Bandwidth Expansion via CXL: A Pathway to Accelerating In-Memory Analytical Processing

Wentao Huang
National University of Singapore
huang@comp.nus.edu.sg

Mo Sha
Alibaba Cloud
shamo.sm@alibaba-inc.com

Mian Lu
4Paradigm Inc.
lumian@4paradigm.com

Yuqiang Chen
4Paradigm Inc.
chenyuqiang@4paradigm.com

Bingsheng He
National University of Singapore
hebs@comp.nus.edu.sg

Kian-Lee Tan
National University of Singapore
tankl@comp.nus.edu.sg

ABSTRACT

The introduction of Compute Express Link (CXL) technology marks a transformative phase for modern database management systems (DBMSs), offering unprecedented enhancements in memory management and system performance. By extending the cache-coherent memory domain to encompass PCIe ports, CXL facilitates a notable expansion in host memory capacity and boosts the performance of peripheral components. While a substantial body of research has explored the benefits of CXL, particularly its capability to augment memory capacity, the vital role of bandwidth expansion—a key determinant of database performance—has frequently been underemphasized. This study shifts focus to the untapped potential of memory bandwidth expansion offered by CXL technology in a real-world deployment context. Our evaluation demonstrates that, rather than relying on conventional tiered memory design, a meticulous interleaving of CXL memory with host memory delivers up to a 1.61x performance gain for in-memory analytical workloads compared to a solely host memory platform.

VLDB Workshop Reference Format:
Wentao Huang, Mo Sha, Mian Lu, Yuqiang Chen, Bingsheng He, and Kian-Lee Tan. Bandwidth Expansion via CXL: A Pathway to Accelerating In-Memory Analytical Processing. VLDB 2024 Workshop: Fifteenth International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures(ADMS 2024).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/fukien/type3-pathway.

1 INTRODUCTION

Modern database management systems (DBMS) have been suffering from limited memory capacity in the past decade. As the DRAM memory scaling ability fails to keep up with the scaling speed on in-chip processors, workload performance per core is expected to drop at a factor of 30% biannually [36]. To mitigate the shortfall of memory resources, DBMSs have resorted to memory over-provisioning [48], aiming to satisfy the demands of real-world applications. This approach, while temporarily effective, escalates the total cost of ownership (TCO) and gives rise to notorious memory stranding issues [1, 30] [1].

Compute Express Link (CXL) technology [10, 42] addresses these challenges by offering a new approach to memory resource management. Integrated within the PCIe layer, CXL enhances traditional DDR memory channels by adding substantial memory resources and providing cache-coherent memory semantics, which enables host systems to efficiently perform load/store operations with peripheral devices. Peripheral devices, such as GPUs, FPGAs, and smart NICs, can now leverage CXL to coherently share host-level cache and memory resources. This capability paves the way for innovative advancements in high-performance DBMS architecture.

The remarkable capabilities of CXL have captured the attention of both the research and industry sectors, prompting efforts to leverage this technology for scaling up or out the memory resources in modern DBMSs [2, 28]. Given that CXL memory and associated pooling resources interface with host systems through PCIe, they inherently exhibit higher access latencies compared to host memory [2]. Consequently, most current designs regard CXL memory as a slower tier of memory, focusing on supporting large-scale in-memory workloads with minimal performance penalties. Noteworthy implementations, such as DRAM-CXL tiered memory designs [31] and disaggregated memory pools [30], represent significant shifts in this strategic approach and have seen deployment in cloud centers in recent years.

Despite the opportunities CXL presents for expanding memory capacity in existing tiered system designs, a significant oversight remains: the underexplored potential of bandwidth expansion through CXL memory. Unlike traditional host memory, commonly referred to as DRAM [3], which connects to the host processor via DDR channels, CXL memory interfaces with the host via PCIe ports, thus potentially increasing the system's overall aggregated memory bandwidth [42]. This is particularly relevant as DDR channels are susceptible to bandwidth interference from concurrent applications [9, 21, 22, 49, 54], especially under the strain of bursty, memory-intensive workloads. The supplementary bandwidth from

Proceedings of the VLDB Endowment. ISSN 2150-8097.

[1] Memory stranding describes a scenario in data centers in which the full depletion of computing resources results in the underutilization of available memory.
[2] In rack-scale deployments, the utilization of CXL resources may even introduce additional overhead due to switch transfer cost.
[3] While other technologies like SRAM or ReRAM can be used for host memory manufacturing, DRAM technology dominates the production of modern host memory devices.

PCIe interfaces can therefore act as an auxiliary resource to sustain system throughput and retain a high application performance without much bandwidth degradation.

Moreover, the limited availability of commercial CXL memory devices has led most studies to rely on hardware emulation (e.g., NUMA emulation [5, 30], and FPGA emulation [28, 48]) or software simulation (e.g., gem5 [7], Ramulator [25], and DRAMSim2 [37]) for system design and evaluation, which may not accurately reflect the performance impacts on a real CXL platform. Given this context, it becomes imperative to reevaluate existing strategies for CXL utilization on genuine platforms.

Henceforth, we investigate the benefits of utilizing real CXL memory, particularly from a bandwidth expansion standpoint. We concentrate on in-memory DBMS analytical tasks, where performance greatly depends on the availability of substantial memory bandwidth and capacity. We conduct a comprehensive evaluation using an authentic CXL memory device. Our experimental findings indicate that, by meticulously configuring DRAM and CXL memory in an interleaved manner, analytical workloads can achieve performance gains comparable to or exceeding those of a pure-DRAM setup (in our experiments, we observe up to a 1.61x performance improvement compared to a pure-DRAM platform.). This result challenges the prevailing assumption that CXL memory serves solely as a slower tier in heterogeneous memory systems, prompting a reconsideration of memory resource management in cloud and data center environments.

2 BACKGROUND

2.1 CXL Memory Technology

Compute Express Link (CXL) represents a groundbreaking cache-coherent interconnect standard designed to facilitate communication between host processors, peripheral accelerators, and memory devices. Since its initial specification release [10], CXL has undergone several iterations, resulting in three primary versions: CXL 1.1, CXL 2.0, and CXL 3.0. These iterations introduce progressively advanced capabilities: CXL 1.1 focuses on memory expansion and tiering, while CXL 2.0 and CXL 3.0 advance towards memory pooling and disaggregation, with CXL 2.0 facilitating pooling via a single switch and CXL 3.0 extending this capability across multiple switches. Currently, only CXL 1.1 is commercially available; therefore, our experimental evaluation primarily focuses on this version.

The CXL transaction layer is segmented into three protocols: CXL.io, CXL.cache, and CXL.mem. CXL.io utilizes PCIe to manage device tasks such as discovery and configuration, in addition to I/O virtualization and DMA operations. CXL.cache allows CXL-equipped devices to access the host processor's memory, facilitating efficient data sharing and reducing latency. CXL.mem, on the other hand, empowers the host to directly manage memory on connected devices through load and store instructions, promoting seamless data operation in a unified memory ecosystem. Collectively, these protocols form the bedrock of the CXL architecture, paving the way for sophisticated memory coherency and connectivity solutions essential for cutting-edge computing platforms.

CXL technology is also engineered to accommodate three main device types: Type 1, Type 2, and Type 3. Types 1 and 2 provide

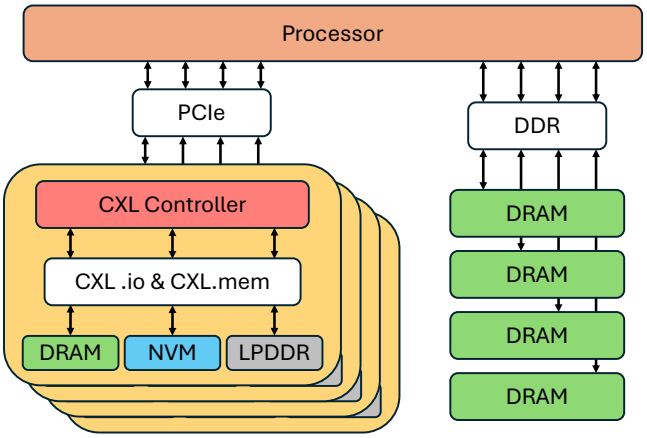

Figure 1: Schematic representation of a platform incorporating CXL Type 3 devices.

cache coherent access for peripheral accelerators, while Type 3 is dedicated to memory expansion. Given our focus on memory expansion, this paper primarily considers Type 3 CXL devices. Figure 1 illustrates a typical memory expansion setup utilizing Type 3 CXL memory modules. Each module encompasses a CXL controller that complies with CXL.io and CXL.mem protocols, facilitating access to a range of memory technologies including DRAM, non-volatile memory (NVM), and low-power double data rate SDRAM (LPDDR) [31]. Unlike conventional DRAM modules that utilize DDR communication channels, CXL type3 memory, regardless of the underlying technology, establishes connections to host processors via PCIe ports, significantly enhancing the bandwidth available to the system. This integration technology markedly increases both the memory capacity and overall system bandwidth, granting host systems the versatility to tailor configurations to meet the varied demands of diverse applications.

2.2 Related Work

A plethora of research efforts have been directed toward harnessing CXL memory for the design and evaluation of tiered or disaggregated systems across diverse workloads. Pond [30] emerges as the pioneering system to advocate for the enhancement of CXL memory pooling at the rack scale, employing machine learning techniques to develop an accurate model for hotspot prediction. This model facilitates memory migration and addresses memory stranding issues within Azure cloud centers. Following this, Meta introduces TPP [31], a DRAM-CXL tiered memory system design. TPP innovates by separating memory allocation from reclamation and introduces a hot page promotion mechanism to improve in-memory workload performance. SAP HANA [2, 28] conducted evaluations on OLAP and OLTP workloads within DRAM-CXL tiered systems using NUMA or FPGA emulation, concluding that performance penalties could be minimized by retaining memory regions pertinent to query execution within DRAM. Further, Lerner et al. [29] and Jang et al. [20] conceptualize CXL memory as a disaggregated memory pool, suggesting strategies to amplify in-memory workload performance. Other studies [5, 12–15, 26] have sought to distill

performance insights for CXL using Optane PMEM [18], primarily considering CXL memory as a capacity expansion rather than efficiency enhancement, thereby often overlooking its potential for bandwidth expansion.

Sun et al. [44] first address the performance gap between real CXL memory and emulation technology, and find that real CXL memory can expand overall memory bandwidth for embedding reduction applications. Although their exploration into embedding reduction yields crucial insights, it is important to note that such a workload is considered elementary. Moreover, their examination omits a crucial comparison with tiered architecture systems, a fundamental element in the analysis of modern in-memory analytical workloads [2, 5, 28, 43, 47]. Recognizing this oversight, our study seeks to bridge this gap by meticulously evaluating in-memory OLAP workloads and contrasting the outcomes with those derived from traditional database systems employing tiered memory models.

3 EVALUATION

We present our experimental results in this section. As our main experimental finding advocates a CXL memory adoption of an interleaving manner, we begin by examining the impact on peak throughput across various memory interleaving ratios. Subsequent analyses compare the performance across different memory configurations, including fundamental memory access patterns such as sequential and random access followed by an assessment of selective access and an evaluation of in-memory radix partitioning. Finally, we extend our study by conducting an in-memory OLAP experimental analysis using the Star Schema Benchmark [33]. Our results demonstrate that, in contrast to employing the typical tier-memory model in conventional DBMS architectures, direct interleaving CXL memory with host-level DRAM yields significant performance enhancements for in-memory analytical workloads.

3.1 Experimental Setup

Our experimental platform is anchored on a dual-socket platform running Linux kernel version 5.19.17. Each socket hosts a 32-core Intel Sapphire Rapids CPU, with 64MB last-level cache (LLC) and 1024-entry translation lookaside buffer (TLB). The socket's capacity is enhanced by 128GB of DRAM, which is configured through four 32*G*B DDR5 DIMMs. To facilitate cross-NUMA communication, the system is equipped with a 96.0 GB/s Ultra Path Interconnect (UPI). Additionally, each DDR5 DIMM provides a memory bandwidth of 35.2 GB/s. We employ an initial CXL memory prototype from Montage [45], which integrates two DDR5 DIMMs as previously described. This prototype interfaces with the host system through an 8-lane PCIe Gen5 port, achieving a bandwidth of 31.5GB/s.

In alignment with prior research [34, 44], our system configuration is optimized for performance analysis by employing sub-NUMA clustering (SNC) mode. As an SNC node comprises a single memory channel with an aforementioned 32GB DDR5 DIMM situated, the theoretical maximum bandwidth of the host DRAM in our platform is 35.2GB/s. Since the SNC mode applies to the whole host system, the NUMA memory node is also configured as a single DDR5 memory DIMM with a capacity of 32GB (cf. Figure 2 for a simplified overview of our system architecture and the respective theoretical bandwidth limits of individual data path). We also disable hyper-threading, prefetching, interrupt requests (IRQ), kernel address space layout randomization (ASLR) as existing studies[11, 19, 41, 46, 50, 52, 53], and opt for a huge-page configuration to mitigate the potential for TLB thrashing penalty [6, 16].

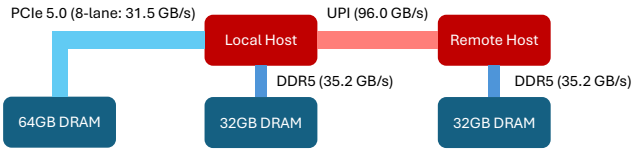

Figure 2: A simplified overview of our host system and the respective theoretical bandwidth limits of each data path.

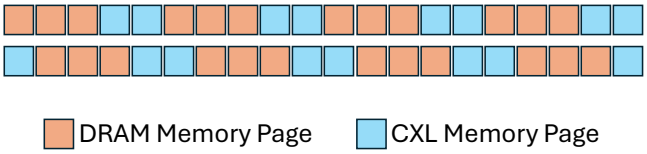

Figure 3: Memory Layout Illustration: DRAM-CXL Interleaving at a 3:2 Ratio.

We apply a Linux patch [51] to facilitate a range of memory interleaving ratios, enabling the distribution of memory pages between a processor-bound memory node and a processor-independent memory node [27][4]. This patch allows for adjusting the page interleaving ratio from 1 to 100. In our study, an interleaving ratio of $X : Y$ signifies a sequential memory configuration with X consecutive local host DRAM pages followed by Y consecutive CXL memory pages, as depicted in Figure 3 for a $3 : 2$ interleaving ratio layout.

Unless specified otherwise, our comparison encompasses four distinct memory configurations: local host memory access only (DRAM), remote NUMA memory access only (NUMA), CXL memory access only (CXL), and a DRAM-CXL interleaved setup (DRAM:CXL). Through experimentation, we determine that an interleaving ratio of $3 : 2$ consistently delivers optimal performance (Sections 3.2). As such, the subsequently presented results are compared with this optimized interleaving ratio. Our codes are publicly available at https://github.com/fukien/type3-pathway.

3.2 Memory Throughput Across Various Memory Interleaving Ratios

In order to reveal the potential benefits of bandwidth expansion from CXL memory, we interleave the local host DRAM with our genuine CXL type3 memory with various memory interleaving ratios. We adjust the interleaving ratio progressively from $100 : 0$ (pure DRAM) to $0 : 100$ (pure CXL), encompassing a total of 13 distinct ratios. To fully uncover the potential throughput gain, especially for the peak memory throughput, we use all 32 cores of a single socket to access memory sequentially. The experimental outcomes are presented in Figure 4.

[4]As remote NUMA memory is affiliated with CPUs, this interleaving patch cannot be applied across NUMA sockets.

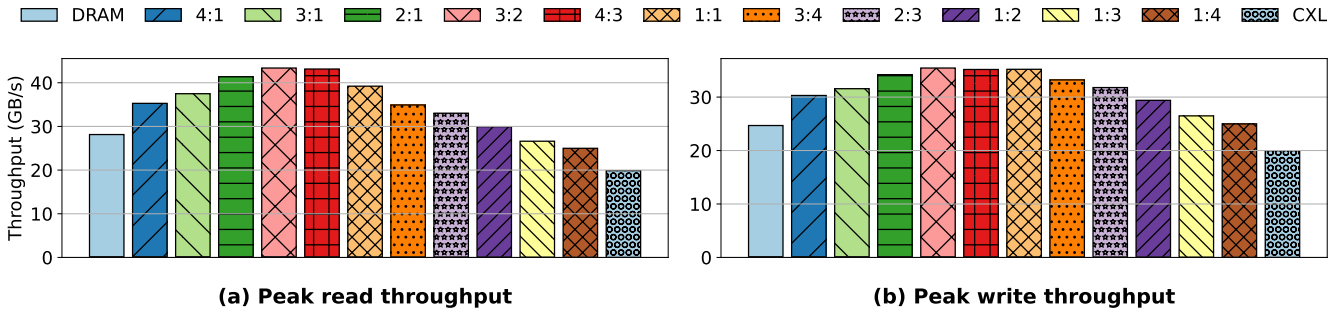

Figure 4: Peak memory throughput w.r.t. DRAM:CXL interleaving ratio ($X : Y$).

Figure 4 illustrates that interleaving DRAM with CXL at any non-zero ratio indeed delivers a better performance than a sole DRAM configuration. This is primarily attributed to the additional bandwidth resource provided by the PCIe interface. We can also observe that the non-interleaving configuration, i.e., the pure DRAM memory setting and a sole CXL memory setting deliver a throughput around 28.5GB/s and 20.1GB/s respectively. Recall that in our SNC-mode platform, the local DRAM bandwidth is capped at the theoretical maximum bandwidth of a single memory channel, which is 35.2GB/s (cf. Figure 2). A peak memory bandwidth of 28.5GB/s indicates that the current DDR technology (DDR5 in our platform) is not able to fully utilize the theoretical DDR channel bandwidth resource to a satisfactory level. Meanwhile, the CXL memory device, though equipped with a 35.2GB/s DDR5 memory and a 31.52GB/s PCIe Gen5 port, only achieves a peak memory throughput at around 20.1GB/s (the rightmost bar in Figure 4), which implies that the underlying CXL controller is the main bottleneck in preventing CXL memory achieving its maximum practical bandwidth [5].

By analyzing the peak memory throughput across various interleaving ratios, we find that an interleaving ratio of $3 : 2$ yields the highest throughput in both read and write scenarios. This interleaving ratio corresponds to the memory bandwidth ratios between DRAM and CXL, suggesting that it effectively balances workload distribution based on the bandwidth capabilities of the different memory technologies. Consequently, we maintain that interleaving DRAM and CXL at a ratio optimized for memory bandwidth can effectively maximize the utilization of system bandwidth resources.

3.3 Sequential Access Evaluation

Sequential access serves as a critical component across numerous in-memory workloads. Its distinctive access pattern, lacking both spatial and temporal locality, allows for the direct manifestation of the available memory bandwidth resources.

In order to measure the throughput of sequential access, we adjust the thread count and devise a microbenchmark, on which we issue three memory-semantic operations: load, store, and non-temporal store (as per "LOAD", "STORE", and "NT-STORE" respectively in Figure 5), sequentially accessing one billion 4-byte records.

Figure 5 presents the outcomes of our experiments, demonstrating the significant thread scalability in load bandwidth among all experimenting memory configurations. The "LOAD" throughput drastically ascends to the peak practical throughput of the respective memory setting. This indicates that the 'LOAD' operation exhibits strong scalability with respect to the number of threads. When examining "STORE" and "NT-STORE" operations, it is evident that "NT-STORE" operation achieves higher performance compared to conventional "STORE" operation. It should be noted that "NT-STORE" operation bypasses the processor's coherence cache domain, whereas "STORE" operation incurs additional overhead due to state modifications in processor-level cache. As a result, "NT-STORE" is more likely to achieve peak memory throughput. The overhead associated with cache coherence maintenance is evident in the lower performance of "STORE" and "NT-STORE" operations when accessing NUMA memory compared to CXL memory. Both memory configurations lie outside the local host processors' northbridge domain, leading to higher cache coherence maintenance overhead compared to local DRAM. However, CXL memory incurs lower coherence maintenance overhead due to its lack of processor affiliation. Memory nodes associated with processors experience additional overhead from exchanging or modifying cache coherence states between the processor-scope cache and the memory-resident cache coherence directory. This overhead can increase further with the number of processors. While it is not possible to explicitly measure this overhead due to Intel's non-disclosure of technical details, the cache coherence maintenance overhead has been documented in numerous related studies [24, 44]. Consequently, the store-related performance (both cache-temporal and non-cache-temporal) on conventional processor-attached NUMA nodes is unlikely to match that of non-processor-affiliated CXL memory nodes.

We can also observe from Figure 5 that memory interleaving between DRAM and CXL achieves optimal performance and strong thread scalability. Specifically, the aggregated "LOAD" throughput around 47.5GB/s, which is 1.61x higher than the local DRAM configuration, and significantly surpasses the theoretical bandwidth limit of a single DDR channel (35.2GB/s) and the limit of our 8-lane PCIe Gen5 ports (31.52GB/s). Additionally, "STORE" and "NT-STORE" operations demonstrate robust throughput beyond 6 threads. This suggests that interleaving effectively amalgamates the available bandwidth resources of various technologies, raising the practical bandwidth limit of the host system and benefiting all types of memory-semantic operations.

[5]We have received confirmation from Montage[45] that their current technology is capable of providing a practical bandwidth limited to $20 - 21$ GB/s.

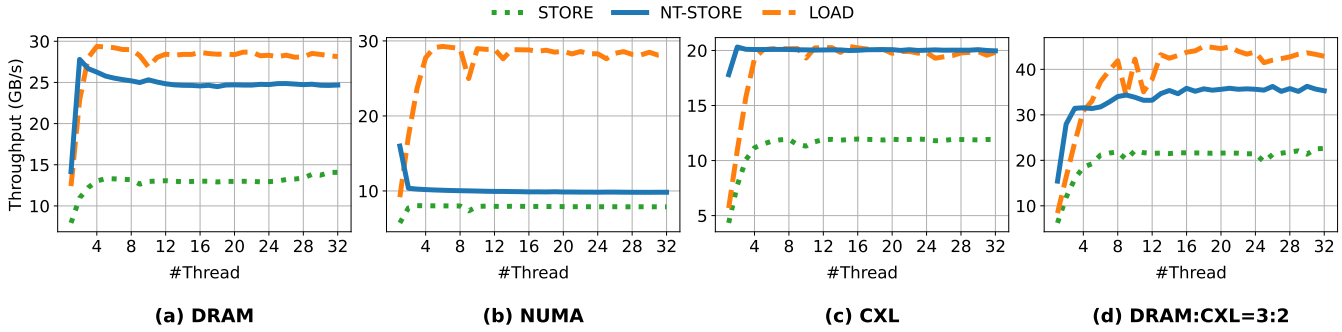

Figure 5: Sequential access throughput w.r.t. thread count with different memory configurations.

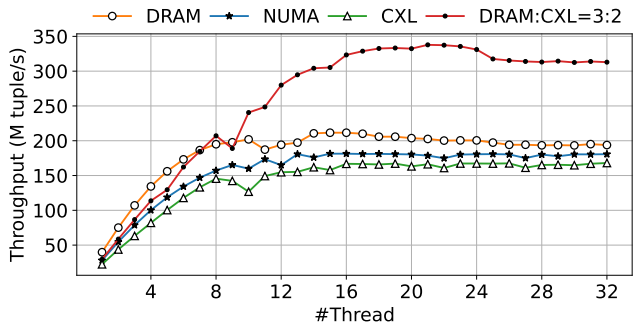

Figure 6: Join performance across various thread number and memory settings.

3.4 Random Access Evaluation: A Hash Join Perspective

Random access, alongside sequential access, is a pivotal building block of modern DBMS applications, particularly in the context of hash joins. The inherently irregular access pattern of random access results in frequent cache misses, making it highly susceptible to the intrinsic latency of the underlying memory technologies and diminishing its efficiency in fully saturating memory bandwidth.

Hash joins are widely reckoned as a major random access operation of in-memory DBMSs. We therefore, synthesize a representative workload as previous hash join studies [3, 4, 6, 8, 16, 17] to benchmark the DBMS random access performance with different memory configuration technologies. In particular, we construct a primary-key-foreign-key join workload featuring uniformly distributed 8-byte tuples. The build and probe sides of the workload have cardinalities of 256 million and 1 billion, respectively.

Figure 6 presents the results of the hash join experiment, demonstrating a consistent performance trend with that observed for sequential access (cf. Section 3.3). The DRAM:CXL interleaving configuration achieves the highest throughput, with a peak performance improvement of approximately 1.59x compared to the local DRAM setting. This suggests that, in addition to the bandwidth-friendly sequential access pattern, the additional bandwidth provided by PCIe also improves performance for irregular random access patterns. The performance enhancement continues to increase until the thread count reaches 16, in contrast to the throughput saturation point of eight threads observed with other memory configurations. This finding validates the previous claim that the amalgamation of bandwidth resources from interleaving DRAM and CXL effectively surpasses the bandwidth limitation of any single memory technology.

We now shift our focus to comparing NUMA memory and CXL memory. It is evident that NUMA memory maintains a marginal but stable advantage over CXL memory, regardless of thread count. This advantage is primarily attributed to the higher practical memory bandwidth provided by NUMA memory (cf. Figure 5). However, a detailed breakdown of the runtime for the explicit hash join phases reveals that the hash join build phase on NUMA memory is 26% slower than that on CXL memory. This further substantiates the previous claim (Section 3.3): CXL memory, as a non-processor-affiliated memory node, demonstrates superior memory write performance compared to conventional processor-affiliated NUMA memory. In contrast, the hash join probe phase, which primarily involves random memory reads, exhibits an opposite performance pattern: the probe phase on NUMA memory is 24% faster than on CXL memory. Given that the probe side is 3x larger than the build side (1 billion probe tuples versus 256 million build tuples), the probing phase dominates the overall hash join execution cost. As a result, the overall hash join performance on NUMA memory is slightly better than that on CXL memory.

3.5 Selective Access Evaluation

We continue our study by evaluating the selective read performance across different memory configurations. The selective access pattern emulates the access behavior in a DBMS column-style scan, where one or several selective predicates are applied. In a column-style DBMS, various tuple attributes are stored in individual memory arrays. If a tuple fails to meet a predicate based on its leading attributes, i.e., the first several columns, scanning the subsequent attributes (columns) of this tuple becomes unnecessary. As a result, such a tuple can be skipped in the subsequent column scans.

We synthesized a workload consisting of 2 billion 8-byte records to mimic this scanning pattern. In order to measure the peak performance of selective access, we utilized all 32 cores of a single socket to sequentially access these tuples. A specific number of records were skipped between every two consecutive record loads. We vary

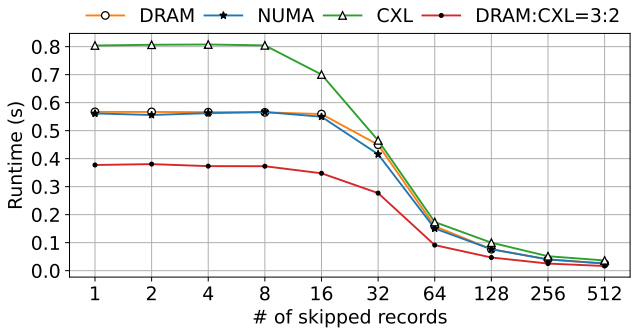

Figure 7: Selective read performance w.r.t. number of skipped records.

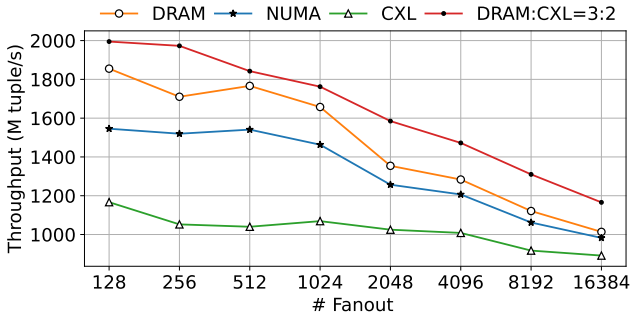

Figure 8: Partitioning performance w.r.t. fanout number.

this number from 1 to 512, and present experimental results in Figure 7.

In Figure 7, we discover that the execution time for all memory configurations decreases monotonically as the number of skipped records increases. This is because increasing the number of skipped records significantly reduces memory-level data movement. Consequently, the overall performance bottleneck shifts from memory-level operations to processor-level computation. Among all memory configurations, the DRAM:CXL interleaving configuration once again achieves the best performance. As indicated in previous subsections, this is primarily attributed to the superior aggregated memory bandwidth provided by the combination of local host DRAM and PCIe-interfaced CXL memory. Meanwhile, NUMA memory achieves performance comparable to local DRAM, due to their similar load throughput characteristics as DRAM (cf. Figure 5), allowing NUMA memory to perform nearly as well as local DRAM. In contrast, the CXL memory configuration does not achieve competitive performance in this experiment, as its maximum load bandwidth is constrained by the CXL internal controller and is approximately 30% lower than that of DRAM or NUMA memory (cf. Figure 5).

3.6 Radix Partitioning Evaluation

Partitioning is another important building block in modern in-memory DBMSs. The primary partitioning techniques include hash, range, and radix partitioning [35], with radix partitioning emerging as a predominant method in analytical processing tasks such as sorting [38], join [16, 17], and aggregation [32]. Therefore, we examine the performance of radix partitioning, with particular attention to the impact of underlying memory configurations on its performance.

Similar to the workload in selective access evaluation, we generate 1 billion 8-byte records and utilize 32 cores to perform radix partitioning for peak throughput measurement. Existing studies have identified that radix partitioning is significantly affected by TLB thrashing. Thus, we vary the number of partition fanouts from 64 to 16384 in increments of powers of two [6], and measure the throughput for each configuration.

The results are depicted in Figure 8. We can see that the partitioning throughput decreases as the partition fanout increases, which is consistent with existing studies that indicate a higher fanout leads to increasesd TLB thrashing penalty [3, 4, 6, 16, 17]. Comparing the performance of different memory configurations, we observe that the DRAM:CXL interleaving configuration delivers the highest throughput across all fanout settings. This performance advantage mainly stems from the sufficient bandwidth resource of combining local DRAM DDR channels and the PCIe ports. However, we do not see a significant performance gain, as observed in sequential or hash join assessments, against the local host DRAM setting (within 7.5% throughput gain), indicating that TLB thrashing heavily hinders the partitioning from effectively utilizing the available bandwidth resources. When comparing CXL with NUMA, we find NUMA memory consistently outperforms CXL memory across all experimented partitioning fanouts. The reason is that, a typical radix partitioning process involves two read passes and one write pass [16, 17, 39], with the write pass being heavily hindered by TLB thrashing. Henceforth, CXL's advantage in store operations does not manifest in radix partitioning, and the overall performance is more constrained by the maximum load throughput. As a consequence, the NUMA setting continuously outperforms CXL, and achieves throughput very close to the local DRAM setting due to their similar load throughput (cf. Figure 5).

3.7 In-Memory Analytical Workload Analaysis

Last but not least, we evaluate various memory configurations using real in-memory analytical workloads of the Star Schema Benchmark (SSB) [33]. SSB serves as a benchmark standard for assessing DBMS performance in data warehousing scenarios, featuring join operations between a large fact table and several smaller dimension tables, a setup commonly encountered in OLAP research.

To precisely examine the impact of different memory configurations on performance, we execute 13 SSB queries at a scale factor of 30 using a custom implementation based on query plans generated by HyPer [23] [7]. As per the previous selective performance assessment (cf. Section 3.5), we use all socket-level physical cores, i.e., 32 cores, to maximize the analytical processing throughput. Our goal is to explore optimal strategies for leveraging CXL memory to enhance in-memory analytical workloads. To this end,

[6] We do not study the impact of smaller partitioning fanouts for they are not practically beneficial.

[7] Our platform cannot accommodate larger workloads due to the limitations of the current 32GB local host DRAM.

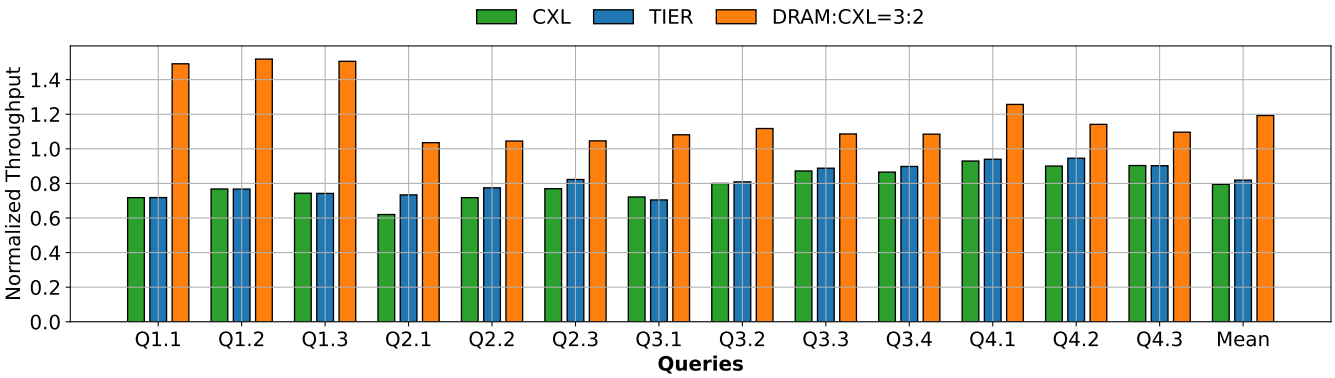

Figure 9: Star schema benchmark throughput (normalized against DRAM performance).

we examine the in-memory SSB benchmarking performance of DRAM, CXL, and the DRAM:CXL interleaving configuration. Additionally, we implement a popular conventional DBMS wisdom model [5, 30, 31, 41, 55] of a tiered-memory architecture (referred to as "TIER" in Figure 9), where local fast DRAM is used as a working "buffer", while the slower CXL memory functions similarly to a faster "disk", which requires data movement into the DRAM "buffer" for analytical query operations. Since different SSB queries have varying fraction factors [8], the execution times among these 13 queries differ significantly. For the purpose of better presentation, we use the DRAM execution throughput of each individual query as the baseline and normalize the query execution throughput of other memory configurations to this DRAM throughput. The results are shown in Figure 9.

We can observe from Figure 9 that the DRAM:CXL interleaving configuration demonstrates superior throughput, outperforming the local DRAM setup in all 13 SSB queries, with an average performance increase of approximately 1.19x (denoted as "Mean" in Figure 9). Although this improvement is noticeable, it is not highly significant and does not fully match the up to 1.61x performance gains observed in sequential and random memory access (Sections 3.3 and 3.4). This phenomenon is primarily attributed to the limited size of the dimension tables at a scale factor of 30. According to the query plans, selection pushdown is applied before hash table construction, allowing the hash tables at this scale factor to be almost entirely accommodated within the on-chip cache [9]. Consequently, the majority of SSB query operations hit the cache, minimizing the performance variance among the different memory configurations. Interestingly, queries Q1.1, Q1.2, and Q1.3 exhibit considerable performance enhancements (over 1.4x). This improvement is primarily attributed to their exclusive reliance on selection operations, without involving join operations, which allows them to benefit from the strong sequential access throughput of the DRAM:CXL interleaved setup.

In contrast, the DRAM-CXL tiered configuration yields less competitive results, with average throughput nearly mirroring that of the pure CXL memory setup. This phenomenon is also caused by the nearly cache-sized hash table, which significantly reduces the number of cache misses in subsequent hash probing. The hash probing performance, therefore, is no longer a primary performance bottleneck, and the placement of these cache-sized hash tables would not have much performance difference since they are mostly buffered in the processor-levle caches. Hence, the performance bottleneck shifts to the loading bandwidth in building or probing the hash tables. Since both the CXL memory configuration and the "TIERED" model initially place tables in CXL memory, the CXL memory "load" throughput dominates the execution phase of SSB query processing.

In summary, the DRAM:CXL interleaving configuration emerges as a promising solution for augmenting overall system bandwidth. In our testbed, a judiciously selected interleaving ratio, particularly one that aligns with the memory bandwidth ratios, can achieve performance gains of up to 1.61x compared to local DRAM setups. This demonstrated advantage encourages further exploration of CXL memory platforms and warrants a revisit of the existing design considerations of tiered-memory DBMS models, particularly in the context of large-scale heterogeneous memory systems.

4 CONCLUSION

In this paper, we perform a comprehensive experimental evaluation of in-memory analytical workloads on a genuine CXL memory platform. Our findings suggest that, beyond the traditional focus on expanding memory capacity, a CXL type3 memory device can also enhance the overall bandwidth of computing systems. Through a series of experimental investigations, we discover that the CXL type3 memory can deliver up to 1.61x in-memory analytical processing performance improvement in our real CXL platform if a proper interleaving of local host DRAM and CXL memory is configured. This performance benefit motivates us to consider more flexible utilization of CXL memory and calls for a critical revisit of the conventional DBMS tiered-memory model within the context of heterogeneous memory or computing platforms.

In the evolving landscape of hardware technology, CXL stands at a pivotal early stage. The synergy between CXL technology and in-memory database processing necessitates an in-depth analysis. Future advancements should aim to support a broad spectrum of

[8] The fraction factor denotes the proportion of the table that is retrieved for specific predicate combinations [40].
[9] Our platform's single socket has 64MB last-level cache (cf. Section 3.1).

database operations, including both analytical queries and transaction processing. These advancements should not only optimize for large memory capacity, but also reducing the total cost of ownership (TCO). The conclusion of our discourse could serve to explore this interplay, setting a forward-looking agenda for research and development in this arena.

ACKNOWLEDGMENTS

This project is supported by a grant funded by the Ministry of Education (Title: inPMdb: An in-Persistent Memory Database System; WBS No: A8000082-00-00). We are also grateful to the dedicated members from 4Paradigm and Montage for their guidance in establishing the experimental platform.

REFERENCES

[1] Bülent Abali, Richard J. Eickemeyer, Hubertus Franke, Chung-Sheng Li, and Marc Taubenblatt. 2015. Disaggregated and optically interconnected memory: when will it be cost effective? *CoRR* abs/1503.01416 (2015).

[2] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna T. Malladi, and Yang-Seok Ki. 2022. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *DaMoN*. ACM, 8:1–8:5.

[3] Martina-Cezara Albutiu, Alfons Kemper, and Thomas Neumann. 2012. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. *Proc. VLDB Endow.* 5, 10 (2012), 1064–1075.

[4] Cagri Balkesen, Jens Teubner, Gustavo Alonso, and M. Tamer Özsu. 2013. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware. In *ICDE*. IEEE Computer Society, 362–373.

[5] Vinay Banakar, Kan Wu, Yuvraj Patel, Kimberly Keeton, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2023. WiscSort: External Sorting For Byte-Addressable Storage. *Proc. VLDB Endow.* 16, 9 (2023), 2103–2116.

[6] Maximilian Bandle, Jana Giceva, and Thomas Neumann. 2021. To Partition, or Not to Partition, That is the Join Question in a Real System. In *SIGMOD Conference*. ACM, 168–180.

[7] Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali G. Saidi, Arkaprava Basu, Joel Hestness, Derek Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib Bin Altaf, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (2011), 1–7.

[8] Spyros Blanas, Yinan Li, and Jignesh M. Patel. 2011. Design and evaluation of main memory hash join algorithms for multi-core CPUs. In *SIGMOD Conference*. ACM, 37–48.

[9] Albert Cho, Anish Saxena, Moinuddin K. Qureshi, and Alexandros Daglis. 2023. A Case for CXL-Centric Server Processors. *CoRR* abs/2305.05033 (2023).

[10] CXL Consortium. 2022. Compute Express Link (CXL) Specification. https://www.computeexpresslink.org/_files/ugd/0c1418_1798ce97c1e6438fba818d760905e43a.pdf

[11] Intel Corporation. accessed in 2024. Intel® Memory Latency Checker (Intel® MLC). https://www.intel.com/content/www/us/en/download/736633/intel-memory-latency-checker-intel-mlc.html

[12] Thomas Coughlin and Objective Analysis Jim Handy. 2022. Persistent Memories: Without Optane,Where Would We Be? https://storagedeveloper.org/events/sdc-2022/agenda/session/324

[13] Peter Desnoyers, Ian F. Adams, Tyler Estro, Anshul Gandhi, Geoff Kuenning, Michael P. Mesnier, Carl A. Waldspurger, Avani Wildani, and Erez Zadok. 2023. Persistent Memory Research in the Post-Optane Era. In *DIMES@SOSP*. ACM, 23–30.

[14] Yehonatan Fridman, Suprasad Mutalik Desai, Navneet Singh, Thomas Willhalm, and Gal Oren. 2023. CXL Memory as Persistent Memory for Disaggregated HPC: A Practical Approach. In *SC Workshops*. ACM, 983–994.

[15] Kaisong Huang, Yuliang He, and Tianzheng Wang. 2022. The Past, Present and Future of Indexing on Persistent Memory. *Proc. VLDB Endow.* 15, 12 (2022), 3774–3777.

[16] Wentao Huang, Yunhong Ji, Xuan Zhou, Bingsheng He, and Kian-Lee Tan. 2023. A Design Space Exploration and Evaluation for Main-Memory Hash Joins in Storage Class Memory. *Proc. VLDB Endow.* 16, 6 (2023), 1249–1263.

[17] Wentao Huang, Yunhong Ji, Xuan Zhou, Bingsheng He, and Kian-Lee Tan. 2022. A Design Space Exploration and Evaluation for Main-Memory Hash Joins in Storage Class Memory [Technical Report]. https://www.comp.nus.edu.sg/~huang/assets/works/VLDB-2023/hashjoin-scm/main-tr.pdf

[18] Intel. 2022. Intel® Optane™ Persistent Memory. https://www.intel.sg/content/www/xa/en/architecture-and-technology/optane-dc-persistent-memory.html

[19] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amir Saman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *CoRR* abs/1903.05714 (2019).

[20] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2023. CXL-ANNS: Software-Hardware Collaborative Memory Disaggregation and Computation for Billion-Scale Approximate Nearest Neighbor Search. In *USENIX Annual Technical Conference*. USENIX Association, 585–600.

[21] Yunhong Ji, Wentao Huang, and Xuan Zhou. 2024. HeterMM: applying in-DRAM index to heterogeneous memory-based key-value stores. *Frontiers Comput. Sci.* 18, 4 (2024), 184612.

[22] Yunhong Ji, Wentao Huang, Xuan Zhou, Bingsheng He, and Kian-Lee Tan. 2024. TaC: An Anti-Caching Key-Value Store on Heterogeneous Memory Architectures. In *EDBT*. OpenProceedings.org, 474–487.

[23] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE*. IEEE Computer Society, 195–206.

[24] Wook-Hee Kim, Madhava Krishnan Ramanathan, Xinwei Fu, Sanidhya Kashyap, and Changwoo Min. 2021. PACTree: A High Performance Persistent Range Index Using PAC Guidelines. In *SOSP*. ACM, 424–439.

[25] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Comput. Archit. Lett.* 15, 1 (2016), 45–49.

[26] Dimitrios Koutsoukos, Raghav Bhartia, Michal Friedman, Ana Klimovic, and Gustavo Alonso. 2023. NVM: Is it Not Very Meaningful for Databases? *Proc. VLDB Endow.* 16, 10 (2023), 2444–2457.

[27] Michael Larabel. accessed in 2024. Linux Developers Discuss Improvements To Memory Tiering. https://www.phoronix.com/news/Linux-Better-Memory-Tiering

[28] Donghun Lee, Thomas Willhalm, Minseon Ahn, Suprasad Mutalik Desai, Daniel Booss, Navneet Singh, Daniel Ritter, Jungmin Kim, and Oliver Rebholz. 2023. Elastic Use of Far Memory for In-Memory Database Management Systems. In *DaMoN*. ACM, 35–43.

[29] Alberto Lerner and Gustavo Alonso. 2024. CXL and the Return of Scale-Up Database Engines. *CoRR* abs/2401.01150 (2024).

[30] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *ASPLOS (2)*. ACM, 574–587.

[31] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit O. Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *ASPLOS (3)*. ACM, 742–755.

[32] Ingo Müller, Peter Sanders, Arnaud Lacurie, Wolfgang Lehner, and Franz Färber. 2015. Cache-Efficient Aggregation: Hashing Is Sorting. In *SIGMOD Conference*. ACM, 1123–1136.

[33] Patrick E. O'Neil, Elizabeth J. O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *TPCTC (Lecture Notes in Computer Science)*, Vol. 5895. Springer, 237–252.

[34] Constantin Pohl and Kai-Uwe Sattler. 2018. Joins in a heterogeneous memory hierarchy: exploiting high-bandwidth memory. In *DaMoN*. ACM, 8:1–8:10.

[35] Orestis Polychroniou and Kenneth A. Ross. 2014. A comprehensive study of main-memory partitioning and its application to large-scale comparison- and radix-sort. In *SIGMOD Conference*. ACM, 755–766.

[36] Moinuddin K. Qureshi. 2014. Memory Scaling is Dead, Long Live Memory Scaling. https://hps.ece.utexas.edu/yale75/qureshi_slides.pdf

[37] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce L. Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Comput. Archit. Lett.* 10, 1 (2011), 16–19.

[38] Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey. 2010. Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort. In *SIGMOD Conference*. ACM, 351–362.

[39] Felix Martin Schuhknecht, Pankaj Khanchandani, and Jens Dittrich. 2015. On the Surprising Difficulty of Simple Things: the Case of Radix Partitioning. *Proc. VLDB Endow.* 8, 9 (2015), 934–937.

[40] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. 1979. Access Path Selection in a Relational Database Management System. In *SIGMOD Conference*. ACM, 23–34.

[41] Anil Shanbhag, Nesime Tatbul, David E. Cohen, and Samuel Madden. 2020. Large-scale in-memory analytics on Intel® Optane™ DC persistent memory. In *DaMoN*. ACM, 4:1–4:8.

[42] Debendra Das Sharma, Robert Blankenship, and Daniel S. Berger. 2023. An Introduction to the Compute Express Link (CXL) Interconnect. *CoRR* abs/2306.11227 (2023).

[43] Kevin Song, Jiacheng Yang, Sihang Liu, and Gennady Pekhimenko. 2023. Lightweight Frequency-Based Tiering for CXL Memory Systems. *CoRR* abs/2312.04789 (2023).

[44] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. In *MICRO*. ACM, 105–121.

[45] Montage Technology. 2023. CXL Memory eXpander Controller (MXC). https://www.montage-tech.com/MXC. https://www.montage-tech.com/MXC

[46] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2019. Persistent Memory I/O Primitives. In *DaMoN*. ACM, 12:1–12:7.

[47] Jacob Wahlgren, Maya B. Gokhale, and Ivy Bo Peng. 2022. Evaluating Emerging CXL-enabled Memory Pooling for HPC Systems. In *MCHPC@SC*. IEEE, 11–20.

[48] Jacob Wahlgren, Gabin Schieffer, Maya B. Gokhale, and Ivy Peng. 2023. A Quantitative Approach for Adopting Disaggregated Memory in HPC Systems. In *SC*. ACM, 60:1–60:14.

[49] Shucheng Wang, Qiang Cao, Ziyi Lu, Hong Jiang, and Yuanyuan Dong. 2022. PATS: Taming Bandwidth Contention between Persistent and Dynamic Memories. In *DATE*. IEEE, 885–890.

[50] Zixuan Wang, Xiao Liu, Jian Yang, Theodore Michailidis, Steven Swanson, and Jishen Zhao. 2020. Characterizing and Modeling Non-Volatile Memory Systems. In *MICRO*. IEEE, 496–508.

[51] Johannes Weiner. accessed in 2024. [PATCH] mm: mempolicy: N:M inter- leave policy for tiered memory nodes. https://lore.kernel.org/all/06f961992a2c119ed0904825d8ab3f2b2a2c682b.1637778851.git.hasanalmaruf@fb.com/

[52] Lingfeng Xiang, Xingsheng Zhao, Jia Rao, Song Jiang, and Hong Jiang. 2022. Characterizing the performance of intel optane persistent memory: a close look at its on-DIMM buffering. In *EuroSys*. ACM, 488–505.

[53] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. 2020. An Empirical Guide to the Behavior and Use of Scalable Persistent Memory. In *FAST*. USENIX Association, 169–182.

[54] Jifei Yi, Benchao Dong, Mingkai Dong, Ruizhe Tong, and Haibo Chen. 2022. MTˆ2: Memory Bandwidth Regulation on Hybrid NVM/DRAM Platforms. In *FAST*. USENIX Association, 199–216.

[55] Ying Zheng and Kian-Lee Tan. 2024. Sorting on Byte-Addressable Storage: The Resurgence of Tree Structure. *Proc. VLDB Endow.* 17, 6 (2024), 1487–1500.