Revisiting Trim for CXL Memory

Hayan Lee[†] Jungwoo Kim[‡], Wookyung Lee[†], Juhyung Park[‡], Sanghyuk Jung[◇]
Jinki Han[◇], Bryan S. Kim[*], Sungjin Lee[°], Eunji Lee[†]
[†]Soongsil University, [‡]DGIST [◇]Eeum, [*]Syracuse University, [°]POSTECH

ABSTRACT

The expansion of memory disaggregation, driven by data-centric applications, increases heterogeneity in memory systems. This shift enables the use of inexpensive, yet lifetime-limited, flash memory to be used as a memory expansion module. We argue that TRIM should be introduced into memory management systems to effectively respond to this transition. In this position paper, we explore the potential adoption of flash memory as memory expansion and present an analytical model that offers a straightforward yet rigorous evaluation of TRIM's effectiveness. Using this model and characteristics extracted from real-world workloads, we evaluate the effectiveness of TRIM in scalable memory systems and prove its necessity.

CCS CONCEPTS

• **Hardware → Memory and dense storage**; • **Information systems → Hierarchical storage management**.

KEYWORDS

CXL, Flash Memory, TRIM, Main Memory

ACM Reference Format:
Hayan Lee[†] Jungwoo Kim[‡], Wookyung Lee[†], Juhyung Park[‡], Sanghyuk Jung[◇] and Jinki Han[◇], Bryan S. Kim[*], Sungjin Lee[°], Eunji Lee[†], [†]Soongsil University, [‡]DGIST [◇]Eeum, [*]Syracuse University, [°]POSTECH, . 2025. Revisiting Trim for CXL Memory. In *17th ACM Workshop on Hot Topics in Storage and File Systems(HotStorage '25), July 10–11, 2025, Boston, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3736548.3737825

1 INTRODUCTION

The rapid advancement of AI, fueled by data-intensive applications and large-scale deep learning, is driving a surge in demand for memory capacity. In particular, as the performance of lightweight LLM models specialized for specific domains continues to improve, interest in AI technologies is reaching new heights [8, 24]. Against this backdrop, data centers are increasingly facing the challenge of meeting high memory capacity requirements for AI model serving [44].

Recently emerging cache-coherent interconnect technologies, such as Compute Express Link (CXL) [11], GEN-Z [4], CCIX [1], and OpenCAPI [2], have been introduced to mitigate this memory wall problem [22]. Among them, CXL stands out as a technology that not only enables the consolidation of stranded memory into high-capacity memory pools but also reduces data movement between processing components, thereby conserving memory bandwidth. While the CXL market remains in its early stages, it is gaining significant traction in both industry and academia [10, 13, 14, 23, 25, 29, 33, 34, 42, 45]. The CXL specification has advanced to version 3.2 [11], and major processor and accelerator vendors, including NVIDIA and ARM, have started incorporating CXL support [40]. Meanwhile, memory vendors are also rolling out devices that support CXL 1.1 and 2.0 [9].

This paper revisits the necessity of the TRIM command in environments where flash memory serves as CXL memory, which we refer to as CXL-Flash hereinafter. TRIM was originally introduced to mitigate write amplification when flash memory is used as block storage [41]. Specifically, it is designed to notify flash memory of data invalidation upon file deletion, preventing unnecessary relocation overhead during garbage collection (GC). A range of studies demonstrate that the proper use of TRIM significantly reduces write amplification and boosts performance [17, 30–32, 35]. For instance, ZNSwap reduces the write amplification factor (WAF) by invalidating unused swap slots [17], while iTRIM improves performance in mobile environments by issuing non-intrusive TRIM commands on obsolete data in COW file systems [35]. Due to its intuitive yet effective nature, TRIM is supported by various file systems, including Ext4 [7], Btrfs [5], and F2FS [6].

When flash memory is repurposed as a memory expander, it encounters the same underlying challenge. CXL-Flash is regarded as a promising architecture for enabling scalable memory disaggregation with CXL [25, 29, 45–47] and

HotStorage '25, July 10–11, 2025, Boston, MA, USA

ACM ISBN 979-8-4007-1947-9/2025/07.
https://doi.org/10.1145/3736548.3737825

has been prototyped as a Type 3 device for memory expansion [10]. In this regard, we argue that the memory management system in the kernel should be equipped with a TRIM-like mechanism to accommodate high-capacity but wear-prone memory. As it stands, memory management systems operate under the assumption that DRAM retains invalid data without incurring additional costs. Thus, even when unused memory is released, it is merely added to the free list, and the host does not notify the underlying device of its invalidity. This causes the underlying device to regard all written areas as valid unless explicitly overwritten, potentially leading to severe write amplification in flash memory.

For instance, data centers often overprovision memory capacity to elastically scale multi-tenant workloads. A study analyzing virtual machine (VM) traces published by Microsoft Azure [3] found that, due to the dynamic execution and termination of VMs, only 50% of the reserved memory capacity for active VMs is utilized on average [28]. This implies that half of the data is falsely marked as valid, causing unnecessary copying during GC in flash.

In this paper, we explore incorporating TRIM into the memory management system to mitigate write amplification, which accelerates memory wear and degrades performance in endurance-limited memory. We make the following contributions with this work:

- We present a TRIM command designed for CXL interconnect protocols.
- We present a near-exact analytical model to evaluate the effectiveness of TRIM, derived from the existing SSD write amplification model [21].
- We analyze the impact of TRIM in large-scale memory environments and demonstrate its necessity.

2 CXL-ENABLED FLASH MEMORY

CXL. CXL is an open industry standard interconnect that has gained popularity due to its adoption of Peripheral Component Interconnect Express (PCIe). Specifically, it enables direct load/store memory accesses between CPUs and end-points with low latency, which opens up the possibility to expand host memory with a variety of memory devices. Therefore, most prior works focus on utilizing CXL for memory expansion and memory disaggregation [10, 13, 14, 23, 25, 29, 33, 34, 42, 45]. The CXL protocol consists of three main sub-protocols: CXL.io, CXL.cache, and CXL.mem. The CXL.io protocol serves as the foundational communication protocol, functioning similarly to PCIe. It is responsible for link initialization, device discovery, and general management of CXL devices. The CXL.cache protocol enables coherent access from attached devices to the processor's memory, reducing the need for software-managed coherence. The CXL.mem protocol allows the host and devices to coherently access device-attached memory, supporting memory pooling, dynamic allocation, and efficient resource utilization.

CXL-Flash. Given its cost and capacity advantages over DRAM, the feasibility of integrating flash memory into CXL as a memory pool is being actively explored [9, 29, 45–47]. Jung advocates CXL-Flash as a viable approach to cost-effectively implementing large-scale working memory [29]. While Non-Volatile Memory Express (NVMe) allows byte access to PCIe storage buffers, its lack of cache coherence leads to performance degradation [15, 16]. In contrast, CXL enables cache-coherent access, making PCIe storage viable as a memory expander. The author demonstrates, using a CXL-Flash prototype, that locality-aware access patterns can hide flash latency via large internal DRAM. Yang et al. [45] explore the feasibility of using flash memory for large-scale CXL memory pooling but identify challenges such as I/O amplification, high latency, and limited endurance. They show that request merging, prefetching, and caching can mitigate these issues, with sufficient internal caching enabling a multi-year lifespan for CXL-Flash.

The potential of CXL-Flash is also evident in industry, exemplified by CMM-H [9], the first memory-semantic CXL-enabled SSD. As a dual-mode CXL Type 3 device, it supports block-level I/O (CXL.io) and cache-line-level I/O (CXL.mem), integrating storage and memory for flexible workloads. CMM-H includes 16GB of DRAM and 2TB of flash, with memory accessed via CXL.mem and the SSD via CXL.io, and exposes Host-managed Device Memory (HDM), mapped 1:1 to NVMe storage, while leveraging the built-in DRAM as a cache for NAND. For reliability, CMM-H supports crash persistence via CXL 2.0 Global Persistent Flush (GPF) and an external battery-backed DRAM cache. With the introduction of CMM-H, a file system designed to efficiently leverage its unique performance characteristics has recently been proposed [46, 47].

Given recent advances and growing attention, CXL-Flash-based memory expansion is highly viable. Exploring the adoption of the TRIM command in memory management systems to enhance the lifespan and performance of CXL-Flash is both timely and necessary.

3 TRIM FOR CXL-FLASH

Design. This section explores how the TRIM command can be integrated into CXL. We suggest that extending CXL.io to implement the TRIM command as an asynchronous operation would be appropriate. Unlike load/store instructions, which must run quickly without kernel intervention, TRIM is infrequent and does not require synchronous execution. For security and system stability, it should be a privileged operation managed by the operating systems (OS) rather

Table 1: Notations used in this article.

Notation	*Description*
N_p	number of per-block pages
T, U	total and user-visible blocks
α	over-provisioning ratio, defined as T/U
D_r	ratio of discard traffic to write traffic
D_c	discarded memory size
$A(\alpha)$	write amplification

than a user-level instruction. CXL.io can be considered a protocol that meets these requirements well.

There are two potential layers from which a TRIM command can be initiated. First, the TRIM command can be managed at the memory management system of the OS (e.g., the buddy system in the Linux kernel). In this case, when a user application releases memory or terminates, the OS memory management system can use the TRIM command to invalidate the reclaimed memory range. An alternative approach is to provide a system call that allows applications to invoke TRIM without returning memory to the OS. As kernel-mediated memory allocation for every user application request incurs non-negligible context switch overhead, various middleware, including memory allocators [27] and the JVM [20], often overprovision memory in user space. Thus, when large portions remain unused, issuing a TRIM command can be an effective optimization. Although not covered in this paper, identifying the optimal initiator layer for TRIM in CXL memory and evaluating the impact of TRIM granularity and frequency will be necessary for optimizing memory management in CXL systems.

A key pitfall in integrating TRIM into memory management is the risk of data loss from improper synchronization. Since TRIM requests via CXL.io are delayed, the OS may reallocate the trimmed region before the TRIM completes. If valid data is written via CXL.mem during this window, it may be invalidated by the delayed TRIM. Proper coordination can eliminate this risk: the device must notify the host upon TRIM completion, and the OS must defer reallocation until the acknowledgment is received.

Analytical model. To evaluate the effectiveness of TRIM in CXL-Flash, an appropriate assessment methodology is essential. Write amplification in flash memory has been a longstanding challenge, prompting extensive research into leveraging TRIM (or similar interfaces) to mitigate it [26, 30, 32, 43]. Most prior studies assess the impact of TRIM using SSD simulators/emulators [32, 43] or TRIM-compliant commercial SSDs [26, 30].

However, these approaches are not well suited for CXL-Flash, where flash memory functions as main memory. First, memory accesses occur orders of magnitude more frequently than storage accesses, making trace collection and simulation virtually impossible. For example, on a 3GHz processor with a 95% LLC hit rate and a memory access per instruction (MAPI) of 0.3, an estimated 30 million memory accesses per second would be generated. As a result, capturing traces over a meaningful duration would produce an overwhelming volume of data, rendering simulation impractical.

Furthermore, the limited availability of CXL-compliant CPUs and memory devices [9, 37, 42] makes testing on real systems challenging. In particular, commercial CXL-based flash devices do not provide public access to their firmware, making it infeasible to control or profile TRIM-related flash memory operations. Alternatively, one can employ NUMA emulation [34] or a CXL-compatible software emulator [12, 36] for CXL testing. However, the former does not provide a CXL-Flash environment, while the latter is limited by functionality or slow execution, making large-scale memory evaluation infeasible in practice.

To overcome this challenge, we present the first analytical model for write amplification in CXL-Flash with TRIM enabled. Table 1 summarizes the variables used in our analytical model. Our proposed model extends an existing analytical model of SSD write amplification [21] to incorporate the effect of TRIM, where the original model is defined as follows.

$$A = \frac{N_p}{N_p - E(v)} = \frac{1}{1 - \left(1 - \frac{1}{UN_p}\right)^{\frac{TN_p}{A}}} \tag{1}$$

This model assumes uniformly distributed random traffic and a garbage collection (GC) policy that selects the least recently written (LRW) block first for cleaning. It expresses write amplification (A) in terms of the number of pages per block (N_p) and the expected number of valid pages remaining in a victim block ($E(V)$). E(V) can be calculated based on the probability of a page remaining valid using the user-visible pages (UN_p) and physical pages (TN_p), as shown in Equation (1). A page remains valid if it is not overwritten from the time it is written until its block is selected as a victim. The probability that a given page is not overwritten by a single write is $1 - \frac{1}{UN_p}$. Assuming one page is written per time unit, and physical pages are consumed at a rate of A, this condition must hold for $\frac{TN_p}{A}$ time units. Finally, Equation (2) is derived from Equation (1) using the Lambert's W function.

$$A = \frac{\alpha}{\alpha + W\left(-\alpha e^{-\alpha}\right)} \tag{2}$$

We model the change in the probability of a page remaining valid in a victim block when TRIM is enabled. To this end, we define D_r, which represents the ratio of discard traffic to write traffic. Since a page becomes invalid when a TRIM operation occurs, the probability of a page being invalidated at each time step changes from $\frac{1}{UN_p}$ to $\frac{1}{UN_p} + \frac{D_r}{UN_p}$. Therefore, when TRIM is enabled, write amplification can be expressed by the following equation:

$$A = \frac{1}{1 - \left(1 - \frac{1+D_r}{UN_p}\right)^{UN_p t}}, \tag{3}$$

where $T = \alpha U$ and $t = \frac{\alpha}{A}$.

Considering $U \cdot N_p \to \infty$, the term $\left(1 - \frac{1+D_r}{UN_p}\right)^{UN_p t}$ converges to $e^{-(1+D_r)t}$, simplifying Equation (3) into:

$$A = \frac{1}{1 - e^{-(1+D_r)t}} = \frac{\alpha}{t}. \tag{4}$$

The closed-form solution to Equation (4) can be expressed using Lambert's W function, defined by $W(z)e^{W(z)} = z$:

$$t = \alpha + \frac{W(-\alpha(D_r+1)e^{-\alpha(D_r+1)})}{D_r+1}, \quad A = \frac{\alpha}{t}. \tag{5}$$

Note that TRIM has the effect of increasing the spare area by the discarded space size D_c, which effectively increases the over-provisioning ratio α. Taking this into account, the equation becomes:

$$A = \frac{\alpha'}{\alpha' + \frac{W(-\alpha'(D_r+1)e^{-\alpha'(D_r+1)})}{D_r+1}}. \tag{6}$$

where $T = \alpha'(U - D_c)$.

Validation. To validate the accuracy of the proposed analytical model, we implement TRIM in `ftlsim` [21], an SSD simulator used for validating the original analytical model, and measure write amplification using a synthetic workload. Note that the proposed analytical model for TRIM is also applicable to SSDs. The total number of logical blocks is set to 10^6, with each block containing 128 pages, and each page having a size of 16KB. The testing scenario consists of two phases. First, a warm-up phase is conducted, where all logical pages are sequentially written to initialize the system. Then, uniformly distributed writes are performed on one-third of the total capacity. Throughout this process, TRIM commands corresponding to D_r are issued every 10 writes.

Fig. 1 compares the write amplification predicted by the analytical model with the simulated results in `ftlsim`. Overall, the model's predictions align closely with the simulated results, with an average error of 2.3% and a maximum error within 6.9%. As shown in the figure, TRIM significantly contributes to reducing write amplification, with its effect becoming more pronounced as the spare factor decreases. When the spare factor is 0.10, TRIM reduces WAF by half compared to the case without TRIM (D_r = 0.0) when TRIM traffic accounts for 10% of writes (D_r = 0.1). Additionally, when TRIM traffic reaches 30% of writes, WAF decreases by 67%. For spare factors of 0.20 and 0.30, performing TRIM reduces WAF by 25–47% and 16–33%, respectively.

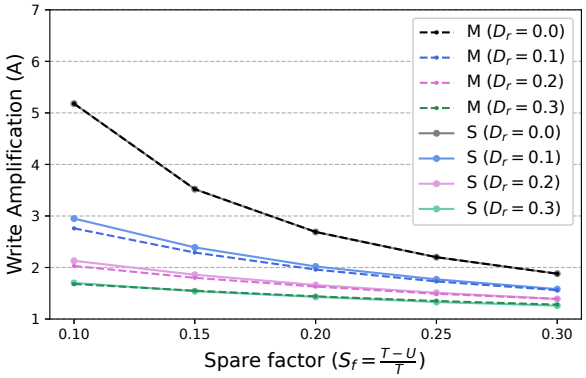


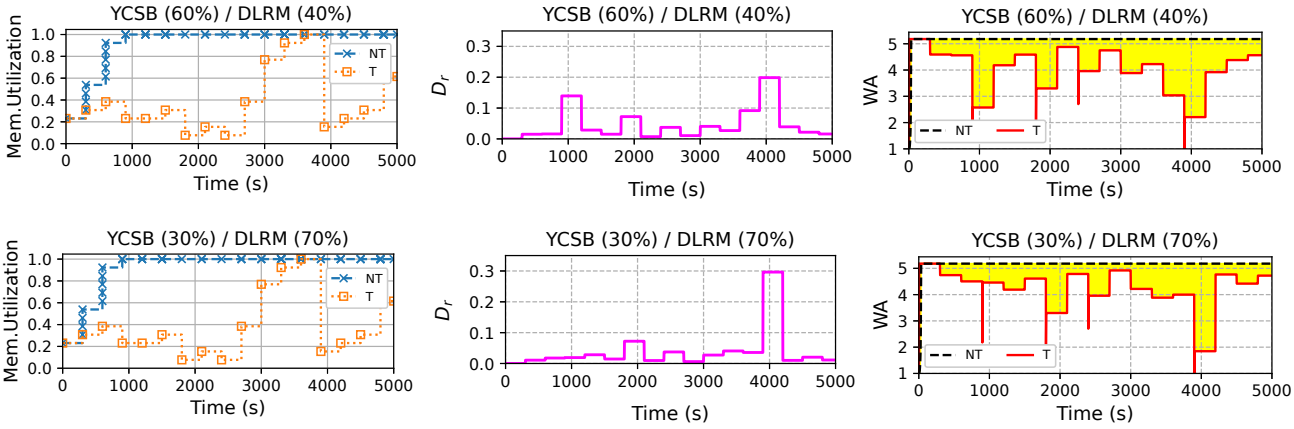
Figure 1: Write amplification as a function of spare factor. M and S denote analytical model and simulation results, respectively.

4 TRIM EFFECTS ON CXL-FLASH

This section explores the effect of TRIM in a large-scale memory system using the proposed analytical model. A key scenario where high-capacity CXL-Flash proves particularly valuable is a data center hosting numerous virtual machines. As a case study, we assess the impact of TRIM in such a dynamic environment by leveraging VM traces publicly released by Microsoft Azure [3], which provide details on each VM's start time, runtime, and memory usage. We compare the write amplification in two cases: when memory released upon VM termination is trimmed and when it is not. The proposed analytical model relies on D_r, the ratio of discarded traffic to write traffic. However, the publicly available Azure trace lacks data on the write traffic generated by each VM. To bridge this gap, we profile memory write traffic using representative workloads: YCSB (Yahoo Cloud Service Benchmark) suites [19] for key-value stores and DLRM (Deep Learning Recommendation Model) benchmark for machine learning applications [38]. For YCSB, we use Redis [39] to execute workloads A–F, each generating one million operations. For DLRM, we run training and inference using the Criteo Kaggle Display Advertising Dataset [18]. Memory access traffic is captured using the perf tool, which extracts `LLC_misses.mem_write` event values. Table 2 summarizes the memory access traffic profile of the studied workloads. Using this information, we define D_r, the ratio of discarded

Table 2: Memory access traffic (MB/s). A–F are YCSB workloads, and TR/IF are DLRM Training/Inference.

WK	A	B	C	D	E	F	TR	IF
R	262	296	226	524	79	190	1200	380
W	147	154	94	388	40	101	500	267



(a) Memory utilization **(b) Discard-to-write traffic ratio** **(c) Write amplification**

Figure 2: TRIM effect on CXL-Flash running VMs over time.

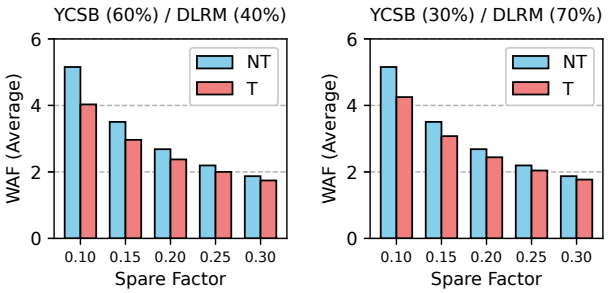


Figure 3: Average write amplification in CXL-Flash running VMs.

size to write traffic, for each VM as:

$$D_r = \frac{\text{memory usage}}{\text{load-phase write traffic} + \text{runtime write traffic}}$$

Here, *runtime write traffic* is given by the product of the VM's execution time and its per-second write traffic. *Load-phase write traffic*, generated when the VM is initially loaded, is conservatively assumed to match memory usage, though the actual load traffic may be smaller. The *discarded size* corresponds to the memory occupied by the VM at termination, which is also equivalent to memory usage. We construct two scenarios by mapping the extracted workloads onto 5,000 Azure VMs at different ratios. In the first scenario, we run VMs with YCSB and DLRM at a 6:4 ratio, and in the second, they are allocated at a 3:7 ratio. All VMs running YCSB have workloads A–F evenly distributed across them. For DLRM, half of the VMs run training and the other half run inference.

Fig. 2 illustrates the effect of TRIM over time in CXL-Flash, where VMs run different workloads. Specifically, they compare the changes in memory utilization (Fig. 2a), discard-to-write traffic ratio (Fig. 2b), and write amplification (Fig. 2c) when TRIM is enabled (denoted as T) versus when it is disabled (denoted as NT). First, memory utilization refers to the ratio of the valid data in CXL-Flash relative to its user-visible capacity. In this study, we assume that the size of CXL-Flash matches the maximum active memory used by the VMs simultaneously (22.75 GB), and that the internal spare factor of CXL-Flash is set to 0.1. When TRIM is disabled, memory is allocated each time a VM is launched but not invalidated upon termination. As a result, at a certain point, CXL-Flash perceives all available space as fully utilized. In contrast, when TRIM is enabled, memory released upon VM termination is properly invalidated. This behavior increases the available spare area in CXL-Flash, significantly reducing write amplification. As shown in Fig. 2c, when TRIM is disabled, the WAF exceeds 5 beyond a certain point. However, with TRIM enabled, WAF drops significantly during periods when a large amount of memory is reclaimed from terminated VMs. Additionally, Fig. 2b shows how the discard-to-write traffic ratio (D_r) evolves dynamically as VMs start and terminate. This ratio depends on the memory size occupied by VMs and the write traffic they generate, with higher values leading to greater reductions in WAF.

Fig. 3 compares the average WAF over the total VM execution time (5,000 seconds) with and without TRIM enabled. In both workloads, enabling TRIM results in a reduction in WAF, with an average decrease of 11.56% and up to 19.69% at its maximum. Notably, the impact of TRIM on WAF reduction is more pronounced when the spare factor is smaller. With a 10% spare factor, enabling TRIM reduces WAF by 19.69%, 3.2x greater than the 6.14% reduction at 30% spare factor. By invalidating free memory, TRIM effectively increases the spare factor, which yields greater benefit when the internal spare area is limited. These findings suggest that TRIM can contribute significantly to building a cost-effective memory expander.

5 CONCLUSION

In this paper, we explored the feasibility of CXL-Flash as a scalable yet wear-prone memory expander and argued for the necessity of TRIM in memory management. To support this, we proposed a near-accurate analytical model to estimate TRIM's effect on write amplification and quantitatively demonstrated its impact in large-scale working memory under real-world workloads. Future work can extend this research in several directions: (1) Optimize TRIM for memory management systems by tuning its frequency, granularity, and the ideal host-side component to issue it, while also exploring extensive uses such as memory initialization. (2) Investigate TRIM's broader impact on performance and lifespan beyond write amplification to fully leverage CXL-Flash. (3) Extend the analytical model to diverse workloads and cleaning policies to clarify real-world effectiveness.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments. This work is funded in part by FADU Inc., the National Research Foundation (NRF) of Korea (RS-2023-00253005) and the MSIT (Ministry of Science and ICT) of Korea, under the Convergence security core talent training business support program (IITP-2024(2025)-RS-2024-00426853) supervised by the Institute of Information & Communications Technology Planning & Evaluation (IITP). Eunji Lee is the corresponding author.

REFERENCES

[1] 2016. *CCIX Consortium.* https://www.ccixconsortium.com/.
[2] 2016. *OpenCapi Consortium.* https://opencapi.org/.
[3] 2017. *AzurePublicDataset.* https://github.com/Azure/AzurePublicDataset.git.
[4] 2018. *Gen-Z Consortium.* https://genzconsortium.org/specifications/.
[5] 2020. *BTRFS documentation - TRIM/DISCARD.* https://btrfs.readthedocs.io/en/latest/Trim.html.
[6] 2020. *https://www.kernel.org/doc/Documentation/filesystems/f2fs.txt.* https://www.kernel.org/doc/Documentation/filesystems/f2fs.txt.
[7] 2022. *ext4(5) — Linux manual page.* https://man7.org/linux/man-pages/man5/ext4.5.html.
[8] 2023. *Grok.* https://x.ai/.
[9] 2023. *Samsung CXL Solutions - CMM-H.* https://semiconductor.samsung.com/emea/news-events/tech-blog/samsung-cxl-solutions-cmm-h/.
[10] 2024. *Compute Express Link (CXL) for Data-Centric Computing.* https://www.nvidia.com/en-us/on-demand/session/gtc24-s63041/.
[11] 2024. *Compute Express Link Specification Revision 3.2.* https://www.computeexpresslink.org/.
[12] 2024. *OpenCIS.* https://www.opencis.io/.
[13] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. 2022. Enabling CXL memory expansion for in-memory database management systems. In *Proceedings of the 18th International Workshop on Data Management on New Hardware*. 1–5.
[14] Moiz Arif, Kevin Assogba, M Mustafa Rafique, and Sudharshan Vazhkudai. 2022. Exploiting cxl-based memory for distributed deep learning. In *Proceedings of the 51st International Conference on Parallel Processing*. 1–11.
[15] Duck-Ho Bae, Insoon Jo, Youra Adel Choi, Joo-Young Hwang, Sangyeun Cho, Dong-Gi Lee, and Jaeheon Jeong. 2018. 2B-SSD: The case for dual, byte-and block-addressable solid-state drives. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 425–438.
[16] Stephan Bates. 2016. Enabling Remote Access to Persistent Memory on an IO Subsystem using NVM Express and RDMA. https://www.snia.org/sites/default/files/SDC/2016/presentations/persistent_memory/Stephen_Bates_Enabling_Remote_Access_Persistent_Memory_IO_Subsystem_using_NVM_Express_RDMA.pdf.
[17] Shai Bergman, Niklas Cassel, Matias Bjørling, and Mark Silberstein. 2023. ZNSwap: Un-block your swap. *ACM Transactions on Storage* 19, 2 (2023), 1–25.
[18] Olivier Chapelle. 2014. *Display Advertising Challenge.* https://www.kaggle.com/competitions/criteo-display-ad-challenge.
[19] 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*. 143–154.
[20] Oracle Corporation and OpenJDK Contributors. 2024. Memory Management in the Java Virtual Machine. https://openjdk.org/
[21] Peter Desnoyers. 2012. Analytic modeling of SSD write performance. In *Proceedings of the 5th Annual International Systems and Storage Conference*. 1–10.
[22] Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. 2024. AI and memory wall. *IEEE Micro* (2024).
[23] Donghyun Gouk, Miryeong Kwon, Hanyeoreum Bae, Sangwon Lee, and Myoungsoo Jung. 2023. Memory pooling with cxl. *IEEE Micro* 43, 2 (2023), 48–57.
[24] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).
[25] Weizhou Huang, Jian Zhou, Meng Wang, You Zhou, Xiaoyi Zhang, Feng Zhu, Shu Li, Kun Wang, and Fei Wu. 2024. TieredHM: Hotspot-Optimized Hash Indexing for Memory-Semantic SSD-Based Hybrid Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 6 (2024), 1755–1768.
[26] Choulseung Hyun, Jongmoo Choi, Donghee Lee, and Sam H Noh. 2011. To TRIM or not to TRIM: Judicious triming for solid state drives. In *Poster presentation in the 23rd ACM Symposium on Operating Systems Principles*.
[27] Jason Evans. 2011. jemalloc: A General Purpose Memory Allocator. https://github.com/jemalloc/jemalloc.
[28] Wenjing Jin, Wonsuk Jang, Haneul Park, Jongsung Lee, Soosung Kim, and Jae W Lee. 2023. Dram translation layer: Software-transparent dram power savings for disaggregated memory. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.
[29] Myoungsoo Jung. 2022. Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd). In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*. 45–51.
[30] Myoungsoo Jung and Mahmut Kandemir. 2013. Revisiting widely held SSD expectations and rethinking system-level implications. *ACM SIGMETRICS Performance Evaluation Review* 41, 1 (2013), 203–216.
[31] Byungjo Kim, Dong Hyun Kang, Changwoo Min, and Young Ik Eom. 2014. Understanding implications of trim, discard, and background command for emmc storage device. In *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*. IEEE, 709–710.

[32] Joohyun Kim, Haesung Kim, Seongjin Lee, and Youjip Won. 2010. FTL design for TRIM command. In *The Fifth International Workshop on Software Support for Portable Storage*. 7–12.

[33] Miryeong Kwon, Sangwon Lee, and Myoungsoo Jung. 2023. Cache in hand: Expander-driven cxl prefetcher for next generation cxl-ssd. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*. 24–30.

[34] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 2023. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 574–587.

[35] Yu Liang, Cheng Ji, Chenchen Fu, Rachata Ausavarungnirun, Qiao Li, Riwei Pan, Siyu Chen, Liang Shi, Tei-Wei Kuo, and Chun Jason Xue. 2020. iTRIM: I/O-aware TRIM for improving user experience on mobile devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 9 (2020), 1782–1795.

[36] Haifeng Liu, Long Zheng, Yu Huang, Jingyi Zhou, Chaoqiang Liu, Runze Wang, Xiaofei Liaot, Hai Jinf, and Jingling Xue. 2024. Enabling efficient large recommendation model training with near cxl memory processing. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 382–395.

[37] Nevine Nassif, Ashley O Munch, Carleton L Molnar, Gerald Pasdast, Sitaraman V Lyer, Zibing Yang, Oscar Mendoza, Mark Huddart, Srikrishnan Venkataraman, Sireesha Kandula, et al. 2022. Sapphire rapids: The next-generation intel xeon scalable processor. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 44–46.

[38] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[39] Redis Labs. 2011. Redis. https://redis.io/.

[40] Paul Schell Reece Hayden. 2024. OPPORTUNITIES AND CHALLENGES FOR COMPUTE EXPRESS LINK (CXL). *ABI Research* (2024).

[41] Frank Shu. 2007. Notification of deleted data proposal for ata8-acs2. *Microsoft Corporation, dated Apr* 21 (2007).

[42] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. 2023. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 105–121.

[43] Zev Weiss, Sriram Subramanian, Swaminathan Sundararaman, Nisha Talagala, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2015. ANViL: Advanced virtualization for modern Non-Volatile memory devices. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. 111–118.

[44] Yi Xu, Ziming Mao, Xiangxi Mo, Shu Liu, and Ion Stoica. 2024. Pie: Pooling CPU Memory for LLM Inference. *arXiv preprint arXiv:2411.09317* (2024).

[45] Shao-Peng Yang, Minjae Kim, Sanghyun Nam, Juhyung Park, Jin-yong Choi, Eyee Hyun Nam, Eunji Lee, Sungjin Lee, and Bryan S Kim. 2023. Overcoming the memory wall with CXL-Enabled SSDs. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 601–617.

[46] Seung Won Yoo, Joontaek Oh, Myeongin Cheon, Bonmoo Koo, Wonseb Jeong, Hyunsub Song, Hyeonho Song, Donghun Lee, and Youjip Won. 2025. DJFS : Directory-Granularity Filesystem Journaling for CMM-H SSDs. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. USENIX Association, Santa Clara, CA, 35–51. https://www.usenix.org/conference/fast25/presentation/yoo

[47] Yekang Zhan, Haichuan Hu, Xiangrui Yang, Shaohua Wang, Qiang Cao, Hong Jiang, and Jie Yao. 2024. RomeFS: A CXL-SSD Aware File System Exploiting Synergy of Memory-Block Dual Paths. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 720–736.