USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

DSA-2LM: A CPU-Free Tiered Memory Architecture with Intel DSA

Ruili Liu, *Tsinghua University and University of Electronic Science and Technology of China;* Teng Ma, *Alibaba Group;* Mingxing Zhang, Jialiang Huang, and Yingdi Shan, *Tsinghua University;* Zheng Liu, *Alibaba Group;* Lingfeng Xiang, Zhen Lin, Hui Lu, and Jia Rao, *The University of Texas at Arlington;* Kang Chen and Yongwei Wu, *Tsinghua University*

https://www.usenix.org/conference/atc25/presentation/liu-ruili

This paper is included in the Proceedings of the 2025 USENIX Annual Technical Conference.

July 7–9, 2025 • Boston, MA, USA

ISBN 978-1-939133-48-9

Open access to the Proceedings of the 2025 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

DSA-2LM: A CPU-Free Tiered Memory Architecture with Intel DSA

Ruili Liu[♠◇*], Teng Ma[♡*+], Mingxing Zhang[♠+], Jialiang Huang[♠♡], Yingdi Shan[♠], Zheng Liu[♡]
Lingfeng Xiang[♣], Zhen Lin[♣], Hui Lu[♣], Jia Rao[♣], Kang Chen[♠], Yongwei Wu[♠]
[♠]Tsinghua University, [♡]Alibaba Group
[♣]The University of Texas at Arlington, [◇]University of Electronic Science and Technology of China

Abstract

Tiered Memory is critical to manage heterogeneous memory devices, such as Persistent Memory or CXL Memory. Existing works make difficult trade-offs between optimal data placement and costly data movement. With the advent of Intel Data Streaming Accelerator (DSA), a CPU-free hardware to move data between memory regions, data movement can be up to 4× faster than a single CPU core. However, the fine memory movement granularity in Linux kernel undermines the potential performance improvement. To this end, we have developed DSA-2LM, a new tiered memory system that adaptively integrates DSA into page migration. The proposed framework integrates fast memory migration workflow and adaptable concurrent data paths with well-tuned DSA configurations. Experimental results show that, compared to three representative tiered memory works: MEMTIS, TPP and NOMAD, DSA-2LM can achieve 20%, 30% and 16% performance improvement under real-world applications.

1 Introduction

Data-intensive applications such as data processing, machine learning and graph processing are experiencing rising memory demands. It couples with the increasing cost of DRAM and limited DRAM capacity have made memory a major expense. Non-DRAM memory devices, such as Compute Express Link (CXL) [33, 40, 50]/Non-volatile Memory (NVM) [37], appear as a larger memory tier with a low per GB price. However, these technologies expose higher latencies than main memory, potentially degrading performance if data is sub-optimally placed in memory hierarchy.

To place hot/cold data in the correct fast/slow tier, the architecture of tiered memory involves three primary stages: detecting hot data, selecting data to be migrated, and executing the data migration. NOMAD [44] presents that a significant amount of time is consumed by page copying during the memory migration process. Despite the implementation of MEMTIS [29], which effectively mitigates page copying using histogram algorithm [29], the measured migration time spent on copying remains around 49%. One solution is to enhance the accuracy of the data hotness detection. Nonetheless, achieving high precision is associated with substantial CPU overhead. Even when tasks are offloaded to hardware solutions such as processor event-based sampling (Intel PEBS [48]), the performance degradation is still recorded at 20% [44]. This is attributed to the increased frequency of migration, which consumes CPU resources (11% CPU overhead in Graph500 [34]) and occupies additional bandwidth, interfering with applications, negatively impacting overall end-to-end performance.

A promising way to reduce CPU bottleneck is to offload overhead to hardware accelerators such as the Intel Data Streaming Accelerator (DSA). DSA can transfer data between memory regions without involving CPU and now ships as a standard feature in 4/5th generation Intel Xeon processors [24]. Its benefits are twofold. First, each page-copy request requires submitting only a small descriptor to work queue (WQ), which takes only a few cycles to complete [10]. This eliminates most of the CPU overhead, even in systems that rely on CXL and CPU interactions. Second, DSA provides a bandwidth of about 32 GB/s per device, comparable to four CPU cores [19]. Modern servers typically include four DSA devices per socket, delivering a combined bandwidth of up to 110 GB/s. Hence, DSA is well-suited for tiered memory.

However, using DSA effectively requires careful design. First, current kernel support for DSA relies on the traditional Direct Memory Access (DMA) interface, introducing large overhead for buffer preparation and (un)mapping. It is inadequate for fine-grained page movement. Second, DSA outperforms CPU-based copying only when there is sufficient concurrency to hide high invocation latency, so asynchronous task partitioning and dispatching must be carefully planned to maximize performance by utilizing the multiple cores provided by the CPU socket. Finally, DSA performance is sensitive to many configurations and requires tuning. We address this by developing an adaptive algorithm for different page types, with dynamic batch sizes and WQ settings.

Based on DSA hardware, we design DSA-2LM (DSA-based two-level memory), a new tiered memory system in Linux kernel with flexible interfaces (§3.2). While inheriting hotness detection and page migration logic via `kmigrated` of MEMTIS [29], DSA-2LM introduces several DSA-specific designs. These designs contain fast memory migrating workflow (§3.3) and adaptable concurrent data migration (§3.4) with well-tuned batch size and number of WQs. It avoids

*: Equal Contributions. +: Corresponding Authors. This work is supported by National Key Research & Development Program of China (2022YFB4502004), Natural Science Foundation of China (62141216, 92467102), Tsinghua University Initiative Scientific Research Program, and Alibaba Group through Alibaba Innovative Research Program.

CPU involvement in the critical data copying path, enabling aggressive CPU-free migrations.

To evaluate the performance of DSA-2LM framework, we choose five widely used applications and use traces of industry workloads. The evaluation environment is a commercial pre-market CXL system. The DSA-based acceleration of page copying allows the system to take full advantage of page migration, thereby improving its responsiveness to changes in memory access patterns. Experimental results show that, compared to three representative page management schemes: MEMTIS [29], TPP [33] and NOMAD [36], DSA-2LM achieves up to 81.7%, 52.9% and 15.6% performance improvement under real-world applications. DSA-2LM is available at https://github.com/madsys-dev/DSA-2LM.

2 Motivation and Challenges

2.1 Tiered Memory and Applications

The memory demand of data-intensive applications such as graph processing and machine learning continues to increase [28]. In Microsoft Azure, memory costs ∼50% of the total server costs [30]. ML models are rapidly growing, and are expected to grow 50× in the next five years [33]. With rapid data growth, memory hierarchy should be redesigned.

Tiered memory fulfills the requirements of data-intensive applications. It uses a two-level hierarchy with a fast, expensive tier and a slow, cost-effective tier [15]. It frequently accesses data in the fast tier, moves less-accessed data to slow tier, and maintains an exclusive data placement. Emerging memory technologies such as high bandwidth memory, CXL memory [8, 32, 40, 43, 51], NVM [47], and NVMe SSDs introduce solutions that challenge the traditional memory hierarchy. For example, CXL [8] integrates various memory devices under a single interface, offering larger storage capacities and narrowing the performance gap. It guided researchers to explore state-of-the-art systems like TPP [33], MEMTIS [29], NOMAD [44], and Nimble [46], each designed to optimize data access across different memory tiers.

2.2 Tiered Memory Top-down

Trade-off between CPU and hotness detection algorithm. Both hardware-based and software-based access tracking involve CPU usage [35, 39], a critical performance bottleneck for tiered memory systems. There is no one-fit-all hotness detection algorithm [45]. Achieving high precision and finer monitoring granularity for detecting hot pages requires tiered memory systems to either shorten the monitoring interval or increase the number of counters, both of which demand additional CPU resources.

Interference between migration and applications. To adapt to dynamically changing workloads, tiered memory requires to frequently migrate data between the fast and slow tiers, leading to data copying overhead. The extra CPU utilization for monitoring and migration will interfere with application performance [23], thus a well-designed tiered memory system may not be the best choice. As shown in Figure 1, increasing the sampling interval results in higher CPU utilization and more page migrations, which can degrade application performance eventually.

Page copy is still a problem for tiered memory. One of the critical challenges is the significant CPU overhead associated with page migration and memory copying. This issue has been explored in previous tiered memory works [27, 29, 44]. Nimble reports 30% in the page migration phase. Our empirical tests further highlight this challenge, particularly in scenarios involving mixed workloads (e.g., co-location [17]) and hot start (e.g., serverless [31]). The core reason for this overhead is the CPU's involvement in memory copying, which leads to CPU stalls. To address this, we explore solutions such as offloading memory copy, which could alleviate the CPU's burden and improve system performance.

Figure 2 shows the function sampling results of kernel migrating process captured by perf [2] on MEMTIS. The proportion of migrating pages is 72.87%, while getting page info from index (i.e., `get_pginfo_idx`, which involves kernel's reverse mapping mechanism and update page access statistics) is 29.82% and memory copying (`migrate_page_list`) is 34.43%. About 73.51% of processor cycles in migrating pages are spent on copying pages.

Huge Pages are commonly found in tiered memory. Transparent Huge Page (THP) is utilized to enhance performance in applications [38], such as cloud [20] and database [21]. Most tiered memory works enable THP, so we measure the processing of TPP. Results show that Huge Pages will be the majority if THP [3] is enabled. For PageRank [22] workload, during the demotion, there are 1249067 4KB Pages and 8670 2MB Huge Pages. Thus Huge Pages account for 78.0% memory.

2.3 Observations of DSA

DSA [24] in Intel 4/5th (Sapphire/Granite Rapids) Xeon CPUs can enhance data copy and transformation. DSA hardware comprises interfaces for host communication, work queues (WQs) for descriptor storage, processing engines (PEs) for task execution, and arbiters for quality of service [25]. DSA devices appear as single root complex integrated endpoints, compatible with PCIe. Users submit 64B descriptors via MMIO portals; these include operation details like operation type, address, and batching list. Descriptors are stored in WQs, which can be either a dedicated WQ for single-client use or a shared WQ for multiple clients without synchronization.

DSA's address translation cache interacts with the IOMMU, supporting coherent shared memory with CPU cores, eliminating the need for memory pinning. The software architecture includes the Intel Data Accelerator Driver (IDXD) [7] for DSA

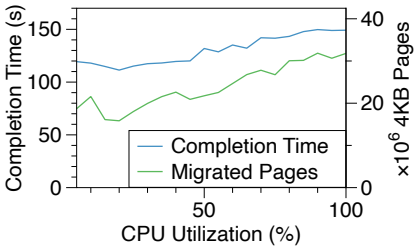

Figure 1: Analysis of hotness detection.

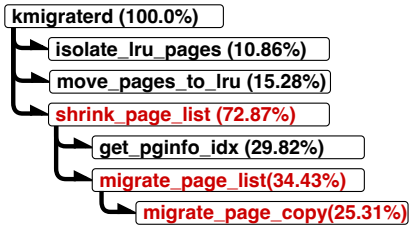

Figure 2: Profiling of migrate process.

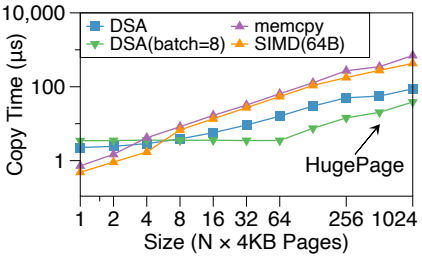

Figure 3: DSA/Memcpy comparisons.

management [19]. Applications exploit `libaccel-config` API to use DSA in user space, and IDXD allows WQ portal address exposed to user space via `mmap`.

The performance of DSA is higher than CPU. While copying data between local/remote NUMA nodes, a DSA can attain a bandwidth of up to 32 GBps, whereas an individual CPU core achieves a maximum bandwidth of only 9 GBps. Moreover, the time required for data copying with a DSA is consistently less than that of a CPU when the size of the data exceeds 32 KB. Furthermore, memory copying increases CPU utilization, leading to interference with ongoing tasks due to significant contention for CPU resources [42].

Not all memory copy sizes benefit from DSA. Figure 3 shows that using DSA performs higher performance than CPU copying when the transfer size is above 32 KB, where each descriptor is submitted and completed before sending another. The submitted descriptors are processed in a pipelined fashion [25], so CPU copying is faster than DSA in small transfer size (< 32 KB). As the transfer size increases, the submission overhead of DSA can be disregarded, thereby showcasing its performance advantage compared to CPU copying.

Number of WQs and batch size of DSA. Figure 4/5 shows that the number of WQs significantly influences maximal throughput. The number of WQs is positively correlated with increased throughput. The throughput attains its maximum when configured with four WQs. Users should carefully consider transfer size, as it affects scalability differently.

The DSA-facilitated batching of descriptors serves to reduce the cost associated with offloading. As shown in Figure 6, the performance improvement plateaus as the batch size reaches 8. Additionally, DSA enhances throughput in a non-linear manner, and it depends on the transfer size.

2.4 DSA v.s., DMA

DMA and DSA operate at different abstraction levels. DMA provides a general interface that various underlying engines, such as DSA or Host Bridge can support, and these interfaces abstract memory operations, letting developers work without dealing with complex hardware drivers [49]. Directly manipulating DSA driver functions (i.e., IDXD) offers less abstraction. However, this approach balances ease of use with advanced functionality and performance. Utilizing DSA directly, without dependence on the DMA interface, presents two principal advantages:

Elimination of memory pinning overhead. Most devices using DMA cannot tolerate page faults, requiring guest buffers to be pinned in host memory and mapped in the IOMMU page table before DMA access. Consequently, DMA becomes problematic in tiered memory scenarios, a major limitation, especially when hosting multiple virtual machines. In contrast, DSA supports the use of either physical or virtual addresses. The use of virtual addresses that are shared with processes running on the CPU is called shared virtual memory. Therefore, address translation does not require IOMMU page table mapping or memory pinning.

Significant performance improvement. In the submitting descriptor phase, DSA shows better performance than DMA. In DSA-2LM, the DSA descriptors are designed as per-cpu variables, so there is no dynamic descriptor (de)allocation. In contrast, using DMA interfaces should wait for the descriptor to be ready. We measure single copy operation for one 4KB page shown in Figure 7. DMA operation spends an extra 88*ns* in total to (un)map DMA buffer, prepare memory copy and submit DMA request. In contrast, only 6*ns* time is required to submit DSA descriptor. The major expensive phase is waiting for the completion of each DMA/DSA operation. According to evaluations in Section 2.3, utilizing features such as DSA batching and multi-channel DSA utilization further boost performance compared to the previous generation DMA engines such as Crystal Beach DMA (CBDMA) and Intel I/O Acceleration Technology. DSA provides 2.1× greater throughput on average and up to 8× higher throughput than CBDMA [25].

2.5 Challenges and Design Principles

Above all, there are three primary challenges: (1) How to directly use DSA in kernel-space for migrating pages and cooperating with tiered memory system (2) How to let both 4KB Page and 2MB Huge Page enjoy the benefit of DSA, and (3) How to dynamically choose appropriate batch size and number of WQs for each descriptor to maximum performance without harming application performance. Thus, we should redesign the tiered memory system considering the features of DSA from three aspects.

Use DSA directly for page migration in kernel space. Utilizing the CPU for page migration will increase CPU footprint, thereby affecting the performance. Section 2.3 shows that using DSA can alleviate CPU overhead and exhibit superior performance in page migration. For in-kernel program-

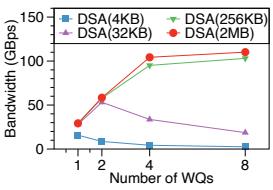

Figure 4: The effects of different number of WQs.

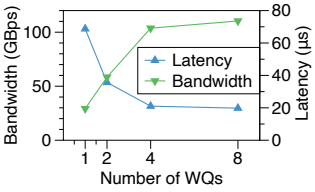

Figure 5: The bandwidth and latency of multi-WQ.

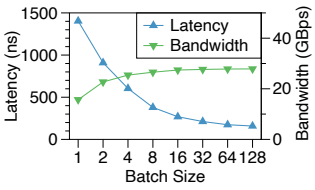

Figure 6: The effects of batch size.

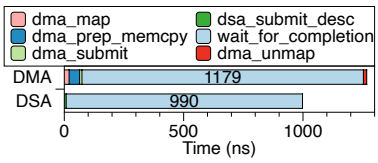

Figure 7: The performance breakdown of DSA and DMA.

ming, we bypass DMA interface and directly use physical address in submitted descriptors. Our evaluation confirms that using DSA in kernel space does not trigger any page fault.

Use asynchronous mode of DSA properly. Handling hardware interrupts in user space can be tricky. Intel officially recommends using `UMONITOR/UMWAIT` instructions after submitting descriptors to reduce power consumption of the CPU core [11, 25, 49]. However, this approach does not achieve true asynchronous mode, as the CPU cannot schedule other tasks while in a low-power state. Intel DTO [1] adopts the `sched_yield` syscall to proactively relinquish the CPU to avoid busy polling, but this sacrifices some responsiveness. Hence, such user-space programming models are not applicable in kernel space. DSA-2LM must carefully design and utilize the advantage of handling interrupts in kernel space, achieving genuine and efficient asynchronous waiting.

Aggregate data path for both small/large size page migration. Section 2.3 presents that only large-sized page migration can benefit from DSA. Naive design is using two separate data paths for page migration. For 4KB Pages, DSA-2LM uses the original `copy_page` method. For 2MB HugePages, DSA-2LM uses DSA-oriented copying functions. A separate data path means extra overhead and small-sized pages cannot benefit from DSA. We need a new data path with adaptable algorithm to aggregate small and large pages.

Carefully tuning parameters for DSA. As we discuss in Section 2.3, using different values of batch size and number of WQs will significantly influence the copying rate. For a page list to be migrated, we should tune the value of parameters to maximize performance. So we need to know the performance changes and adjust the parameters dynamically according to the page number and each page size in the page list.

3 Design of DSA-2LM

3.1 Architecture Overall

DSA-2LM consists of three steps: track page access using hardware performance counter, choose hotness pages, and migrate pages in the background with DSA (Figure 8). (1) DSA-2LM uses the page histogram mechanism, which is the same as MEMTIS [29], to record the access of pages. It samples memory access in the page granularity using PEBS, and updates the hotness distribution of all allocated pages periodically. (2) DSA-2LM ensures that the size of the hot set is

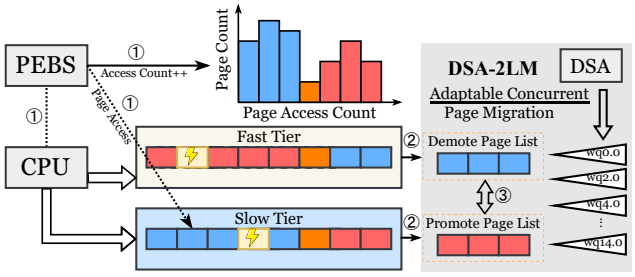

Figure 8: DSA-2LM overall.

maintained near the capacity of the fast tier, thereby enabling the fast tier to accommodate all hot pages. It chooses hot/cold pages according to its access count higher/lower than the thresholds T_{hot} and T_{cold}. These thresholds are dynamically adjusted to fit into the total pages. Then selected hot/cold pages are moved to promotion/demotion page list. (3) The per-node kernel threads responsible for migration perform the migration operation including demotion and promotion. Such a kernel thread will be woken up periodically in a time interval (e.g., 200 ms). DSA-2LM decides to migrate pages according to the available space in the fast tier. It detects the status of DSA, and it migrates the page list using available DSA devices in the kernel space. DSA-2LM uses an adaptable concurrent migration algorithm to copy page lists that include 4KB Pages and 2MB Huge Pages.

Compared to MEMTIS, DSA-2LM inheriting its hotness detection algorithm, page placement strategies and relying on the `kmigrated` kernel thread for page migration. Additionally, DSA-2LM improves MEMTIS's sampling algorithm to consistently utilize the most recent samples, thus avoiding using outdated samples. Crucially, DSA-2LM introduces several key improvements that are specific to DSA. Inspired by Nimble [46], DSA-2LM introduces a concurrent page migration strategy. DSA-2LM applies the DSA adaptive migration algorithm to optimize page copying. Because DSA is CPU-free, DSA-2LM shortens the wake-up interval of the kmigrated thread, adopting a more aggressive migration strategy without incurring extra overhead.

3.2 Implementation in Kernel

An extra moudle is implemented under the `misc/exp` directory, which exposes two new APIs: `dsa_multi_copy_pages` and `dsa_copy_page_lists`. The functionality of DSA-2LM

can be toggled via a sysAPI interface, and various statistical parameters are presented through `procfs` for monitoring purposes. The state-of-the-art tiered memory systems like MEMTIS [29] and TPP [33] are based on the 5.x kernel. To ensure compatibility with existing systems, the overall implementation is based on Linux kernel version 5.13/5.15. The modifications to the memory management subsystem and DSA acceleration module implementation comprise approximately 2K LoC, while the backporting of the IDXD and IOMMU drivers from kernel 6.4 introduces around 8K LoC.

3.3 Migration Workflow

DSA-2LM creates a migration kernel thread for each memory tier to handle promotion and demotion operations in the background. The system maintains two lists: a promotion list containing hot pages for the slow tier and a demotion list containing cold pages for the fast tier.

Whenever processing a memory access sample, DSA-2LM compares the page's hotness factor (H_i) to the threshold T_{hot}. If the page is determined as hot, it is moved to the promotion list. Also, the migration kernel thread periodically performs cooling by halving a page's access count for every N sampled events. This process can cause some pages to transition to cold states, in which case they are moved to the demotion list.

In the slow tier, the migration thread checks for hot pages and determines if there is available space in the fast tier. If both conditions are met, it promotes hot pages to the fast tier. Conversely, when the available memory in the fast tier drops below a predefined free-space threshold (set at 3% of the fast tier's total size for future page allocations and promotions), the demotion operation is initiated.

This demotion process involves selecting victim pages from the fast tier. Initially, cold pages ($H_i \leq T_{\text{cold}}$) are demoted back to the slow tier. It stops until sufficient free space is reclaimed. Otherwise, cold pages are demoted until the necessary free space is achieved. This strategy allows DSA-2LM to retain as many cold pages as possible within the fast tier.

3.4 Adaptable Concurrent Migration

After promotion/demotion page list is prepared, the migration thread uses DSA to copy pages containing mixed 4KB Pages and 2MB Huge Pages.

Before migrating (see Algorithm 1), the total number of available WQs must be initialized. `limit_chans` specifies the total number of available DSAs used for page migration and should be configured properly according to configurations (typically a power-of-two value with DSA-2LM defaulting to 8). In DSA-2LM, each DSA device enables four PEs and one WQ, so DSA device and WQ are one-to-one. DSA-2LM enumerates each WQ, updating the count of available WQs when DSA-2LM is initialized first or `limit_chans`/`dsa_state` are updated through sysfs API.

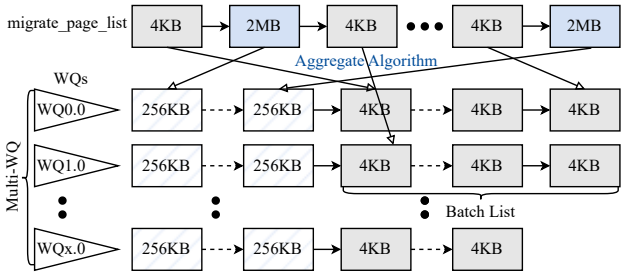

Figure 9: Workflow of adaptable page migration.

Algorithm 1 DSA Migrate Page List

Input: *page_list*: 4KB and 2MB mixed page pair lists to be migrated (A page pair include source/destination page)

1: get `idxd_desc/dsa_hw_desc/dsa_completion_record/completion` pointers from per-cpu `struct` variables
2: **repeat**
3: *nr_base_pages* ← 0
4: **for** each unsolved *page_pair* in *page_list* **do**
5: **if** *page_pair* is THP/HugePage **then**
6: *transfer_size* ← 2MB / `limit_chans`
7: **for** each available WQ *i* **do**
8: Prepare a `memmove` DSA descriptor for (*page_pair*, *transfer_size*, WQ *i*)
9: Submit a DSA descriptor
10: **else**
11: *nr_base_pages* ← *nr_base_pages* + 1
12: *batch_size* ← *nr_base_pages* / `limit_chans`
13: *batch_list* ← ∅
14: **for** each unsolved 4KB *page_pair* in *page_list* **do**
15: Prepare a `memmove` DSA descriptor *desc* for *page_pair*
16: *batch_list* ← *batch_list* ∪ *desc*
17: **if** number of *batch_list* fits *batch_size* **then**
18: Choose a WQ *t* in round robin
19: Prepare a `batch` DSA descriptor for (*batch_list*, *batch_size*, WQ *t*);
20: Submit a DSA descriptor
21: *batch_list* ← ∅
22: **for** each available WQ *i* **do**
23: Yield and wait for WQ *i* completion
24: **until** each page pair in *page_list* has been migrated

Aggregate data path. When the page order is greater than 3 (i.e., 32 KB), utilizing DSA for copying pages achieves lower latency compared to using the CPU. For non-contiguous pages, we employ batch processing to reduce additional overhead. In case of contiguous pages, such as 2MB Huge Pages, we leverage multi-WQ parallel copying to enhance efficiency.

Combining these strategies altogether, Figure 8 and Algorithm 1 show the workflow of handling the promotion/demotion page list. In the first pass, DSA-2LM picks the 2MB Page and splits it into `limit_chans` (e.g., 8) subpages whose size is 2MB / `limit_chans` (e.g., 256 KB), and assigns subpages to each WQ in round robin. In the second pass, DSA-2LM calculates the average transfer size (i.e., `batch_size`) for each WQ based on the remaining 4KB Pages, and distributes them into batch lists in each WQ. According to the "shortboard effect", DSA-2LM balances the total transfer size submitted to each WQ so that the difference among WQs does

not exceed the size of a single 4KB page. If the `batch_size` is less than two in a corner case, DSA-2LM directly submits the remaining 4KB pages to each WQ.

Concurrent migration with both batch and multi-WQ. According to `limit_chans` of batch lists, DSA-2LM initializes the necessary DSA data structures for processing memory copy, including `idxd_desc`, `dsa_hw_desc`, and `dsa_completion_record`. These data structures are created as per-cpu global variables in the initialization phase, to avoid the preparation overhead that exists in DMA processing.

Subsequently, for each WQ, DSA-2LM submits DSA descriptors, ensuring any errors encountered are handled appropriately. Since adopting asynchronous mode, DSA-2LM leverages the Linux kernel's `completion` mechanism to await completion of copy requests. Once the `status` field in the `dsa_completion_record` is set by hardware, it triggers an MSI-X interrupt that awakens the `idxd_wq_thread`, which then scans all pending DSA descriptors. For each completed descriptor, `idxd_wq_thread` invokes the corresponding callback function, thereby signaling the `completion`. Consequently, the migration thread is unblocked from `wait_for_completion_timeout` and proceeds with the subsequent operations. DSA-2LM sets a timeout for each submitted descriptor, and returns fails if it occurs. This workflow significantly enhances the efficiency and speed of page copying operations in the kernel.

4 Evaluations

4.1 Evaluation Configurations

Hardware configuration. Our testbed is a dual-socket server equipped with 4th Gen Xeon Platinum CPU (48×2 physical cores), where each socket has 1 TB (256 GB×4) DDR5 DRAM. One of the sockets is connected to a 64 GB ASIC-based Montage Technology CXL device. We disabled Intel Hyper-Threading and fixed the CPU frequency to 3.2 GHz. We set CPU affinity for workloads and used only 32 cores of one socket. Similar to prior works, we used a DRAM NUMA node (load/store: 112 ns) and a CPU-less node with CXL memory (load/store: ∼300 ns), and THP [16] is enabled. DSA-2LM can also support two NUMA nodes without CXL. We configured the CXL-attached memory to `system-ram` mode. It allows load/store access to CXL memory in kernel space.

Software configuration. We use different methods to equitably constrain the fast tier. For AutoNUMA, TPP and NOMAD, we changed the kernel boot argument (memmap GRUB option [5]) to limit the fast tier size. The fast tier size is set as 16/32 GB. For MEMTIS, we used a memory `cgroup` interface to control the size of the fast tier. The ratios of fast/slow tier memory size were established from 1:2 to 1:16. The proportion of the fast tier size was adjusted from 33% (1/3) to 5.9% (1/17) of the resident set size (RSS) for each benchmark.

Workload	RSS	Description
Graph500	68.0 GB	Graph generation and breadth-first search of some random vertices [18].
PageRank	12.3 GB	Compute the PageRank score with an iterative method [18]. (Twitter dataset [26])
XSBench	63.4 GB	Monte Carlo neutron transport algorithm [41].
BTree	38.3 GB	In-memory index lookup benchmark [4].
Pandas	20.2∼92.6 GB	Data processing with six different queries [12].

Table 1: Workload characteristics.

4.2 Performance of Copy Pages

Two distinct workloads A/B, are employed to assess the impacts of our adaptable copying algorithm. Workload A comprises 1023 4KB pages and a single 2MB huge page, with the smaller 4KB pages constituting the bulk of the workload. Conversely, workload B is characterized by a high page intensity, consisting of 1000 4KB pages and 24 2MB huge pages.

As shown in Figure 10, we measure the bandwidth of continuous page migration with three strategy: 1) CPU: only use CPU to copy pages; 2) DSA (raw): migrating 4KB page with CPU and 2MB page with DSA; and 3) DSA (opt): use adaptable concurrent migration proposed in Section 3.4. For workload B, the bandwidth of DSA-2LM (106.3 GBps) is 14.38/2.19× higher than CPU/DSA (raw) implementations. We conclude DSA-2LM can easily saturate the bandwidth (the peak bandwidth in our evaluation platform is ∼110 GBps).

4.3 Performance Breakdown

We repeat to capture the result of function samples in kernel mode by perf and Figure 11 shows the results. The overall proportion of migrating pages in is dropped from 39.0% to 4.24%, which is only 2.51% compared with the original overhead. This proves that our approach can effectively reduce CPU overhead in the migration data path.

Figure 12 shows copying duration in real-time during the execution of Graph500. The application completes in 142s, with a total copying time of 14.9s when employing CPU-based migration. Conversely, using DSA-based methods DSA-2LM for page migration reduces the total copying time to 1.39s, representing only 9.3% of the baseline duration. In the results reported in DSA-2LM, the copying time accounts for only 0.97% of the execution time, whereas it comprises 10.5% in the context of original tiered memory.

4.4 Real-world Applications

We evaluate five representative workloads in Table 1. Since MEMTIS and TPP limit fast-tier memory size in different ways, it is difficult to reserve the same fast-tier memory size. For a fair comparison, we presented them in separate figures.

The upper part of Figure 13 compares DSA-2LM with TPP, NOMAD, and AutoNUMA. We successfully applied DSA to other state-of-the-art tiered memory systems using similar approaches, denoted as TPP+, NOMAD+, and AutoNUMA+ in Figure 13. We set the fast tier as 16/32 GB.

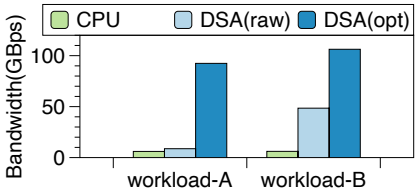

Figure 10: The performance of adaptable copying algorithm.

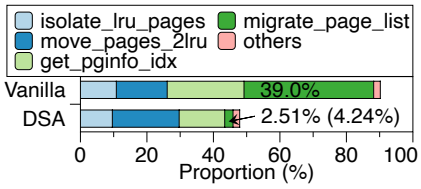

Figure 11: The perf breakdown in DSA-2LM.

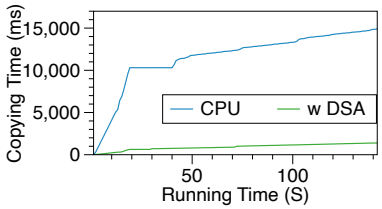

Figure 12: CDF of copying time.

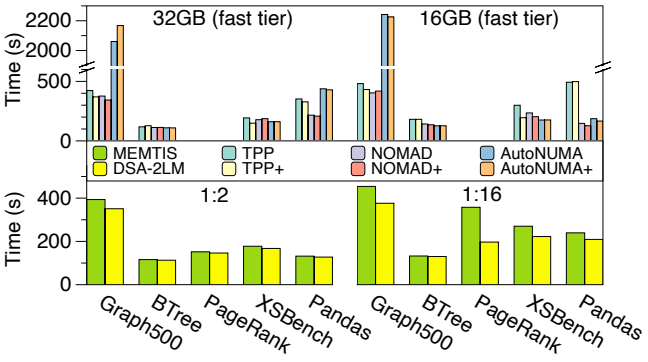

Figure 13: Performance with different applications.

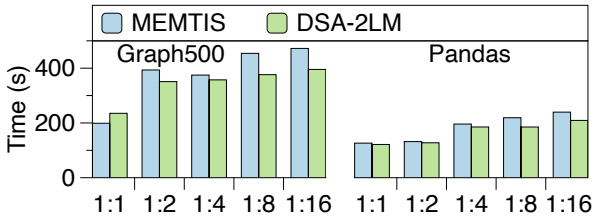

Figure 14: Performance with different ratio.

Note that PageRank's RSS is less than 16 GB, so it is excluded from this comparison. For TPP, Graph500 and XSBench show performance improvements of 14.5%/11.5% and 29.8%/53.9% respectively for the 32/16 GB configuration. For NOMAD, the baseline already outperforms TPP in the majority of cases across four workloads. When DSA is integrated with NOMAD, two-thirds of the cases show 4%∼16% performance improvements. This enhancement is attributed to the increased page migration efficiency, which effectively reduces the migration window and consequently decreases the probability of transactional migration failures. For AutoNUMA, since it employs a more conservative page placement strategy, DSA integration generally yields minimal performance changes. In Pandas, AutoNUMA+ achieves performance improvements of 2.0% and 12.1% for the 32 GB and 16 GB configurations, respectively.

The bottom part of Figure 13 compares DSA-2LM against MEMTIS. When the ratio of fast/slow tier is 1:2, DSA-2LM outperforms MEMTIS in all evaluated benchmarks by 2.5∼12.0% for five applications. When increasing the ratio to 1:16, the performance enhancement averages 28% across all benchmarks. In the best case, DSA-2LM achieves a 1.8× speedup compared to MEMTIS. This significant improvement occurs because the fast tier memory can only accommodate a small subset of hot pages, causing the hottest page set to change rapidly. DSA-based fast page migration enables better adaptation to these dynamic access pattern changes.

We measure Graph500/Pandas as represented applications for further case analysis (Figure 14). When the ratio is 1:1, the performance increase is insignificant. When the ratio is 1:8, the completion time can decrease by 20.68% in Graph500. The effects of DSA will be better with the ratio increasing.

5 Related Works

Compatibility with other tiered memory systems. Nomad [44] employs Transactional Page Migration (TPM), abandoning the page copy if the page is modified during migration. Since DSA can speed up page migration by 5∼10×, which significantly reduces the migration window and consequently decreases the probability of TPM failures. In the case of Colloid [42], the key insight is balancing access latencies across different memory tiers. Then it determines the direction of page migration based on global access latency. In this context, page migration is still necessary, and our work is orthogonal to Colloid. Additionally, recent works like NeoMem [52] have been able to offload the page placement strategy to hardware as well. It would be interesting to integrate DSA-2LM with NeoMem to achieve a fully CPU-free solution.

Intel accelerator use cases. Recent research has demonstrated the effectiveness of Intel accelerators in optimizing storage, networking, and virtualization. For instance, AnolisOS leverages "Memory Fill" capability to pre-zero memory pages, reducing VM startup latency in large-memory configurations by offloading page-zeroing from CPUs to DSA [6, 9]. The Intel IAA plugin for RocksDB provides accelerated compression/decompression in RocksDB [14]. For storage and networking, DSA accelerates CRC/DIF generation and memory comparison through specialized hardware pipelines while maintaining data integrity [13, 19]. DSA-2LM is the first work to utilize DSA in tiered memory.

6 Conclusion

This paper introduces DSA-2LM, an efficient tiered memory system using DSA. It exploits the hardware capabilities of DSA to design an aggregation algorithm for adaptable data paths. DSA-2LM leverages batch/multi-WQ approaches for concurrent migration. Results on the real CXL platform show that applications benefit significantly from DSA.

References

[1] Intel® dsa transparent offload library. https://github.com/intel/DTO. [Accessed 29-05-2025].

[2] Perf Wiki — perf.wiki.kernel.org. https://perf.wiki.kernel.org/index.php/Main_Page. [Accessed 24-06-2024].

[3] Transparent hugepage support. https://docs.kernel.org/admin-guide/mm/transhuge.html. [Accessed 24-06-2024].

[4] Mitosis workload btree. https://github.com/mitosis-project/mitosis-workload-btree, 2020.

[5] Using the memmap kernel option. https://docs.pmem.io/persistent-memory/getting-started-guide/creating-development-environments/linux-environments/linux-memmap, 2020.

[6] Anolis cloud kernel. https://gitee.com/anolis/cloud-kernel, 2022.

[7] idxd driver for intel data streaming accelerator. https://lwn.net/Articles/805291/, 2022.

[8] Intel® agilex™ i-series fpga development kit user guide. https://www.intel.com/content/www/us/en/docs/programmable/683288/current/overview.html, 2022.

[9] Introduce page prezero with dsa support. https://gitee.com/anolis/cloud-kernel/pulls/702, 2022. [Accessed 03-06-2025].

[10] Intel® data streaming accelerator architecture specification. https://www.intel.com/content/www/us/en/content-details/671116/intel-data-streaming-accelerator-architecture-specification.html, 2024.

[11] Intel® data streaming accelerator user guide. https://www.intel.com/content/www/us/en/content-details/759709/intel-data-streaming-accelerator-user-guide.html, 2024.

[12] Polars decision support (pds) benchmarks. https://pola.rs/posts/benchmarks/, 2024.

[13] Tuning guide for user space network stack acceleration with intel ® dsa. https://www.intel.com/content/www/us/en/developer/articles/guide/user-space-network-stack-acceleration-dsa.html, 2024.

[14] Intel® in-memory analytics accelerator plugin for rocksdb storage engine. https://github.com/intel/iaa-plugin-rocksdb, 2025.

[15] Neha Agarwal and Thomas F Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 631–644, 2017.

[16] Andrea Arcangeli. Transparent hugepage support. In *KVM forum*, volume 9, 2010.

[17] Yossi Azar, Seny Kamara, Ishai Menache, Mariana Raykova, and Bruce Shepard. Co-location-resistant clouds. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, pages 9–20, 2014.

[18] Scott Beamer, Krste Asanović, and David Patterson. The gap benchmark suite, 2017.

[19] André Berthold, Constantin Fürst, Antonia Obersteiner, Lennart Schmidt, Dirk Habich, Wolfgang Lehner, and Horst Schirmeier. Demystifying intel data streaming accelerator for in-memory data processing. In *Proceedings of the 2nd Workshop on Disruptive Memory Systems*, pages 9–16, 2024.

[20] Alibaba Cloud. Thp-related performance optimization in alibaba cloud linux. https://www.alibabacloud.com/help/en/ecs/transparent-huge-page-thp-related-performance-optimization-in-alibaba-cloud-linux-2, 2024.

[21] Google Cloud. Cloud sql for postgresql now supports linux huge pages. https://cloud.google.com/blog/products/databases/cloud-sql-postgresql-now-supports-linux-huge-pages, 2024.

[22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[23] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, et al. 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, Volume 3*, pages 727–741, 2023.

[24] Intel. Intel® data streaming accelerator (intel® dsa). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/data-streaming-accelerator.html. [Accessed 24-06-2024].

[25] Reese Kuper, Ipoom Jeong, Yifan Yuan, Ren Wang, Narayan Ranganathan, Nikhil Rao, Jiayu Hu, Sanjay

Kumar, Philip Lantz, and Nam Sung Kim. A quantitative analysis and guidelines of data streaming accelerator in modern intel xeon scalable processors. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 37–54, 2024.

[26] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 591–600, New York, NY, USA, 2010. Association for Computing Machinery.

[27] Woosuk Kwon, Gyeong-In Yu, Eunji Jeong, and Byung-Gon Chun. Nimble: Lightweight and parallel gpu task scheduling for deep learning. *Advances in Neural Information Processing Systems*, 33:8343–8354, 2020.

[28] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, et al. Software-defined far memory in warehouse-scale computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 317–330, 2019.

[29] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. Memtis: Efficient memory tiering with dynamic page classification and page size determination. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 17–34, 2023.

[30] Huaicheng Li, Daniel S Berger, Stanko Novakovic, Lisa Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. Pond: Cxl-based memory pooling systems for cloud platforms. In *ASPLOS 2023*, page 0, 2023.

[31] Zijun Li, Linsong Guo, Jiagan Cheng, Quan Chen, Bing-Sheng He, and Minyi Guo. The serverless computing survey: A technical primer for design architecture. *ACM Computing Surveys (CSUR)*, 54(10s):1–34, 2022.

[32] Teng Ma, Zheng Liu, Chengkun Wei, Jialiang Huang, Youwei Zhuo, Haoyu Li, Ning Zhang, Yijin Guan, Dimin Niu, Mingxing Zhang, et al. {HydraRPC}:{RPC} in the {CXL} era. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 387–395, 2024.

[33] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. Tpp: Transparent page placement for cxl-enabled tiered-memory. pages 742–755, 2023.

[34] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. Introducing the graph 500. *Cray Users Group (CUG)*, 19(45-74):22, 2010.

[35] Alan Nair, Sandeep Kumar, Aravinda Prasad, Ying Huang, Andy Rudoff, and Sreenivas Subramoney. Telescope: telemetry for gargantuan memory footprint applications. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 409–424, 2024.

[36] Stanko Novakovic, Yizhou Shan, Aasheesh Kolli, Michael Cui, Yiying Zhang, Haggai Eran, Boris Pismenny, Liran Liss, Michael Wei, Dan Tsafrir, et al. Storm: a fast transactional dataplane for remote data structures. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, pages 97–108, 2019.

[37] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. Hemem: Scalable tiered memory management for big data applications and real nvm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 392–407, 2021.

[38] Jie Ren, Dong Xu, Junhee Ryu, Kwangsik Shin, Daewoo Kim, and Dong Li. Mtm: Rethinking memory profiling and migration for multi-tiered large memory. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 803–817, 2024.

[39] Muhammad Aditya Sasongko, Milind Chabbi, Paul HJ Kelly, and Didem Unat. Precise event sampling on amd versus intel: Quantitative and qualitative comparison. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1594–1608, 2023.

[40] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. Demystifying cxl memory with genuine cxl-ready systems and devices. pages 105–121, 2023.

[41] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis. *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.

[42] Midhul Vuppalapati and Rachit Agarwal. Tiered memory management: Access latency is the key! In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 79–94, 2024.

[43] Zixuan Wang, Suyash Mahar, Luyi Li, Jangseon Park, Jinpyo Kim, Theodore Michailidis, Yue Pan, Tajana Rosing, Dean Tullsen, Steven Swanson, Kyung Chang Ryoo, Sungjoo Park, and Jishen Zhao. The hitchhiker's

guide to programming and optimizing cxl-based heterogeneous systems, 2024.

[44] Lingfeng Xiang, Zhen Lin, Weishu Deng, Hui Lu, Jia Rao, Yifan Yuan, and Ren Wang. Nomad:{Non-Exclusive} memory tiering via transactional page migration. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 19–35, 2024.

[45] Dong Xu, Junhee Ryu, Kwangsik Shin, Pengfei Su, and Dong Li. {FlexMem}: Adaptive page profiling and migration for tiered memory. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 817–833, 2024.

[46] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. Nimble page management for tiered memory systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 331–345, 2019.

[47] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 169–182, 2020.

[48] Jifei Yi, Benchao Dong, Mingkai Dong, and Haibo Chen. On the precision of precise event based sampling. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 98–105, 2020.

[49] Yifan Yuan, Ren Wang, Narayan Ranganathan, Nikhil Rao, Sanjay Kumar, Philip Lantz, Vivekananthan Sanjeepan, Jorge Cabrera, Atul Kwatra, Rajesh Sankaran, et al. Intel accelerators ecosystem: An soc-oriented perspective: Industry product. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 848–862. IEEE, 2024.

[50] Mingxing Zhang, Teng Ma, Jinqi Hua, Zheng Liu, Kang Chen, Ning Ding, Fan Du, Jinlei Jiang, Tao Ma, and Yongwei Wu. Partial failure resilient memory management system for (cxl-based) distributed shared memory. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 658–674, 2023.

[51] Yuhong Zhong, Daniel S Berger, Carl Waldspurger, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D Hill, Mosharaf Chowdhury, and Asaf Cidon. Managing memory tiers with cxl in virtualized environments. In *Symposium on Operating Systems Design and Implementation*, 2024.

[52] Zhe Zhou, Yiqi Chen, Tao Zhang, Yang Wang, Ran Shu, Shuotao Xu, Peng Cheng, Lei Qu, Yongqiang Xiong, Jie Zhang, and Guangyu Sun. NeoMem: Hardware/Software Co-Design for CXL-Native Memory Tiering . In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1518–1531, Los Alamitos, CA, USA, November 2024. IEEE Computer Society.