Demystifying CXL Memory Bandwidth Expansion for Analytical Workloads

Georgiy Lebedev
EPFL
Lausanne, Switzerland
georgiy.lebedev@epfl.ch

Hamish Nicholson
EPFL
Lausanne, Switzerland
hamish.nicholson@epfl.ch

Musa Ünal
EPFL
Lausanne, Switzerland
musa.unal@epfl.ch

Sanidhya Kashyap
EPFL
Lausanne, Switzerland
sanidhya.kashyap@epfl.ch

Anastasia Ailamaki
EPFL
Lausanne, Switzerland
anastasia.ailamaki@epfl.ch

ABSTRACT

The bottleneck of modern data-intensive applications is increasingly shifting towards memory access, as the CPU core count growth continues to outpace the available DRAM bandwidth. The CXL protocol has emerged as a technological solution to this problem, offering software applications transparent cache-coherent access to additional memory resources. Recent work has shown that a proper interleaving configuration of DRAM and CXL memory can improve the performance of bandwidth-intensive analytical workloads, and that a single interleaving configuration can give optimal performance across any workloads.

In this paper, we revisit CXL memory bandwidth expansion for analytical workloads. We find that interleaving DRAM and CXL memory transparently to a state-of-the-art query execution engine Proteus, when running the SSB workload, degrades its average performance by 20%. We show that realizing the full potential of CXL memory bandwidth expansion requires efficient hardware utilization through maximizing memory-level parallelism and adapting to the workload's access patterns and characteristics.

VLDB Workshop Reference Format:
Georgiy Lebedev, Hamish Nicholson, Musa Ünal, Sanidhya Kashyap, and Anastasia Ailamaki. Demystifying CXL Memory Bandwidth Expansion for Analytical Workloads. VLDB 2025 Workshop: 16th International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures (ADMS25).

1 INTRODUCTION

Modern data-intensive applications face an increasingly severe memory bottleneck as performance limitations shift from computation to memory access. Figure 1 shows that available DRAM bandwidth per CPU core has stagnated across recent Intel® Xeon® server generations [11], with AMD EPYC® processors following a similar trend [4]. Although the number of CPU cores continues to increase, the DRAM bandwidth scalability has been fundamentally limited by the DDR scalability [31, 41]. This widening gap creates an imminent memory wall that threatens both performance and cost efficiency [4, 33]. The new memory wall will become a critical bottleneck in high-performance computing environments, particularly affecting data management systems that process large-scale analytical workloads [4].

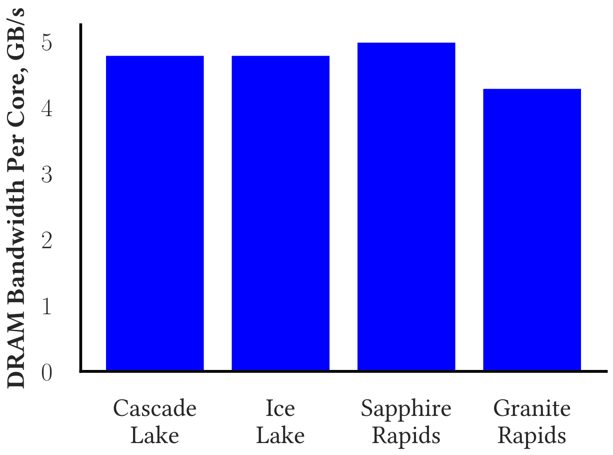


Figure 1: Theoretical maximum DRAM bandwidth per CPU core across latest 2-socket Intel® Xeon® server CPU generations.

Heterogeneous memory technologies [19, 39] present new opportunities to address the memory wall. The Compute eXpress Link (CXL) protocol [13] represents a significant advancement in memory expansion technology, providing software applications with transparent cache-coherent access to memory resources beyond traditional DRAM. CXL enables both memory capacity and bandwidth expansion, attracting adoption by major industry players including Microsoft Azure [26], Meta [30], Google [14], Alibaba [46], and SAP HANA [1, 2, 18, 23].

Previous research has demonstrated promising results for CXL memory capacity expansion [1, 2, 18, 23, 36]. However, bandwidth expansion remains less thoroughly investigated, particularly for data management systems. The effectiveness of bandwidth expansion varies significantly based on hardware characteristics (available bandwidth, memory-level parallelism, etc.) and workload properties (memory intensity, access patterns, etc.).

Recent studies have primarily evaluated CXL memory bandwidth expansion for HPC and AI workloads [33, 37, 41, 45]. Data management systems present distinct optimization opportunities since they can tailor memory usage patterns to specific workloads rather than relying on generic operating system solutions. Prior research suggests that proper interleaving of DRAM and CXL memory enhances performance for bandwidth-intensive analytical workloads, proposing that a single interleaving configuration suffices across diverse workloads [16].

This paper provides a comprehensive reevaluation of CXL memory bandwidth expansion for analytical workloads. Through systematic evaluation of the Star Schema Benchmark (SSB) [32] using a state-of-the-art query execution engine Proteus [5, 6, 20, 21] across multiple hardware configurations, we demonstrate that CXL memory bandwidth expansion actually degrades performance for most queries, with individual query improvements limited to 10% in optimal cases.

We provide three key insights for designing data management systems that effectively leverage CXL memory bandwidth expansion:

- **Transparent memory interleaving degrades analytical performance.** Transparent interleaving of DRAM and CXL memory degrades SSB workload's average performance by 20% (Section 5). Disabling hardware prefetching reduces performance degradation to 7%, revealing that current prefetching mechanisms operate inefficiently with interleaved DRAM and CXL memory configurations.
- **Workload-adaptive interleaving is essential.** No single memory interleaving configuration achieves optimal performance across workload variations, with optimal ratios varying by up to 88% based on workload parameters (Section 6). This necessitates adaptive approaches that respond to runtime workload characteristics.
- **Hardware utilization determines bandwidth expansion benefits.** Memory bandwidth underutilization reduces potential performance gains by up to 60% (Section 7). Our analysis demonstrates that hyper-threading improves performance of bandwidth expansion, while sub-NUMA clustering is insignificant. While hardware prefetching does not always work efficiently with interleaved DRAM and CXL memory configurations, it is crucial for getting performance improvements. Maximizing hardware utilization through efficient memory-level parallelism is essential for realizing the potential of CXL memory bandwidth expansion.

We discuss the implications of our insights for data management system design in Section 8 and conclude in Section 9.

2 BACKGROUND

Compute eXpress Link (CXL) [40] is an open industry standard interconnect technology that enables high-bandwidth, low-latency connections between processors and various devices, including accelerators, memory expanders, and storage systems. Built upon the PCIe physical layer, CXL has emerged as a critical technology for memory expansion in data-intensive computing environments, offering particular advantages for database systems and memory-bound analytical workloads. The CXL specification defines three complementary protocols that work together to enable coherent memory access. CXL.io leverages existing PCIe functionality for device discovery, configuration, and basic I/O operations. CXL.cache provides cache-coherent access to host memory from attached devices. CXL.mem enables hosts to access device-attached memory through standard load/store instructions with full cache coherence.

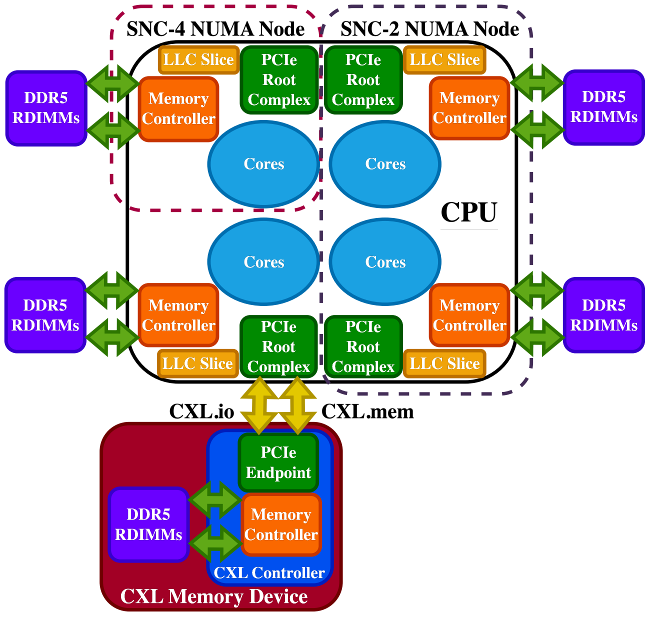


Figure 2: CXL memory expansion platform topology.

This paper focuses on CXL memory expansion capabilities, which utilize the CXL.io and CXL.mem protocols. CXL.io handles the initial interface establishment and device management between the CPU and CXL memory device, while CXL.mem provides the cache-coherent memory access semantics that make CXL memory transparent to software applications. This transparency allows existing applications to benefit from expanded memory resources without modification. Further on, we refer to CXL memory simply as CXL.

3 RELATED WORK

CXL research has rapidly evolved from early hardware emulation studies to comprehensive evaluations using commercial devices, establishing a diverse research landscape that spans system-level memory management, application-specific optimization strategies, and performance characterization methodologies. The field has primarily concentrated on memory capacity expansion applications, where CXL serves as a cost-effective alternative to traditional DRAM scaling, while bandwidth expansion capabilities have received comparatively limited attention despite their significant potential for memory-intensive workloads.

CXL performance characterization. Early CXL research relied heavily on hardware emulation, which introduced significant limitations in understanding real-world performance characteristics. The availability of commercial CXL devices has enabled more accurate performance analysis and revealed important discrepancies with emulated results.

Sun et al. [41] conducted one of the first evaluations using genuine CXL hardware, revealing substantial differences between emulated and real CXL performance. Their work provides insights into fundamental performance characteristics and latency-bandwidth tradeoffs of CXL memory devices and highlights the opportunity of bandwidth expansion for bandwidth-intensive applications Subsequent studies have provided deeper insights into CXL performance characteristics. Liu et al. [27, 28] have performed a systematic characterization, performance modeling and performance analysis of CXL. Additional research has characterized the hardware parallelism [47] and interference [29] of CXL.

Transparent memory management. The systems community has developed various approaches for transparent CXL memory management without requiring application modifications. These solutions operate at the hypervisor and operating system levels to efficiently manage heterogeneous memory tiers.

Hypervisor-level solutions include memory pooling systems that aggregate CXL memory across multiple nodes. Microsoft's Pond [26] implements CXL-based memory pooling for cloud platforms, while other research explores virtualized CXL memory environments [49] that enable dynamic memory allocation across virtual machines.

Operating system approaches focus on intelligent page placement and migration policies. Meta's TPP [30] implements proactive page demotion and promotion strategies. Google's TMTS [14] provides application-transparent memory tiering that maintains service-level objectives across diverse workloads. Other notable systems include MEMTIS [24], vTMM [38], and Colloid [44], each contributing different optimization strategies for tiered memory management. Overall, these systems focus on designing efficient tiered memory management transparently to userspace applications.

Explicit memory management. The data management research community has focused on efficiently managing CXL memory explicitly, since data management applications have more precise knowledge of their current and future workloads. Riekenbrauck et al. have proposed a tiered buffer manager design for CXL [36]. Further, SAP HANA researchers have proposed various data placement techniques [1, 2, 23] for efficient memory expansion and memory pooling. Finally, various memory-conscious algorithms [3, 18] and architectures [4, 15, 25] for CXL have been proposed. Overall, prior research work has focused on the memory capacity expansion aspects of CXL. In this paper, we study the memory bandwidth expansion aspect.

Memory bandwidth expansion research. CXL memory bandwidth expansion research has primarily focused on artificial intelligence and high-performance computing workloads [33, 37, 45]. Recent work by Huang et al. [16] specifically examined bandwidth expansion for analytical workloads through interleaving DRAM and CXL. However, their evaluation was limited to a single hardware configuration using sub-NUMA clustering and the same interleaving ratio across all workloads.

This paper revisits CXL memory bandwidth expansion for analytical workloads by providing comprehensive analysis across multiple hardware configurations, examining hardware prefetching effects, analyzing memory bandwidth utilization, and evaluating a wide range of memory interleaving configurations across different workloads.

4 EXPERIMENTAL SETUP

This section presents our experimental methodology for evaluating CXL memory bandwidth expansion across a range of hardware configurations. Our evaluation addresses three fundamental research questions:

- **DRAM bandwidth scaling.** How does systematically varying available DRAM bandwidth per-core via sub-NUMA clustering alter the effectiveness of CXL memory bandwidth expansion?
- **Prefetching behavior.** What role does hardware prefetching play in the performance of CXL memory bandwidth expansion?
- **Memory interleaving ratios.** How does weighted memory interleaving affect performance across different workloads?

To answer these questions systematically, we evaluate multiple hardware configurations with varying DRAM bandwidth capabilities, examine both enabled and disabled hardware prefetching modes, and test a comprehensive range of interleaving ratios between DRAM and CXL.

4.1 Experimental Platform

We conduct our experiments on a server running Linux kernel version 6.9.0 with support for weighted memory interleaving [7]. The server has a dual-socket 48-core Intel® Xeon® Platinum 8468 Sapphire Rapids CPU supporting the CXL 1.1 protocol with 2 hyperthreads per physical core, a 210 MB last-level cache (LLC), and 8 DDR5 DRAM channels with 1x32 GB DDR5 RDIMMs per channel. Throughout our evaluation, we utilize a single CPU socket with all 48 physical cores, disabling hyper-threading, except for specific scalability experiments detailed in Section 7. All of the data is equally partitioned between the execution threads. For CXL memory expansion, we use a SMART® CXA-4F1W device [42], which is a CXL 2.0 Type 3 memory expander, populated with 4x64 GB DDR5 RDIMMs. This device connects to the utilized CPU socket via a 16-lane (x16) PCIe 5.0 interface. Consequently, the CXL link operates under the CXL 1.1 protocol, utilizing the PCIe 5.0 physical layer's data rate of 32 GT/s per lane.

Controlling DRAM bandwidth. We use Intel® Sub-NUMA Clustering (SNC) [12] to systematically vary available DRAM bandwidth. SNC partitions the CPU's memory subsystem into multiple NUMA domains, each with dedicated DRAM channels (Figure 2). Our Sapphire Rapids CPU integrates four chiplets, each containing 12 cores and 2 DDR5 channels. We evaluate three configurations:

- **Full Socket.** SNC disabled, exposing all chiplets as a single NUMA node (8 DRAM channels).
- **SNC-2.** Two chiplets per socket (4 DRAM channels).
- **SNC-4.** One chiplet per socket node (2 DRAM channels).

Memory interleaving implementation. We use the `/sys/kernel/mm/mempolicy/weighted_interleave/`

Table 1: Peak sustainable memory bandwidth estimations in GB/s.

Hardware Configuration	DRAM (Sysbench)	DRAM (Intel® MLC)	CXL (Sysbench)	CXL (Intel® MLC)
Full Socket, hardware prefetching enabled	252.9	268.6	41.3	43.0
Full Socket, hardware prefetching disabled	184.0	194.5	44.5	46.4
SNC-2, hardware prefetching enabled	125.7	133.3	39.5	41.1
SNC-2, hardware prefetching disabled	106.9	107.1	44.3	46.2
SNC-4, hardware prefetching enabled	58.4	66.8	40.5	41.9
SNC-4, hardware prefetching disabled	48.0	51.0	44.4	46.3

Table 2: Idle memory latency estimations in nanoseconds.

Hardware Configuration	DRAM	CXL
Full Socket	111	229
SNC-2	105	226
SNC-4	96	212

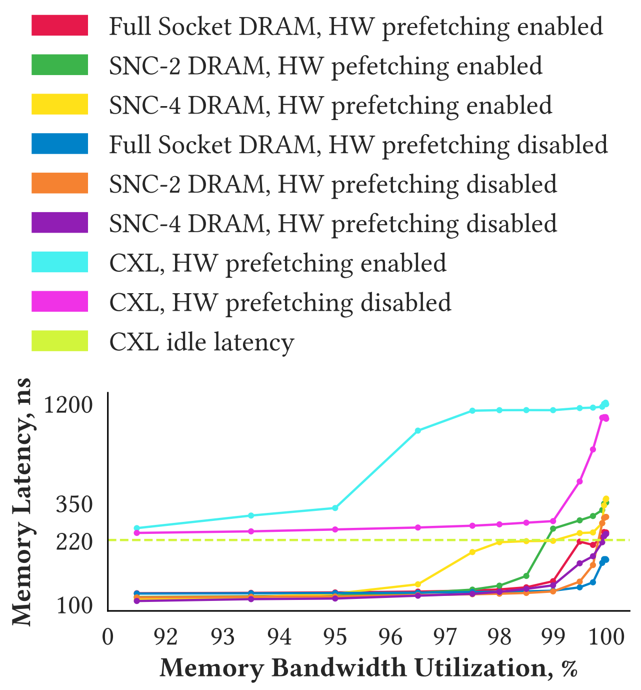


Figure 3: Loaded memory latency as a function of memory bandwidth utilization on logarithmic scales.

interface and the `MPOL_WEIGHTED_INTERLEAVE` memory allocation policy available in recent Linux kernel versions. A weighted memory interleaving configuration X:Y means that every X consecutive OS memory pages allocated on DRAM are followed by Y consecutive OS memory pages allocated on CXL. We use 2 MiB huge pages to minimize the effects of TLB pressure. We use the percentage of memory bound pipeline slots and the percentage of memory bandwidth-bound and latency-bound stalls from the Intel® VTune™ Profiler Top-down Microarchitecture Analysis Method [10, 48] to characterize the performance implications of interleaving DRAM and CXL from the CPU perspective. Further on, we refer to interleaved DRAM and CXL as DRAM:CXL.

4.2 Memory Characteristics Heterogeneity

To interpret bandwidth expansion results, we need to understand the fundamental performance characteristics of DRAM and CXL across our hardware configurations. We characterize both bandwidth and latency properties using established benchmarking tools.

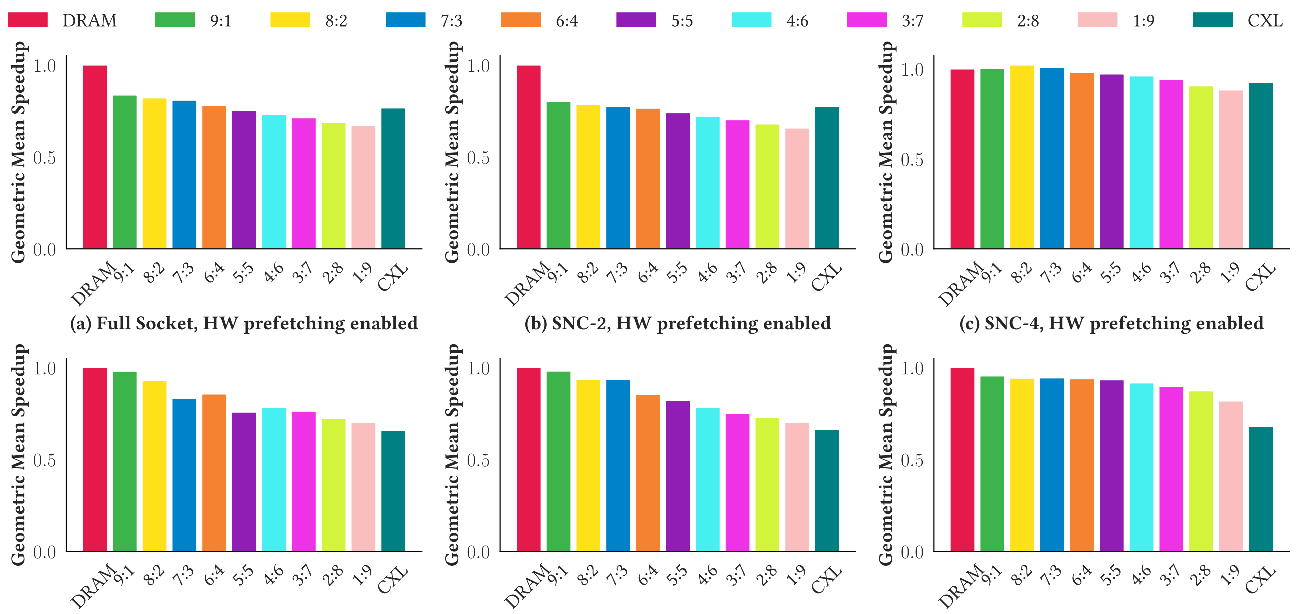
Bandwidth characterization. We observe the peak sustainable memory bandwidths of DRAM and CXL for each hardware configuration using two benchmarks: the bandwidth matrix test of Intel® Memory Latency Checker (Intel® MLC) [8] and the local scope sequential read memory test of sysbench [22]. Both benchmarks allocate 1 GB of memory per thread entirely on DRAM or CXL and access it using all 48 physical CPU cores. The results are presented in Table 1.

The benchmarks report different results, and the peak bandwidth ratio between DRAM and CXL does not remain the same. To understand this discrepancy, we also use the Intel® Performance Counter Monitor (Intel® PCM) [9] to measure the memory bandwidth using hardware performance counters while running sysbench, and it reports the same results as Intel® MLC. Thus, we conclude that the discrepancy comes from the fact that the Intel® MLC reports raw hardware memory bandwidth, while sysbench reports software throughput.

Latency characterization. We measure the latency of DRAM and CXL using the idle and loaded latency tests of Intel® MLC. They characterize the latency under, correspondingly, low and high bandwidth utilization. The results are presented in Table 2 and Figure 3.

As the bandwidth utilization increases, the loaded DRAM latency exceeds the idle CXL latency in all hardware configurations, except for the full socket configuration with hardware prefetching disabled. But even before the memory bandwidth is fully utilized, the loaded CXL latency increases up to 5x times when hardware prefetching is enabled. Disabling hardware prefetching delays this effect until the memory bandwidth is saturated. While the SNC-4 configuration has DRAM bandwidth comparable to CXL, the loaded latency increases only 3x times. Similar effects of memory latency inflation under load have also been observed in recent work on DRAM [17] and CXL [27, 43]. CXL thus requires more attention than DRAM due to its higher sensitivity to latency inflation, as overloading it causes significantly more severe performance degradation.

These findings suggest that CXL memory bandwidth expansion can improve a bandwidth-intensive workload's performance by reducing both its latency- and bandwidth- bound stall cycles.



(a) Full Socket, HW prefetching enabled (b) SNC-2, HW prefetching enabled (c) SNC-4, HW prefetching enabled

(d) Full Socket, HW prefetching disabled (e) SNC-2, HW prefetching disabled (f) SNC-4, HW prefetching disabled

Figure 4: Geometric mean speedups across the SSB workload for different DRAM:CXL interleaving weights normalized against the DRAM-only configuration.

Conversely, CXL can hurt performance when either the DRAM bandwidth is underutilized or the CXL bandwidth is saturated.

5 SSB WORKLOAD

To evaluate CXL memory bandwidth expansion for analytical workloads, we conduct experiments using the Star Schema Benchmark (SSB) [32], a standard data warehouse benchmark. We use SSB scale factor 100 to ensure the dataset fits within the memory capacity of all hardware configurations[1].

For query execution, we use Proteus [5, 6, 20, 21], a state-of-the-art pipelined parallel in-memory analytical engine with just-in-time LLVM-based code generation. We use hash joins for equi-joins, where build-side table cardinalities result in cache-resident hash tables for all queries. We analyze out-of-cache hash joins separately in Section 6.2.

We evaluate SSB queries across multiple memory interleaving configurations and compute geometric mean speedups normalized against the DRAM-only baseline. Figure 4 presents aggregate results, while Figure 5 shows individual query performance.

5.1 Average Workload Performance

Across all SSB queries, CXL memory bandwidth expansion degrades performance for all hardware configurations except SNC-4 with hardware prefetching enabled. The performance impact is substantial: for full socket and SNC-2 configurations with hardware prefetching enabled, even the lightest CXL interleaving (9:1) causes a 20% performance degradation. Profiling of the 9:1 interleaving configuration reveals that more than 44% of memory stall cycles are memory latency-bound, while memory-bound pipeline slots increase by at least 1.4 percentage points relative to the DRAM-only configuration.

However, disabling hardware prefetching dramatically reduces this penalty, limiting performance degradation to at most 7% for the lightest interleaving configurations (9:1, 8:2). This finding indicates that hardware prefetching operates inefficiently with heterogeneous memory configurations.

Supporting this conclusion, CXL-only often outperforms interleaving configurations with heavier CXL weights. Only the SNC-4 configuration with hardware prefetching enabled shows improvement: the 8:2 interleaving reduces bandwidth-bound memory stall cycles by up to 4.1 percentage points and memory-bound pipeline slots by up to 1.5 percentage points compared to the DRAM-only configuration, achieving a modest 2% speedup.

5.2 Individual Query Analysis

To understand per-query performance implications, we show individual SSB query runtimes in Figure 5 for the best-performing memory interleaving configurations compared to pure DRAM and CXL configurations. All runtimes are normalized against the DRAM-only baseline.

For full socket and SNC-2 configurations, individual queries benefit from bandwidth expansion only when hardware prefetching is disabled. Notably, the CXL-only outperforms interleaving for nearly half of all queries in these configurations. Disabling hardware prefetching significantly reduces performance variation

[1]We observed similar results when evaluating scale factor 1000 on the full socket configuration.

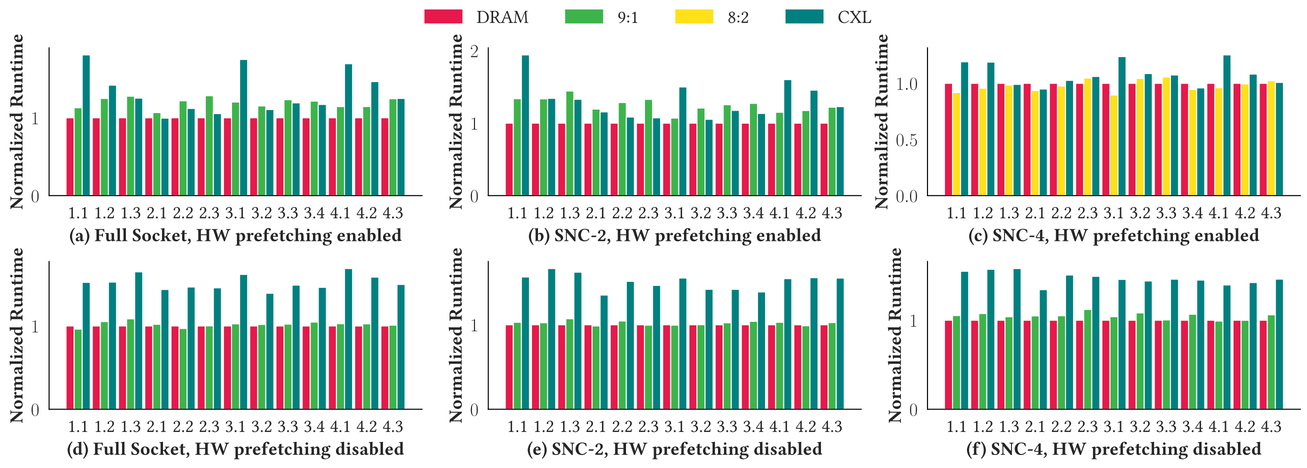


Figure 5: SSB query runtimes for different memory configurations normalized against the DRAM-only configuration.

across queries in all hardware configurations, with the gap between DRAM-only and optimal weighted interleaving configurations shrinking to at most 12%.

The SNC-4 configuration with hardware prefetching enabled shows different behavior: the 8:2 interleaving reduces memory-bound pipeline slots by up to 3.2 percentage points compared to the DRAM-only configuration, delivering a 10% speedup for query 3.1. These results consistently demonstrate that hardware prefetching operates inefficiently with memory interleaving, supporting our general findings.

6 WORKLOAD DIVERSITY

The SSB workload we evaluated in Section 5 is comprised of four query groups, and, within each query group, the query's selectivity decreases with the rank. For instance, query 1.3 is more selective than query 1.2, which is more selective than query 1.1. Furthermore, the execution of each query goes through multiple execution phases with distinct access patterns and characteristics, such as: (i) sequential read input scans; (ii) sequential write output flushes; (iii) hash joins; (iv) filtering with various selectivities; (v) group-by hash partitioning with various fanouts. A typical analytical workload has a plethora of different execution phases with distinct characteristics.

This suggests that a single DRAM:CXL interleaving configuration cannot give optimal performance across all workloads. To understand how the optimal interleaving configuration can change depending on workload characteristics, we analyze the performance of four distinct execution phases typical for analytical workloads on a range of interleaving weights.

6.1 Sequential Access

We start by analyzing the performance of sequential read and write microbenchmarks, the former modeling an input scan, and the latter an output flush. We sequentially read (write) a 32 GB array of 8-byte integers, measuring the throughput as the amount of data read (written) per second. We normalize the throughputs against the DRAM-only configuration. The results are presented in Figure 6.

The optimal interleaving weights for the SNC-4 configuration are the same for both read and write configurations when hardware prefetching is enabled, but in all other cases they are different. The optimal interleaving configurations for sequential reads reduce bandwidth-bound memory stall cycles by up to 2.5 percentage points and memory-bound pipeline slots by up to 31 percentage points relative to the DRAM-only configuration, giving up to 72% speedup in the SNC-4 configuration. For sequential writes, the optimal interleaving configurations reduce store-bound pipeline stalls by up to 2.5 percentage points and memory-bound pipeline slots reduce by up to 2.5 percentage points compared to the DRAM-only configuration, giving a 76% speedup in the SNC-4 configuration.

While the ratio of peak sustainable bandwidth estimations for DRAM and CXL is similar to the sequential read and write optimal interleaving weights for the SNC-4 configuration with hardware prefetching enabled, it significantly differs in all other cases. Both of these findings are also confirmed by recent work [37] that shows that the optimal interleaving weights change depending on the read-write ratio of the workload. We conclude that the correlation between the ratio of peak DRAM and CXL bandwidth estimations and the sequential read and write optimal interleaving weights is weak, and that the optimal weights are determined by both the workload and the hardware.

6.2 Hash Join

Next, we analyze the performance of a hash join microbenchmark. To model the build phase of the hash join, we sequentially access a pre-generated 2 GB array of unique random 8-byte integer indices to fill in a 16 GB array of hash table buckets consisting of 8-byte integer keys and values. To maximize the memory bandwidth intensity of this microbenchmark, we use an identity hash function which is also a perfect hash function in this scenario, i.e., there are no collisions. To model the probe phase, we sequentially access the array of random indices to probe the array of hash table buckets 2^32 times, wrapping around the array of random indices, ensuring that the probe phase exceeds 95% of the execution time of the hash join. The sizings we choose ensure that none of the data gets completely

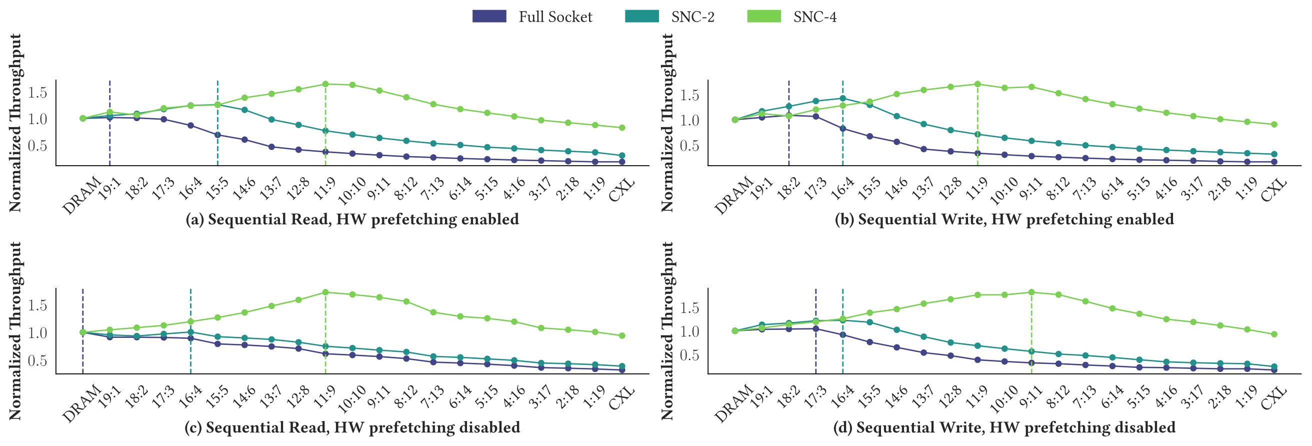


Figure 6: Sequential read and write throughputs for different DRAM:CXL interleaving weights normalized against the DRAM-only configuration.

cached, allowing us to stress the memory accesses. We estimate the throughput of the build phase as the number of values written per second, and the throughput of the probe phase as the number of values probed per second. For both phases, we normalize the throughputs against the DRAM-only configuration. The results are presented in Figure 7.

The optimal interleaving weights for both phases are close to those for sequential writes and reads respectively, and they do not depend on hardware prefetching. Even though we do random writes (reads) during the build (probe) phase, we sequentially read the random indices at which we write (read). Hence, the workload is still amenable to hardware prefetching, and we attribute the similarity between the optimal interleaving weights for scans and hash joins to this. The results diverge only for the full socket configuration with hardware prefetching enabled, which shows the DRAM bandwidth underutilization discussed in Section 7 further exacerbated by indirect memory accesses. The optimal interleaving configurations for the build phase reduce memory-bound pipeline slots by up to 11.4 percentage points compared to the DRAM-only configuration, giving a 82% speedup in the SNC-4 configuration. For the probe phase, the optimal interleaving configurations reduce memory-bound pipeline slots by up to 10 percentage points relative to the DRAM-only configuration, giving a 70% speedup in the SNC-4 configuration.

6.3 Filtering

Next, we analyze the performance of a filtering microbenchmark to model a selective access execution phase. We sequentially read a 32 GB array of 8-byte integers, varying the number of tuples skipped between two consecutive accesses (selectivity) as powers of 2, and estimate the throughput as the amount of filtered data read per second. We normalize the throughputs against the DRAM-only configuration for each selectivity. The results are presented in Figure 8.

All hardware configurations benefit from interleaving for at least 1 selectivity value, regardless of hardware prefetching. The optimal interleaving configurations depend on the selectivity, and they tend to shift to CXL as the selectivity increases. The optimal interleaving configuration reduces the bandwidth-bound memory stall cycles by at most 12 percentage points and the memory-bound pipeline slots by at most 8.6 percentage points compared to the DRAM-only configuration, giving a 90% speedup for selectivity 512 in the SNC-4 configuration. In this hardware configuration, CXL-only reduces the bandwidth-bound memory stall cycles by at 7.7 percentage points and the memory-bound pipeline slots by at most 1.7 percentage points relative to the DRAM-only configuration, giving a 24% speedup for selectivity 512. The optimal interleaving ratio can change by up to 55% as in the SNC-4 configuration with hardware prefetching enabled (from 6:4 to 4:6).

6.4 Radix Partitioning

Finally, we study the performance of a partitioning microbenchmark to model a group-by hash partitioning execution phase. We perform radix partitioning [34] on a 16 GB array of 8-byte integers, varying the number of partitions (fanout) as powers of 2, and estimate the throughput as the number of partitioned tuples per second. We normalize the throughputs against the DRAM-only configuration for each fanout. The results are presented in Figure 9.

There is no hardware configuration for which a single memory interleaving configuration would be optimal across all fanouts, except for the full socket configuration with hardware prefetching disabled, which does not benefit from interleaving at all. The optimal configurations drastically change as the fanout increases, and configurations with heavier CXL weights become optimal for some of the fanouts. Since partitioning also involves writing data, this may be a consequence of the higher cache coherence overhead of CPU-associated memory when accessing it under high contention. For the full socket configuration with hardware prefetching enabled, the optimal interleaving configuration reduces latency-bound memory stall cycles by at most 2.7 percentage points and memory-bound pipeline slots by at most 0.7 percentage points compared to the DRAM-only configuration, giving a 32% speedup for fanout 8192.

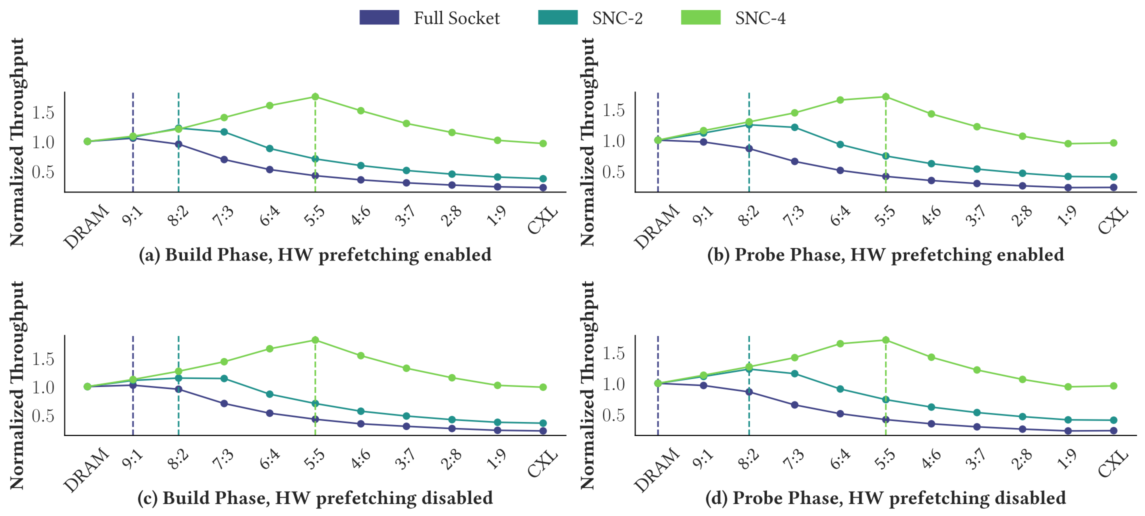


Figure 7: Hash join build and probe phases throughputs for different DRAM:CXL interleaving weights normalized against the DRAM-only configuration.

For other hardware configurations, the optimal interleaving configuration reduces the store-bound memory stalls by up to 6.8 percentage points and the memory-bound pipeline slots by at most 4.3 percentage points relative to the DRAM-only configuration, giving a 71% performance improvement for fanout 8192 in the SNC-4 configuration. The optimal interleaving ratio can change by up to 88% as in the SNC-2 configuration with hardware prefetching enabled (from 9:1 to 5:5).

7 MEMORY BANDWIDTH UTILIZATION

To understand the reasons for the poor performance of CXL memory bandwidth expansion when running the SSB workload, we analyze the throughput scalability of the sequential read microbenchmark used in Section 6.1 with respect to the number of utilized CPU cores. In the SNC configurations, we first utilize the CPU cores local to the SNC NUMA node. For memory interleaving, we use the optimal interleaving configurations for sequential reads found in Section 6.1. We omit memory interleaving for the full socket configuration with hardware prefetching disabled, since it does not improve performance for it. We evaluate throughput improvements below as the the absolute improvements over the DRAM-only configuration normalized against the CXL configuration to account for the DRAM bandwidths of different hardware configurations. The results are presented in Figure 10.

7.1 Scalar Reads

For scalar reads in the area of physical CPU cores, bandwidth expansion improves the throughput only when the number of utilized CPU cores saturates the DRAM bandwidth. This is evident by the flattening of the throughput curve of the DRAM-only configuration for Subfigures (b), (c) and (f). However, the hardware configurations in Subfigures (a), (d) and (e) scale up to all physical CPU cores, and, apparently, the CPU does not have enough physical cores to saturate the DRAM bandwidth. In the SNC-4 configuration, the throughput of the DRAM-only configuration degrades after 24 of its local cores are utilized. We attribute it to the NUMA effects of accessing the memory of the SNC-4 node from other chiplets, and to the contention over its memory interconnect. Conversely, the throughput continues to grow in the memory interleaving configuration. While interleaving improves the throughput of the SNC-2 (SNC-4) configurations by to 90% (96%), it improves the throughput of the full socket configuration by at most 7.3%. This suggests that there is headroom for improvement in the full socket configuration coming from memory bandwidth underutilization.

7.2 SIMD Reads

To demonstrate the memory bandwidth underutilization, we use SIMD reads. We use the 512-bit load instruction from the AVX-512 instruction set. Figure 10 shows that SIMD reads are capable of saturating the DRAM bandwidth even in the full socket configuration, when hardware prefetching is enabled, and in the SNC-2 configuration with hardware prefetching disabled. This again corresponds to the flattening of the throughput curve of the DRAM-only configuration in all of the Subfigures except for (d). For the full socket configuration with hardware prefetching disabled, even with the smallest of sampled interleaving weights, 19:1, there is no performance improvement. With memory interleaving, SIMD reads do not improve throughput compared to scalar reads in the SNC-2 and SNC-4 configurations with hardware prefetching enabled, in which scalar reads can saturate the DRAM bandwidth on their own, however they allow to acheive the same peak throughput using less CPU cores. Furthermore, SIMD reads improve the throughput by up to 90% (96%) of the theoretical potential in the SNC-2 (SNC-4) configuration with hardware prefetching disabled and by at least 67.4% of it in the full socket configuration with hardware prefetching enabled.

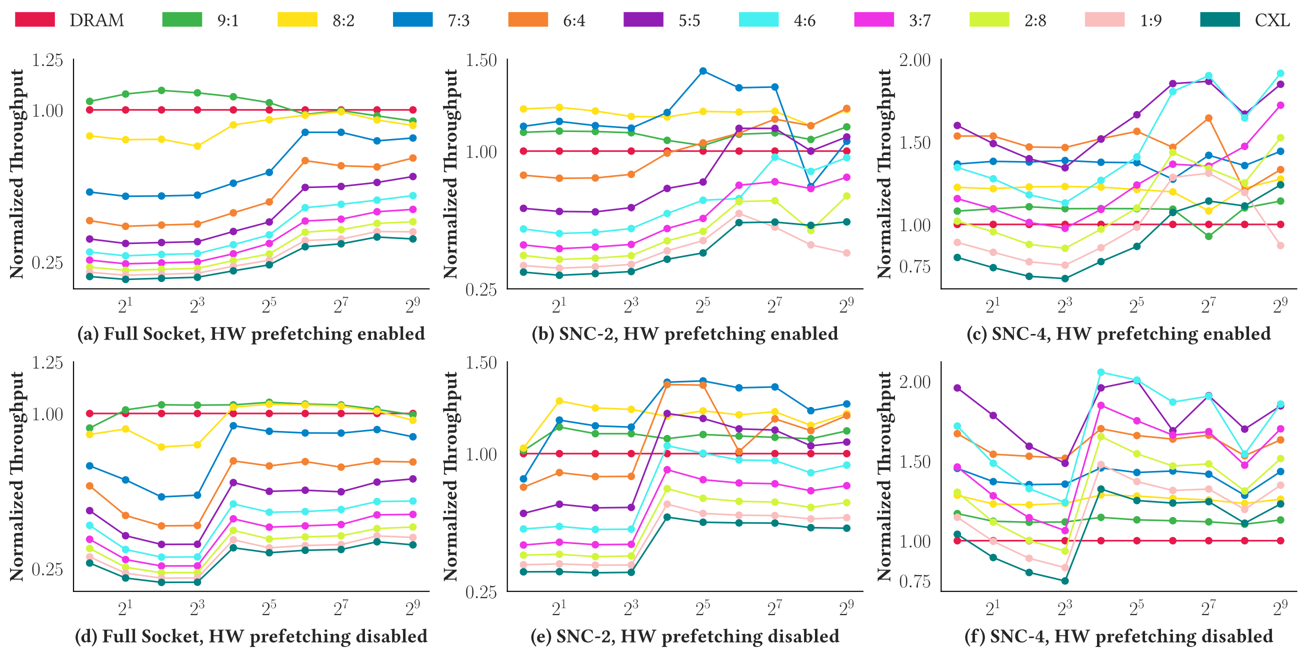


Figure 8: Filtering throughputs for different DRAM:CXL interleaving weights as a function of selectivity on a logarithmic scale normalized against the DRAM-only configuration.

7.3 Hardware Prefetching

Hardware prefetching increases the benefits gained from bandwidth expansion in the SNC-2 configuration by 5% and enables bandwidth expansion in the full socket configuration. However, in the SNC-4 configuration, disabling hardware prefetching allows to get a 6% higher peak throughput. As the memory interconnect contention increases with number of utilized CPU cores in the SNC-2 and SNC-4 configurations, hardware prefetching stabilizes the performance (Subfigures (b) and (c)) and prevents it from significantly degrading (Subfigures (e) and (f)).

7.4 Hyper-Threading

Another way to demonstrate the underutilization of the memory bandwidth is hyper-threading. Hyper-threading allows to multiplex the hardware resources of a CPU core between two execution threads, allowing to improve the memory bandwidth utilization, when it is underutilized. We scale all our hardware configurations to hyper-threads, adding them in the same order as physical CPU cores. For the full socket configuration with hardware prefetching enabled, scalar reads are able to saturate the DRAM bandwidth, and memory interleaving improves the throughput by at most 64% compared to DRAM. However, in the rest of the hardware configurations, using memory interleaving with hyper-threading either improves the throughput by at most 10% in the SNC-4 configuration with hardware prefetching disabled and all available CPU cores utilized, or degrades it. In these configurations, the memory resources can be saturated without hyper-threading.

7.5 Sub-NUMA Clustering

SNC exposes fine-grained memory localization domains and was designed to improve performance when such locality can be achieved. In our experiments with different hardware configurations, the finer-grained SNC configurations consistently achieve a higher throughput improvement compared to the full socket configuration. However, when SIMD reads are used in the full socket configuration with hardware prefetching enabled, the potential for memory bandwidth expansion is also saturated, since the throughput curve flattens. Finer-grained configurations achieve a higher throughput improvement than the full socket configuration mainly because they have less DRAM bandwidth per core. This means fewer cores are needed to saturate DRAM bandwidth, so more cores use the CXL bandwidth. Thus we conclude that realizing the full potential of bandwidth expansion does not require using SNC.

8 DISCUSSION

Based on our experimental findings, we identify three key design principles for data management systems to effectively leverage CXL memory bandwidth expansion. We discuss each principle along with concrete implementation strategies and their limitations.

8.1 Hardware-Consciousness

Transparent DRAM and CXL interleaving degrades average performance of the SSB workload by 20% (Section 5), demonstrating that a hardware-conscious approach is essential for effective CXL bandwidth expansion.

DRAM and CXL exhibit fundamentally different performance characteristics that vary dynamically with load on the memory

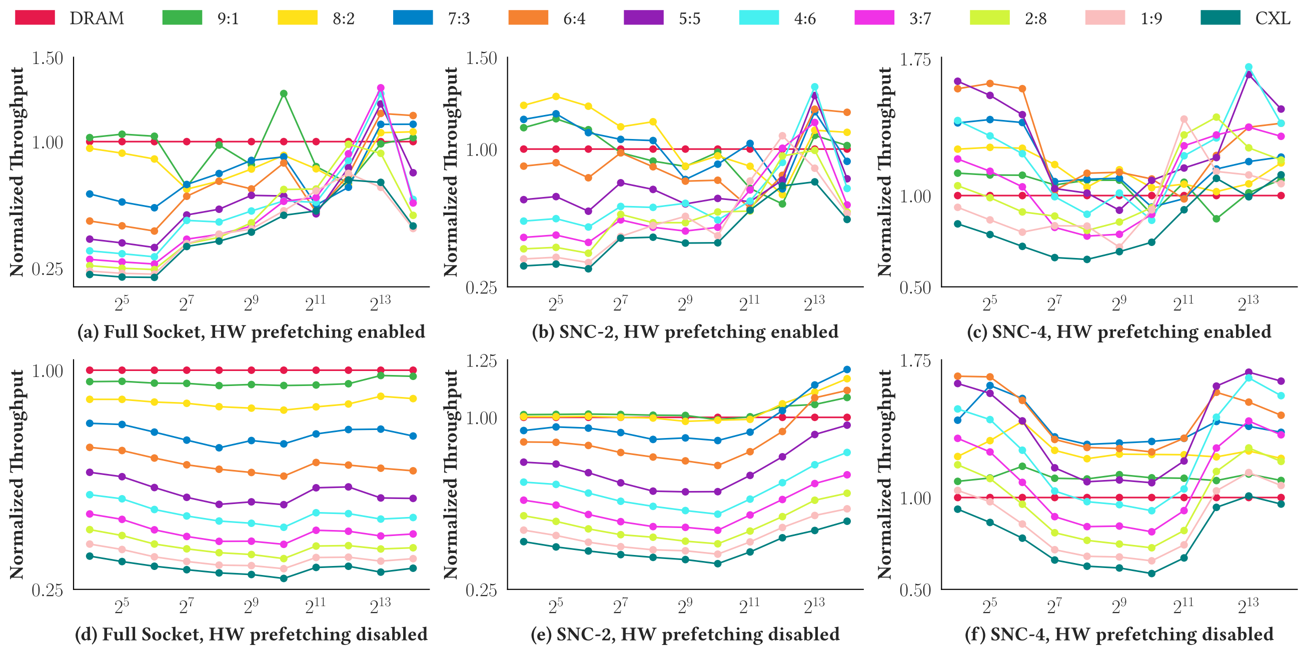


Figure 9: Partitioning throughputs for different DRAM:CXL interleaving weights as a function of fanout on a logarithmic scale normalized against the DRAM-only configuration.

system (Section 4.2). CXL can degrade performance, relative to a DRAM baseline, when DRAM bandwidth is underutilized or when CXL bandwidth becomes saturated. Additionally, our results show that hardware prefetching operates inefficiently with interleaved memory configurations, particularly under high memory bandwidth utilization.

Implementation strategies. Data management systems should implement two key mechanisms. First, *dynamic performance monitoring* using hardware performance counters to track memory-bound pipeline slots, bandwidth-bound stalls, and latency-bound stalls in real-time. Second, *adaptive prefetching control* that selectively disables hardware prefetching for workload phases that exhibit poor prefetching efficiency with CXL or interleaved memory.

Concrete approaches include extending existing buffer pool managers to track access patterns per page and use these in combination with hardware performance counters to make timely decisions about data placement and prefetching control.

8.2 Workload-Adaptivity

Our evaluation reveals that optimal memory interleaving configurations vary dramatically across workload characteristics, with optimal ratios changing by up to 88% depending on the workload characteristics (Section 6). This variability, also observed in HPC and AI workloads [37], necessitates runtime adaptation rather than static configuration.

Analytical workloads present unique adaptation opportunities due to their structured execution. Each physical query operator (e.g., scan, filter, hash join build and probe phases, partition) exhibits distinct optimal interleaving ratios that also depend on their parameters, e.g., filter selectivity and partitioning fanout. For example, our results show that filtering operations require progressively heavier CXL weighting as selectivity increases, while sequential operations perform optimally with interleaving configurations that approximate the DRAM:CXL bandwidth ratio measured with standard tools.

Implementation strategies. Systems should implement *operator aware memory management* with three components. First, *workload characterization* during query compilation that analyzes operators to predict access patterns, data volumes, and selectivity estimates. Second, *runtime adaptation mechanisms* that monitor execution progress and adjust memory interleaving weights between execution phases. Third, *predictive data migration* that proactively moves data based on upcoming operator requirements, for example, building a hash table and then moving it for the probe phase of a hash join.

Practical approaches include extending query optimizers to generate memory placement hints, implementing lightweight profiling during initial query executions to build adaptation models, and developing cost-based models that balance migration overhead against potential performance gains. The adaptation frequency should align with operator boundaries to minimize disruption, with more aggressive adaptation for long-running operators that process large data volumes.

Adaptation overhead considerations. Runtime adaptation introduces computational overhead and potential data movement costs. Systems should implement hysteresis mechanisms to avoid thrashing between configurations and establish minimum execution time thresholds before triggering adaptations.

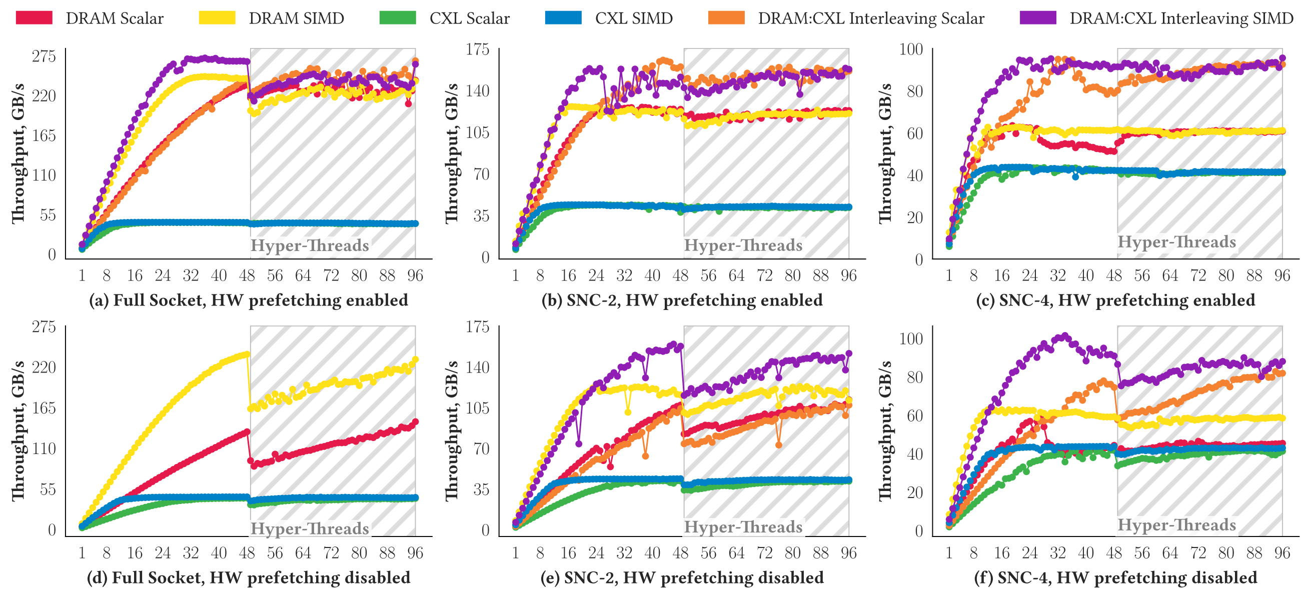


Figure 10: Memory access throughput scaling across DRAM and CXL configurations as a function of CPU core count.

8.3 Efficient Memory Bandwidth Utilization

CXL memory bandwidth expansion provides benefits only when DRAM bandwidth is a bottleneck (Section 7). The relationship between bandwidth utilization and performance is nuanced. Higher utilization enables CXL bandwidth expansion benefits, but can simultaneously increase memory latency. Our experiments show that memory contention causes DRAM latency to increase up to 2× and CXL latency to increase up to 5× under high bandwidth utilization with hardware prefetching enabled (Section 4.2).

Implementation strategies. Systems should focus on three techniques to optimize bandwidth utilization. First, *vectorized execution* using SIMD instructions for sequential access patterns. Second, *improve memory-level parallelism* for random access patterns, for example, through software prefetching with coroutines [35] which improves memory-level parallelism without the interference effects observed with hardware prefetching. Third, *intelligent thread scheduling* that leverages hyper-threading when memory bandwidth is underutilized but avoids it when bandwidth is saturated.

Concrete approaches include implementing vectorized scan operators that process multiple cache lines per instruction, developing coroutine-based hash join implementations that overlap memory accesses with computation, and creating dynamic thread pool managers that adjust parallelism based on real-time bandwidth utilization measurements.

Bandwidth utilization monitoring. Systems should monitor both achieved bandwidth (via performance counters) and memory-bound pipeline slots to distinguish between insufficient parallelism and memory bandwidth saturation. The optimal target utilization varies by hardware configuration.

9 CONCLUSION

In this paper, we revisit CXL memory bandwidth expansion for bandwidth-intensive analytical workloads. We find that interleaving DRAM with CXL transparently to a state-of-the-art query execution engine when running the SSB workload degrades its average performance by 20%, and we show that realizing the full potential of CXL memory bandwidth expansion requires efficient hardware utilization through maximizing memory-level parallelism and adapting to the workload's access patterns and characteristics.

REFERENCES

[1] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. 2022. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *Proceedings of the 18th International Workshop on Data Management on New Hardware* (Philadelphia, PA, USA) *(DaMoN '22)*. Association for Computing Machinery, New York, NY, USA, Article 8, 5 pages. https://doi.org/10.1145/3533737.3535090

[2] Minseon Ahn, Thomas Willhalm, Norman May, Donghun Lee, Suprasad Mutalik Desai, Daniel Booss, Jungmin Kim, Navneet Singh, Daniel Ritter, and Oliver Rebholz. 2024. An Examination of CXL Memory Use Cases for In-Memory Database Management Systems Using SAP HANA. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 3827–3840. https://doi.org/10.14778/3685800.3685809

[3] Alexander Baumstark, Marcus Paradies, Kai-Uwe Sattler, Steffen Kläbe, and Stephan Baumann. 2024. So Far and yet so Near - Accelerating Distributed Joins with CXL. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) *(DaMoN '24)*. Association for Computing Machinery, New York, NY, USA, Article 7, 9 pages. https://doi.org/10.1145/3662010.3663449

[4] Yannis Chronis, Anastasia Ailamaki, Lawrence Benson, Helena Caminal, Jana Gičeva, Dave Patterson, Eric Seldar, and Lisa Wu Wills. 2025. Databases in the Era of Memory-Centric Computing.

[5] Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. HetExchange: Encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines. *Proc. VLDB Endow.* 12, 5 (2019), 544–556. https://doi.org/10.14778/3303753.3303760

[6] Periklis Chrysogelos, Panagiotis Sioulas, and Anastasia Ailamaki. 2019. Hardware-conscious Query Processing in GPU-accelerated Analytical Engines. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org.

http://cidrdb.org/cidr2019/papers/p127-chrysogelos-cidr19.pdf

[7] Jonathan Corbet. Last accessed 26.05.2025. *Weighted interleaving for memory tiering*. https://lwn.net/Articles/948037/

[8] Intel® Corporation. Last accessed 26.05.2025. *Intel® Memory Latency Checker (Intel® MLC)*. https://www.intel.com/content/www/us/en/download/736633/intel-memory-latency-checker-intel-mlc.html

[9] Intel® Corporation. Last accessed 26.05.2025. *Intel® Performance Counter Monitor (Intel® PCM)*. https://www.intel.com/content/www/us/en/developer/articles/tool/performance-counter-monitor.html

[10] Intel® Corporation. Last accessed 26.05.2025. *Intel® VTune™ Profiler Top-down Microarchitecture Analysis Method*. https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2025-0/top-down-microarchitecture-analysis-method.html

[11] Intel® Corporation. Last accessed 26.05.2025. *Intel® Xeon® Processors Product Specifications*. https://www.intel.com/content/www/us/en/ark/products/series/595/intel-xeon-processors.html

[12] Intel® Corporation. Last accessed 26.05.2025. *Technical Overview Of The 4th Gen Intel® Xeon® Scalable Processor Family*. https://www.intel.com/content/www/us/en/developer/articles/technical/fourth-generation-xeon-scalable-family-overview.html

[13] Debendra Das Sharma, Robert Blankenship, and Daniel Berger. 2024. An Introduction to the Compute Express Link (CXL) Interconnect. *ACM Comput. Surv.* 56, 11, Article 290 (July 2024), 37 pages. https://doi.org/10.1145/3669900

[14] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, Brian Morris, Chiranjit Mukherjee, Jingliang Ren, Greg Thelen, Paul Turner, Carlos Villavieja, Parthasarathy Ranganathan, and Amin Vahdat. 2023. Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 727–741. https://doi.org/10.1145/3582016.3582031

[15] Yunyan Guo and Guoliang Li. 2024. A CXL- Powered Database System: Opportunities and Challenges. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 5593–5604. https://doi.org/10.1109/ICDE60146.2024.00447

[16] Wentao Huang, Mo Sha, Mian Lu, Yuqiang Chen, Bingsheng He, and Kian-Lee Tan. 2024. Bandwidth Expansion via CXL: A Pathway to Accelerating In-Memory Analytical Processing. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org. https://vldb.org/workshops/2024/proceedings/ADMS/ADMS24_01.pdf

[17] Akanksha Jain, Hannah Lin, Carlos Villavieja, Baris Kasikci, Chris Kennelly, Milad Hashemi, and Parthasarathy Ranganathan. 2024. Limoncello: Prefetchers for Scale. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 577–590. https://doi.org/10.1145/3620666.3651373

[18] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2024. Bridging Software-Hardware for CXL Memory Disaggregation in Billion-Scale Nearest Neighbor Search. *ACM Trans. Storage* 20, 2, Article 10 (Feb. 2024), 30 pages. https://doi.org/10.1145/3639471

[19] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (High Bandwidth Memory) DRAM Technology and Architecture. In *2017 IEEE International Memory Workshop (IMW)*. 1–4. https://doi.org/10.1109/IMW.2017.7939084

[20] Manos Karpathiotakis, Ioannis Alagiannis, and Anastasia Ailamaki. 2016. Fast queries over heterogeneous data through engine customization. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 972–983. https://doi.org/10.14778/2994509.2994516

[21] Manos Karpathiotakis, Ioannis Alagiannis, Thomas Heinis, Miguel Branco, and Anastasia Ailamaki. 2015. Just-In-Time Data Virtualization: Lightweight Data Management with ViDa. In *Seventh Biennial Conference on Innovative Data Systems Research, CIDR 2015, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2015/Papers/CIDR15_Paper8.pdf

[22] Alexey Kopytov. Last accessed 26.05.2025. *Scriptable database and system performance benchmark*. https://github.com/akopytov/sysbench

[23] 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 *Proceedings of the 19th International Workshop on Data Management on New Hardware* (Seattle, WA, USA) *(DaMoN '23)*. Association for Computing Machinery, New York, NY, USA, 35–43. https://doi.org/10.1145/3592980.3595311

[24] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. MEMTIS: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 17–34. https://doi.org/10.1145/3600006.3613167

[25] Alberto Lerner and Gustavo Alonso. 2024. CXL and the Return of Scale-Up Database Engines. *Proceedings of the VLDB Endowment* 17, 10 (June 2024), 2568–2575. https://doi.org/10.14778/3675034.3675047

[26] 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 *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 574–587. https://doi.org/10.1145/3575693.3578835

[27] Jinshu Liu, Hamid Hadian, Yuyue Wang, Daniel S. Berger, Marie Nguyen, Xun Jian, Sam H. Noh, and Huaicheng Li. 2025. Systematic CXL Memory Characterization and Performance Analysis at Scale. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 1203–1217. https://doi.org/10.1145/3676641.3715987

[28] Jinshu Liu, Hamid Hadian, Hanchen Xu, Daniel S. Berger, and Huaicheng Li. 2024. Dissecting CXL Memory Performance at Scale: Analysis, Modeling, and Optimization. arXiv:2409.14317 [cs.OS] https://arxiv.org/abs/2409.14317

[29] Shunyu Mao, Jiajun Luo, Yixin Li, Jiapeng Zhou, Weidong Zhang, Zheng Liu, Teng Ma, and Shuwen Deng. 2024. CXL-Interference: Analysis and Characterization in Modern Computer Systems. arXiv:2411.18308 [cs.AR] https://arxiv.org/abs/2411.18308

[30] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 742–755. https://doi.org/10.1145/3582016.3582063

[31] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2022. A modern primer on processing in memory. In *Emerging computing: from devices to systems: looking beyond Moore and Von Neumann*. Springer, 171–243.

[32] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 237–252.

[33] Vinicius Petrucci, Eishan Mirakhur, Nikesh Agarwal, Su Wei Lim, Vishal Tanna, Rita Gupta, and Mahesh Wagh. 2024. *CXL Memory Expansion: A Closer Look on Actual Platform*. White Paper. Micron and AMD. https://www.micron.com/content/dam/micron/global/public/products/white-paper/cxl-memory-expansion-a-close-look-on-actual-platform.pdf

[34] 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 *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 755–766. https://doi.org/10.1145/2588555.2610522

[35] George Psaropoulos, Thomas Legler, Norman May, and Anastasia Ailamaki. 2019. Interleaving with Coroutines: A Systematic and Practical Approach to Hide Memory Latency in Index Joins. *The VLDB Journal* 28, 3 (2019), 451–471. https://doi.org/10.1007/s00778-018-0533-6

[36] Niklas Riekenbrauck, Marcel Weisgut, Daniel Lindner, and Tilmann Rabl. 2024. A Three-Tier Buffer Manager Integrating CXL Device Memory for Database Systems. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. 395–401. https://doi.org/10.1109/ICDEW61823.2024.00063

[37] Rohit Sehgal, Vishal Tanna, Vinicius Petrucci, and Anil Godbole. 2024. *Optimizing System Memory Bandwidth with Micron CXL® Memory Expansion Modules on Intel® Xeon® 6 Processors*. White Paper. Micron and Intel. https://www.micron.com/content/dam/micron/global/public/products/white-paper/optimize-system-bandwidth-for-hpc-ai-micron-cxl-intel-xeon-whitepaper.pdf

[38] Sai Sha, Chuandong Li, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. 2023. vTMM: Tiered Memory Management for Virtual Machines. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 283–297. https://doi.org/10.1145/3552326.3587449

[39] Debendra Das Sharma. 2023. Compute Express Link (CXL): Enabling Heterogeneous Data-Centric Computing With Heterogeneous Memory Hierarchy. *IEEE Micro* 43, 2 (2023), 99–109. https://doi.org/10.1109/MM.2022.3228561

[40] Debendra Das Sharma, Robert Blankenship, and Daniel S. Berger. 2024. An introduction to the compute express link (CXL) interconnect. *Acm Computing Surveys* 56, 11 (2024), 290:1–290:37. https://doi.org/10.1145/3669900

[41] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL Memory with

Genuine CXL-Ready Systems and Devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 105–121. https://doi.org/10.1145/3613424.3614256

[42] Smart® Modular Technologies. Last accessed 26.05.2025. *CXL® Memory Solutions*. https://www.smartm.com/product/list/196

[43] Musa Unal, Vishal Gupta, Yueyang Pan, Yujie Ren, and Sanidhya Kashyap. 2025. Tolerate It if You Cannot Reduce It: Handling Latency in Tiered Memory. In *Proceedings of the 20th Workshop on Hot Topics in Operating Systems (HotOS XX)*.

[44] Midhul Vuppalapati and Rachit Agarwal. 2024. Tiered Memory Management: Access Latency is the Key!. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (Austin, TX, USA) *(SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 79–94. https://doi.org/10.1145/3694715.3695968

[45] Xi Wang, Jie Liu, Jianbo Wu, Shuangyan Yang, Jie Ren, Bhanu Shankar, and Dong Li. 2025. Exploring and Evaluating Real-world CXL: Use Cases and System Adoption. arXiv:2405.14209 [cs.PF] https://arxiv.org/abs/2405.14209

[46] Xinjun Yang, Yingqiang Zhang, Hao Chen, Feifei Li, Gerry Fan, Yang Kong, Bo Wang, Jing Fang, Yuhui Wang, Tao Huang, Wenpu Hu, Jim Kao, and Jianping Jiang. 2025. Unlocking the Potential of CXL for Disaggregated Memory in Cloud-Native Databases. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25)*. Association for Computing Machinery, New York, NY, USA, 689–702. https://doi.org/10.1145/3722212.3724460

[47] Yujie Yang, Lingfeng Xiang, Peiran Du, Zhen Lin, Weishu Deng, Ren Wang, Andrey Kudryavtsev, Louis Ko, Hui Lu, and Jia Rao. 2025. Architectural and System Implications of CXL-enabled Tiered Memory. arXiv:2503.17864 [cs.AR] https://arxiv.org/abs/2503.17864

[48] Ahmad Yasin. 2014. A Top-Down method for performance analysis and counters architecture. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 35–44. https://doi.org/10.1109/ISPASS.2014.6844459

[49] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. 2024. Managing memory tiers with CXL in virtualized environments. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation* (Santa Clara, CA, USA) *(OSDI'24)*. USENIX Association, USA, Article 3, 20 pages.