe SSD achieves substantially lower latency due to its internal DRAM buffer, which quickly caches data and asynchronously flushes it to the slower NAND media. In contrast, CMM-H cannot mask NAND latency, as the RMW operation requires it to synchronously read the 4 KB data directly from NAND media.

Takeaway #1: *CMM-H exhibits higher cache-hit latency than CXL memory due to internal hardware-based tiered memory management overhead; its cache-miss latency is significantly increased (up to 200x) due to slow internal NAND flash and costly RMW operations caused by the granularity mismatch between the external byte-addressable interface (via* `cxl.mem`*) and internal block-addressable NAND storage.*

Bandwidth tests: We measured CMM-H bandwidth using varying thread counts and three memory instructions (`load`, `store`, and `ntstore`). Each thread sequentially accesses its dedicated 1 GB region, i.e., working set size (WSS).

Figure 3 presents the measured throughput of CMM-H under conditions where the device's DRAM cache is prewarmed prior to each test run. We observe that throughput initially scales linearly with increasing thread count, reaching a peak of approximately 25 GB/s for `load` operations at eight threads. However, unexpectedly, throughput declines sharply beyond eight threads, dropping by more than 90% at 32 threads. Further investigation reveals that the observed reduction in throughput primarily results from cache contention and the corresponding increase in cache miss rate. Although the total WSS remains well below the 48 GB DRAM cache capacity of the CMM-H, the device employs a coarse-grained, 8-way associative caching architecture [51], rather than a direct-mapped cache, likely due to its simpler and more efficient hardware implementation. However, this choice leads to increased contention under concurrent memory accesses. As further illustrated in Figure 4, increasing WSS results in a higher cache miss rate, causing additional internal traffic between the DRAM cache and NAND SSD tiers due to cache replacement operations. Given that the overhead associated with cache replacement in CMM-H appears significant, even a modest increase in the cache miss rate substantially degrades the overall throughput.

Inspired by [47], we developed a simple software-based prefetching approach that leverages a dedicated prefetcher thread running slightly ahead of each regular worker threads accessing the same data. By accessing the data a bit earlier, the prefetcher thread proactively loads data into the cache, effectively reducing cache misses for its subsequent worker thread. As illustrated in Figure 5, this prefetching mechanism greatly enhances the throughput of worker threads (by 6x), particularly under memory usage of 32 GB and 40 GB, where cache contention is notably pronounced. Our prefetching approach, though preliminary, highlights the feasibility of developing software or hardware-based prefetching techniques to mitigate cache miss penalties for CMM-H.

Figure 3 also shows the throughput of CMM-H under cache-miss conditions, in which the device's DRAM cache was purged at the start of each test. In this case, the throughput remains relatively low (below 700 MB/s), although it gradually improves with increasing thread count. This observation highlights the substantial overhead associated with cache misses in CMM-H's tiered memory management. Handling a cache miss involves multiple steps, including cache-line lookups that trigger the miss, allocating new cache-lines, issuing I/O requests to fetch missing data from the underlying NAND SSD, busy-waiting until the I/O operation completes, and finally returning the requested data to the application. The significant performance impact of this process is further corroborated by comparing the measured peak throughput under cache-miss conditions (approximately 700 MB/s) to the standalone peak bandwidth of the internal NAND SSD, which is capable of achieving up to 7 GB/s for reads and 4 GB/s for writes.

Takeaway #2: *CMM-H effectively achieves high throughput under cache-hit conditions at moderate concurrency but faces significant performance degradation at higher thread counts due to cache contention and increasing miss rates. Under cache-miss scenarios, throughput drops dramatically, highlighting substantial cost and overhead from internal cache-management operations and the importance of prefetching.*

4 Software vs. Hardware Tiering

The performance characterization presented in Section 3 shows that CMM-H performs reasonably well in cache-hit scenarios but suffers significant performance degradation under cache-miss conditions due to the high hardware overhead involved in handling cache misses. This observation raises an interesting question: can the hardware-based approach still outperform software-based solutions under cache-miss conditions, given the conventional understanding that software-based methods typically incur high software overhead?

MMAP vs CMM-H: We first compare the performance of CMM-H against Linux's `mmap`, a classic OS-based tiered memory management approach. For the CMM-H experiments, each thread accessed a dedicated 512 MB memory region allocated directly on the CMM-H device. For `mmap`, we memory-mapped a 512 MB file into the application's address space. In both setups, data was accessed via `load` instructions. To accurately capture cache-miss performance, we explicitly evicted the DRAM cache within the CMM-H and dropped the Linux page cache before each measurement. We measured (sequential) read bandwidth using 4, 8, and 16 threads, all pinned to four physical CPU cores. We compared three configurations: CMM-H, `mmap`-no-prefetching, and `mmap`-with-prefetching.

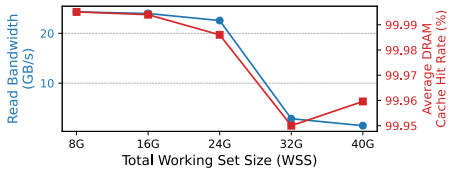


Figure 4. WSS scaling (32 threads).

Figure 5. Prefetching.

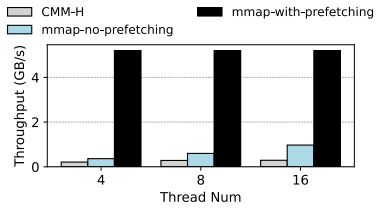
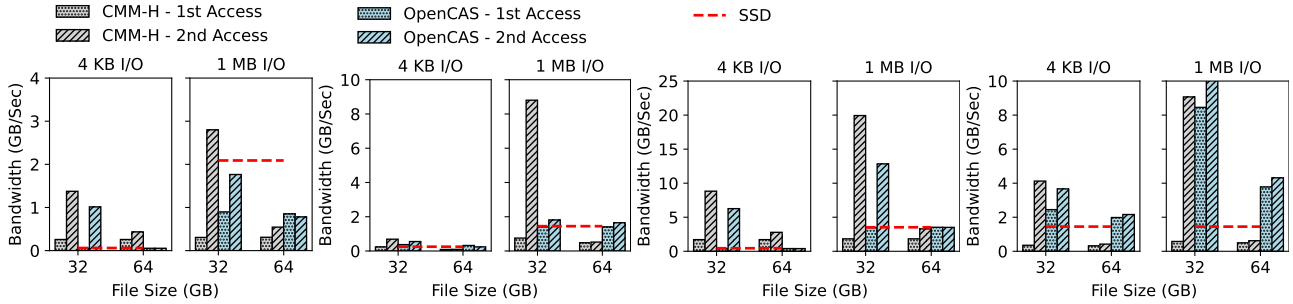


Figure 6. CMM-H vs. mmap.



(a) 1-thread random read. **(b)** 1-thread random write. **(c)** 8-thread random read. **(d)** 8-thread random write.

Figure 7. Persistent caching: CMM-H vs. OpenCAS.

Figure 6 compares the read throughput of CMM-H with that of Linux `mmap`. The CMM-H device achieves only approximately 200 MB/s of bandwidth and demonstrates *no* scalability with increasing thread count. This limitation is primarily caused by the *synchronous and blocking* memory interface provided by `cxl.mem`; upon encountering a cache miss, the application thread becomes blocked and is unable to yield the CPU to other threads, resulting in wasted CPU cycles as it waits for data promotion from the NAND SSD of CMM-H. In contrast, the *mmap*-no-prefetching configuration exhibits near-linear scalability as the thread count increases. This is because cache misses trigger major page faults, which the kernel handles *asynchronously*, enabling the CPU to execute other threads during I/O operations. In this scenario, readahead is disabled, resulting in the Linux kernel fetching data from the SSD in small 4 KB I/O units (i.e., cannot fully explore SSD's performance). Furthermore, *mmap*-with-prefetching delivers much higher bandwidth across all tested concurrency. This is due to the kernel's readahead mechanism, which proactively fetches data from the SSD in larger I/O units (up to 1 MB).

Takeaway #3: *The synchronous and blocking memory interface (via* `cxl.mem`*) of CMM-H significantly degrades performance upon cache misses, whereas the software-based approach (Linux* `mmap`*) achieves notably higher performance due to its flexible, asynchronous cache-miss handling.*

Persistent caching: In addition to the tiered memory mode evaluated above, CMM-H can also operate in persistent memory (PM) mode. Existing software-based counterparts to CMM-H primarily operate at the block storage level [2, 6, 9, 44], because they were designed to manage tiered block devices such as SSDs atop HDDs.

We compared CMM-H against Open CAS [9, 29], a state-of-the-art, block-level persistent caching system. To ensure a fair evaluation, both systems operated under the kernel file system, and we used FIO [7] to generate block-level I/O workloads. We simulated cache-hit and cache-miss scenarios using two different working set sizes (WSS): (1) a small WSS with a 32 GB file and (2) a large WSS with a 64 GB file. The workloads were executed in two consecutive rounds of random read/write operations. During the first round, all I/O operations resulted in cache misses; during the second round, if the WSS was smaller than the available cache capacity (48 GB), all accesses became cache hits, whereas if the WSS exceeded the cache size (e.g., 64 GB), all accesses remained cache misses with costly cache eviction operations.

Figure 7a shows that when the file size is smaller than the cache capacity (48 GB), OpenCAS outperforms CMM-H on read cache misses (i.e., first accesses) with large I/O sizes (e.g., 1MB). This advantage stems from OpenCAS's ability to efficiently and directly read large I/Os for its capacity device (e.g., NVMe SSD). In contrast, CMM-H fetches data from its SSD into the internal DRAM cache strictly in 4 KB pages, limiting its performance for large I/O operations. For the second access (i.e., cache hit), CMM-H surpasses OpenCAS due to its more efficient hardware-accelerated cache lookup mechanism, whereas OpenCAS experiences higher overhead from its software-based caching mechanism. Figure 7b presents a similar trend for write operations. On cache misses (i.e., first writes), OpenCAS again outperforms CMM-H. For CMM-H, handling a write cache miss requires promoting data from the NVMe SSD to its internal DRAM cache before the write can proceed, due to mismatches between the `cxl.mem` and

NVMe interfaces (*Takeaway #1*). In contrast, OpenCAS persists writes directly to the cache tier.

Takeaway #4: *Compared to software-based persistent caching, CMM-H exhibits notable performance bottlenecks on cache misses, particularly due to its fixed 4 KB data promotion granularity for large I/O operations and the additional latency introduced by RMW operations for writes caused by interface mismatches between* `cxl.mem` *and NVMe.*

5 Insights and Conclusions

This paper studies the performance of hardware-managed memory tiering in Samsung's hybrid CMM-H CXL memory device. While CMM-H offers superior cost-efficiency and a legacy byte-addressable programming interface, its performance is inferior to DRAM-based CXL memory even when the WSS fits in the internal DRAM cache. The management overhead of the DRAM cache within CMM-H inevitably lies on the critical path of data access. Moreover, our evaluation highlights key differences between hardware- and software-managed memory tiering. Although hardware-managed tiering simplifies data management across memory tiers and frees users from manually optimizing tiered memory usage, it is generally less efficient than software-managed tiering. Without application knowledge, DRAM cache misses in CMM-H can only be serviced in fixed-sized blocks (4KB pages), while software tiering promotes hot data to the performance tier in various sizes according to the size of data requests. Additionally, servicing DRAM cache misses is transparent to both the OS and user applications but occurs synchronously and lies on the critical path of data access. Together, these two factors make a high DRAM cache hit rate essential for CMM-H to achieve performance comparable to DRAM. Otherwise, its performance can fall below that of conventional NVMe-based SSDs. These insights motivate the design of applications with strong data locality relative to the size of CMM-H's DRAM cache, as well as the development of software and hardware techniques to mitigate cache miss penalties – both of which are crucial to fully unlocking the potential of byte-addressable hybrid CXL memory.

6 Acknowledgments

We thank our shepherd, Yu Hua, and the anonymous reviewers for their constructive feedback. This work was supported in part by NSF grants CCF-1845706, CNS-2415774, and CCF-2415473, and the equipment from Hewlett Packard Labs.

References

[1] Autonuma: the other approach to numa scheduling. https://lwn.net/Articles/488709/.

[2] Bcache. https://bcache.evilpiepirate.org/.

[3] Cmm-h tiered memory. https://download.semiconductor.samsung.com/resources/data-sheet/cmm-h-tm-datasheet.pdf.

[4] Compute express link™: The breakthrough cpu-to-device interconnect. https://www.computeexpresslink.org/download-the-specification.

[5] Damon-based reclamation. https://docs.kernel.org/admin-guide/mm/damon/reclaim.html#:~:text=DAMON%2Dbased%20Reclamation%20(DAMON_RECLAIM),of%20memory%20pressure%20and%20requirements.

[6] Dmcache. https://www.kernel.org/doc/Documentation/device-mapper/cache.txt.

[7] Fio. https://manpages.ubuntu.com/manpages/jammy/man1/fio.1.html.

[8] Memory-semantic ssd™. https://samsungmsl.com/cmmh/.

[9] Open CAS - open cache acceleration software. https://open-cas.github.io/index.html#who-already-uses-ocf.

[10] Samsung pm9a3 ssd review. https://www.storagereview.com/review/samsung-pm9a3-ssd-review.

[11] Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In *10th USENIX Conference on File and Storage Technologies (FAST 12)* (San Jose, CA, Feb. 2012), USENIX Association.

[12] Abulila, A., Mailthody, V. S., Qureshi, Z., Huang, J., Kim, N. S., Xiong, J., and Hwu, W.-m. Flatflash: Exploiting the byte-accessibility of ssds within a unified memory-storage hierarchy. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (2019), pp. 971–985.

[13] Agarwal, N., and Wenisch, T. F. Thermostat: Application-transparent page management for two-tiered main memory. *SIGPLAN Not. 52*, 4 (apr 2017), 631–644.

[14] Ahmadian, S., Salkhordeh, R., and Asadi, H. Lbica: A load balancer for i/o cache architectures. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2019), pp. 1196–1201.

[15] Berg, B., Berger, D. S., McAllister, S., Grosof, I., Gunasekar, S., Lu, J., Uhlar, M., Carrig, J., Beckmann, N., Harchol-Balter, M., and Ganger, G. R. The CacheLib caching engine: Design and experiences at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (Nov. 2020), USENIX Association, pp. 753–768.

[16] Bergman, S., Faldu, P., Grot, B., Vilanova, L., and Silberstein, M. Reconsidering os memory optimizations in the presence of disaggregated memory. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on Memory Management* (New York, NY, USA, 2022), ISMM 2022, Association for Computing Machinery, p. 1–14.

[17] Burnett, N. C., Bent, J., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Exploiting Gray-Box knowledge of Buffer-Cache management. In *2002 USENIX Annual Technical Conference (USENIX ATC 02)* (Monterey, CA, June 2002), USENIX Association.

[18] Chen, F., Koufaty, D. A., and Zhang, X. Hystor: Making the best use of solid state drives in high performance storage systems. In *Proceedings of the international conference on Supercomputing* (2011), pp. 22–32.

[19] Chou, C., Jaleel, A., and Qureshi, M. Batman: Techniques for maximizing system bandwidth of memory systems with stacked-dram. In *Proceedings of the International Symposium on Memory Systems* (New York, NY, USA, 2017), MEMSYS '17, Association for Computing Machinery, p. 268–280.

[20] Forney, B. C., and Arpaci-Dusseau, A. C. Storage-Aware caching: Revisiting caching for heterogeneous storage systems. In *Conference on File and Storage Technologies (FAST 02)* (Monterey, CA, Jan. 2002), USENIX Association.

[21] Guerra, J., Pucha, H., Glider, J., Belluomini, W., and Rangaswami, R. Cost effective storage using extent based dynamic tiering. In *9th USENIX Conference on File and Storage Technologies (FAST 11)* (San Jose, CA, Feb. 2011), USENIX Association.

[22] Holland, D. A., Angelino, E., Wald, G., and Seltzer, M. I. Flash caching on the storage client. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)* (San Jose, CA, June 2013), USENIX Association, pp. 127–138.

[23] Jiang, S., Ding, X., Chen, F., Tan, E., and Zhang, X. DULO: An effective buffer cache management scheme to exploit both temporal and spatial localities. In *4th USENIX Conference on File and Storage Technologies (FAST 05)* (San Francisco, CA, Dec. 2005), USENIX Association.

[24] Kim, J., Choe, W., and Ahn, J. Exploring the design space of page management for Multi-Tiered memory systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (July 2021), USENIX Association, pp. 715–728.

[25] Kim, Y., Gupta, A., Urgaonkar, B., Berman, P., and Sivasubramaniam, A. Hybridstore: A cost-efficient, high-performance storage system combining ssds and hdds. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems* (2011), pp. 227–236.

[26] Koller, R., Marmol, L., Rangaswami, R., Sundararaman, S., Talagala, N., and Zhao, M. Write policies for host-side flash caches. In *11th USENIX Conference on File and Storage Technologies (FAST 13)* (San Jose, CA, Feb. 2013), USENIX Association, pp. 45–58.

[27] Koller, R., Mashtizadeh, A. J., and Rangaswami, R. Centaur: Host-side ssd caching for storage performance control. In *2015 IEEE International Conference on Autonomic Computing* (2015), pp. 51–60.

[28] Kwon, Y., Fingler, H., Hunt, T., Peter, S., Witchel, E., and Anderson, T. Strata: A cross media file system. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), pp. 460–477.

[29] Lin, Z., Cao, L., Ahmed, F., Lu, H., and Sharma, P. When caching systems meet emerging storage devices: A case study. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems* (New York, NY, USA, 2023), HotStorage '23, Association for Computing Machinery, p. 37–43.

[30] Lin, Z., Xiang, L., Rao, J., and Lu, H. P2CACHE: Exploring tiered memory for In-Kernel file systems caching. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)* (Boston, MA, July 2023), USENIX Association, pp. 801–815.

[31] Lin, Z., Xiang, L., Rao, J., and Lu, H. P2CACHE: Exploring tiered memory for In-Kernel file systems caching. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)* (Boston, MA, July 2023), USENIX Association, pp. 801–815.

[32] Liu, J., Hadian, H., Xu, H., Berger, D. S., and Li, H. Dissecting cxl memory performance at scale: Analysis, modeling, and optimization, 2024.

[33] Liu, K., Zhang, X., Davis, K., and Jiang, S. Synergistic coupling of ssd and hard disk for qos-aware virtual memory. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (2013), pp. 24–33.

[34] Liu, R., Ma, T., Zhang, M., Huang, J., Shan, Y., Liu, Z., Xiang, L., Lin, Z., Lu, H., Rao, J., Chen, K., and Wu, Y. Dsa-2lm: A cpu-free tiered memory architecture with intel dsa. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)* (Boston, MA, July 2025), USENIX Association, pp. 1213–1222.

[35] Maruf, A., Ghosh, A., Bhimani, J., Campello, D., Rudoff, A., and Rangaswami, R. Multi-clock: Dynamic tiering for hybrid memory systems. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (Los Alamitos, CA, USA, apr 2022), IEEE Computer Society, pp. 925–937.

[36] Maruf, H. A., Wang, H., Dhanotia, A., Weiner, J., Agarwal, N., Bhattacharya, P., Petersen, C., Chowdhury, M., Kanaujia, S., and Chauhan, P. Tpp: Transparent page placement for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (New York, NY, USA, 2023), ASPLOS 2023, Association for Computing Machinery, p. 742–755.

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

[38] Papagiannis, A., Xanthakis, G., Saloustros, G., Marazakis, M., and Bilas, A. Optimizing memory-mapped {I/O} for fast storage devices. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)* (2020), pp. 813–827.

[39] Ruan, Z., Schwarzkopf, M., Aguilera, M. K., and Belay, A. {AIFM}:{High-Performance},{Application-Integrated} far memory. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (2020), pp. 315–332.

[40] Sim, J., Loh, G. H., Kim, H., OConnor, M., and Thottethodi, M. A mostly-clean dram cache for effective hit speculation and self-balancing dispatch. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture* (2012), pp. 247–257.

[41] Soundararajan, G., Prabhakaran, V., Balakrishnan, M., and Wobber, T. Extending ssd lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (USA, 2010), FAST'10, USENIX Association, p. 8.

[42] Sun, Y., Yuan, Y., Yu, Z., Kuper, R., Song, C., Huang, J., Ji, H., Agarwal, S., Lou, J., Jeong, I., Wang, R., Ahn, J. H., Xu, T., and Kim, N. S. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2023), MICRO '23, Association for Computing Machinery, p. 105–121.

[43] Wu, K., Guo, Z., Hu, G., Tu, K., Alagappan, R., Sen, R., Park, K., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus. In *19th USENIX Conference on File and Storage Technologies (FAST 21)* (2021), pp. 307–323.

[44] Wu, K., Guo, Z., Hu, G., Tu, K., Alagappan, R., Sen, R., Park, K., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus. In *19th USENIX Conference on File and Storage Technologies (FAST 21)* (2021), pp. 307–323.

[45] Wu, X., and Reddy, A. N. Exploiting concurrency to improve latency and throughput in a hybrid storage system. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (2010), pp. 14–23.

[46] Xiang, L., Lin, Z., Deng, W., Lu, H., Rao, J., Yuan, Y., and Wang, R. Nomad: non-exclusive memory tiering via transactional page migration. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation* (USA, 2024), OSDI'24, USENIX Association.

[47] Xiang, L., Zhao, X., Rao, J., Jiang, S., and Jiang, H. Characterizing the performance of intel optane persistent memory: A close look at its on-dimm buffering. In *Proceedings of the Seventeenth European Conference on Computer Systems* (2022), pp. 488–505.

[48] Yan, Z., Lustig, D., Nellans, D., and Bhattacharjee, A. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2019), ASPLOS '19, Association for Computing Machinery, p. 331–345.

[49] Yang, Q., and Ren, J. I-cash: Intelligently coupled array of ssd and hdd. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture* (2011), pp. 278–289.

[50] Yang, Y., Xiang, L., Du, P., Lin, Z., Deng, W., Wang, R., Kudryavtsev, A., Ko, L., Lu, H., and Rao, J. Architectural and system implications of cxl-enabled tiered memory. *arXiv preprint arXiv:2503.17864* (2025).

[51] Zeng, J., Pei, S., Zhang, D., Zhou, Y., Beygi, A., Yao, X., Kachare, R., Zhang, T., Li, Z., Nguyen, M., et al. Performance characterizations and usage guidelines of samsung cxl memory module hybrid prototype. *arXiv preprint arXiv:2503.22017* (2025).