Hash Joins Meet CXL: A Fresh Look

Wentao Huang
National University of Singapore
huangwentao@u.nus.edu

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

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

ABSTRACT

Compute Express Link (CXL) has emerged as a promising technology for expanding memory capacity and bandwidth in data-intensive systems. Recent studies advocate interleaving CXL memory with local DRAM to create a new interleaved memory tier, thereby increasing aggregate bandwidth for workloads that are constrained by memory bandwidth. Following this trend, many systems place data in the interleaved tier to accelerate query processing. However, for most applications where data is stored on CXL memory, existing works fail to account for the additional data movement overhead to load the data into the interleaved memory. In this paper, we revisit this design decision through the lens of main-memory hash joins, breaking down performance across execution phases and developing a performance model that captures both bandwidth benefits and data movement costs. Our analysis demonstrates that moving only a portion of data from CXL memory to DRAM can outperform the conventional strategy of relocating the entire dataset to the interleaved memory tier for subsequent in-place processing, due to reduced data movement and a balanced use of memory bandwidth resources. This work challenges established practices and offers practical guidance for designing CXL-aware query operators [1].

1 INTRODUCTION

Modern database management systems (DBMSs) have suffered from the main memory (DRAM[2]) scaling wall over the past decade. As on-chip logic size continues to shrink, DRAM manufacturing has struggled to keep pace. Consequently, memory resources such as bandwidth-per-core and memory-capacity-per-core have plateaued, resulting in diminishing performance gains for modern DBMS and other memory-intensive data systems [11, 28, 31].

Compute Express Link (CXL) [23] has emerged as a recent advancement aimed at addressing this memory wall. It enables cache coherent memory expansion over the Peripheral Component Interconnect Express (PCIe) interface, supporting rack-scale memory pooling and providing auxiliary bandwidth beyond conventional Double Data Rate (DDR) based memory, albeit at higher access latency [12, 13, 24]. In addition, the PCIe interface offers improved scalability in pin count and lower integration cost, making CXL a more economical and power efficient alternative to traditional DDR technologies [18, 26]. Given these advantages, CXL is increasingly viewed as a practical foundation for building augmented memory systems that can accelerate in-memory DBMS, analytical workloads, and other memory and bandwidth-intensive operations [8, 13, 24].

As CXL sits atop PCIe, accessing CXL memory and DRAM in parallel allows the system to utilize bandwidth from both DDR and PCIe interfaces, resulting in a total bandwidth that exceeds what either can provide individually. Based on this observation, many have advocated forming a new memory tier by interleaving DRAM and CXL memory at a system-specific ratio [8, 24, 26], relocating the entire workload to the interleaved tier, and executing it there for improved performance (see Section 2.1 for details). This strategy has reported significant gains across multiple bandwidth-intensive workloads [8, 13, 17, 25, 32].

At first glance, this widely adopted practice appears to be a reasonable approach for bandwidth-intensive workloads. However, we argue that it may become suboptimal, as it overlooks the cost of moving data into the interleaved memory tier. Many such workloads span hundreds of gigabytes or more and are often stored in slower but more economical storage layers to preserve precious DRAM capacity. With CXL memory introduced, these large workloads are typically placed in CXL memory, which is a common deployment scenario for large-scale DBMSs and deep learning serving systems built on tiered or rack-scale memory architectures [1, 9, 14–16]. In such settings, relocating the entire workload into a newly formed interleaved tier may introduce substantial data movement overhead. This overhead can negate the benefits of increased bandwidth, and in some cases, result in worse end-to-end performance than running the workload entirely in CXL memory [3].

This paper validates this argument through the lens of main-memory hash joins, which are widely regarded as bandwidth-intensive and serve as fundamental building blocks in OLAP and other in-memory DBMS workloads. Specifically, we study two state-of-the-art main-memory hash join algorithms, partitioned hash join (PHJ) [2, 7, 21] and non-partitioned hash join (NPHJ) [3, 4], by decomposing them into execution phases and evaluating end-to-end performance, including the cost of loading data from CXL memory to a faster memory tier (either DRAM or the aforementioned DRAM-CXL interleaved memory tier). We then formulate a simple yet generic performance model to quantify the benefits of increased bandwidth and the penalties of data movement. Our analysis reveals that, contrary to conventional wisdom, judiciously moving a portion of data to DRAM can properly balance data movement costs and bandwidth benefits. This approach delivers better end-to-end performance than either fully relocating data to the interleaved memory tier or executing directly from CXL memory, even when the DRAM-to-CXL data ratio deviates from the system-optimal interleaving configuration for peak bandwidth. Moreover, since our performance model is developed independently of any particular

[1]The source code is available at https://github.com/fukien/hashjoins-meet-cxl .
[2]Since DRAM is the primary medium used for main memory, we use the terms "main memory" and "DRAM" interchangeably throughout this paper.

[3]We assume that the interleaved memory tier can accommodate the entire workload. Otherwise, performance will further degrade due to additional data movement.

 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, Chaminade, USA

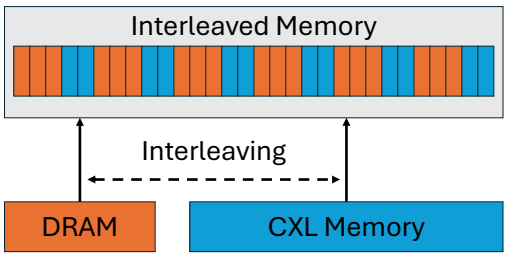

Figure 1: Illustration of DRAM and CXL memory interleaving with a $3:2$ page placement ratio.

operator, we envision its application not only to broader bandwidth-intensive workloads but also as a blueprint for future CXL-aware query execution algorithms.

We summarize our contributions as follows:

(1) We revisit main-memory hash joins in CXL systems from an end-to-end perspective. Our analysis reveals that the prevailing practice of relocating the entire workload to the interleaved memory tier for execution introduces additional data movement overhead, which can in some cases degrade performance rather than improve it.
(2) We develop a simple yet generic performance model and advocate a novel execution strategy for main-memory hash joins in CXL systems. We show that by judiciously moving a portion of data from CXL to DRAM, hash joins can effectively balance the cost of data movement and the benefits of increased bandwidth during join processing, resulting in improved overall performance.

The remainder of the paper is organized as follows. Section 2 discusses background and related work. Section 3 presents the design tradeoffs of main-memory hash joins on CXL platforms and introduces our performance model. Section 4 provides our experimental evaluation. We conclude in Section 5.

2 BACKGROUND AND RELATED WORK

2.1 The CXL Memory Expansion

Compute Express Link (CXL) is a promising technology to break through the memory scaling wall. It introduces a set of cache coherence protocols and device types [23], among which type-3 devices are specifically designed for memory expansion [16, 28]. In addition, the CXL specifications 2.0 and 3.0 support rack-scale memory pooling and sharing, offering increased memory resource per machine without significantly raising economic cost [6, 16, 23, 26, 30].

As an open industry standard [23], CXL memory has been adopted by various vendors [27], resulting in CXL memory implementations with varying performance characteristics, particularly in terms of access latency and bandwidth [16, 24]. Nevertheless, all implementations share common traits: cost-efficient memory scaling, higher access latency, and lower bandwidth compared to DRAM [10, 30].

Although CXL memory alone does not offer higher bandwidth than DRAM, its auxiliary bandwidth from the PCIe interface can supplement the local DRAM bandwidth, yielding an aggregate bandwidth that exceeds what either memory can provide in isolation [8, 24, 26]. To leverage this, DRAM and CXL memory pages should be interleaved to form a unified memory tier (see Figure 1 for an illustration). The optimal interleaving ratio depends on the system's bandwidth characteristics, typically proportional to the ratio of DRAM bandwidth to CXL memory bandwidth [8, 18, 22, 24, 26, 34]. Since DRAM bandwidth usually exceeds that of CXL memory, the optimal configuration includes more DRAM pages than CXL memory pages. Currently, this memory interleaving can only be configured at the operating system level. Once set, the interleaving ratio is fixed and shared across all user space applications [20, 26].

Because of this aggregate system-level bandwidth, prior work [8, 13, 22, 24, 26] recommends processing bandwidth-intensive workloads by first constructing an interleaved memory tier using the aforementioned optimal ratio (i.e., the DRAM-to-CXL bandwidth ratio) and then relocating the workload to this tier for in-place execution. However, as we argue in this paper, such a strategy overlooks the cost of additional data movement and therefore warrants re-evaluation.

2.2 Main-Memory Hash Joins

Main-memory hash joins are central to query processing in modern DBMS and have been extensively studied for decades. Research and practice have converged on two dominant algorithms: partitioned hash join (PHJ) [2, 5, 21] and non-partitioned hash join (NPHJ) [3, 4].

PHJ is motivated by the view that cache miss penalties are the primary bottleneck in the memory era. It follows a partition-then-join paradigm: input tables are first partitioned into cache-sized chunks using radix partitioning [2, 5], after which join operations can be executed entirely within the cache, avoiding cache misses. In PHJ, the partition phase is the dominant contributor to execution time. While various techniques have been proposed to accelerate this phase [21], its cost often remains significant.

NPHJ, by contrast, leverages hardware capabilities such as simultaneous multithreading (SMT) and out-of-order execution (OOE) to hide cache miss latency. It implements the conventional two-phase hash join: the build phase constructs the hash table, and the probe phase queries over it. As in typical analytical workloads, the build side is smaller than the probe side, making the probe phase dominant in NPHJ execution.

Both PHJ and NPHJ are known to be bandwidth-intensive [2–4], and are thus expected to benefit from the aforementioned DRAM and CXL memory interleaving configuration. However, as we demonstrate in the sections that follow, this strategy is not always optimal in end-to-end scenarios. A more judicious placement of data across DRAM and CXL memory can yield better performance, even when the allocation ratio deviates from the system-optimal interleaving configuration for peak bandwidth.

3 HASH JOINS IN CXL SYSTEMS

We examine the design choices for main-memory hash joins in a genuine CXL-equipped system and introduce a performance model that guides the amount of data to be moved from CXL memory to DRAM for optimal join performance. Notably, this model is not limited to hash joins alone; it can be applied more broadly to any bandwidth-intensive applications to support better design tradeoffs in CXL-enabled platforms.

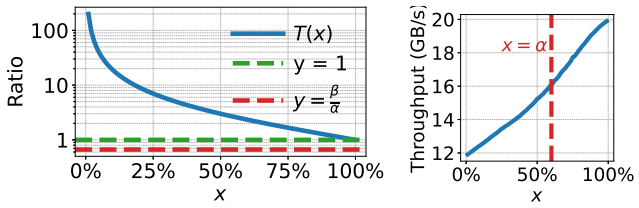

Figure 2: (a) The memory traffic ratio $T(x)$, corresponding to the ratio of CXL to DRAM page access counts, as a function of the DRAM portion x. The green dashed line denotes equal traffic to DRAM and CXL memory, while the red dashed line marks the optimal traffic ratio for the system. (b) Data movement throughput measured across different DRAM portions x, with the red dashed vertical line highlighting the configuration point $x = \alpha$, which corresponds to the optimal DRAM traffic ratio on our system ($\alpha = 60\%$, $\beta = 40\%$).

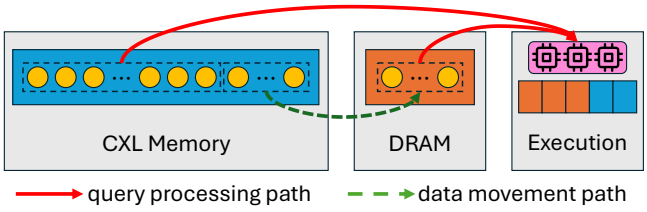

Figure 3: Workflow illustrating partial data movement followed by the subsequent query execution.

Before presenting our analysis, we first examine data movement behavior in CXL systems, a foundational aspect in performance for data-intensive workloads.

3.1 Preliminary Analysis of Data Movement

A CXL system delivers maximum bandwidth when DRAM and CXL memory are accessed at a system-specific ratio. To fully utilize this bandwidth, memory accesses must be carefully distributed across both memory types. The most common and straightforward approach is to construct a new interleaved memory tier and move data from CXL memory into it.

While this setup ultimately ensures that data pages are distributed according to the optimal interleaving ratio, the data movement phase (from CXL memory to the interleaved tier) introduces a mismatch between the actual memory traffic ratio and the system-level optimal ratio. This is because moving pages results in equal memory footprint at both the source (CXL memory) and the destination (the interleaved memory tier). If the destination interleaved tier is already configured with the optimal DRAM-to-CXL ratio, the act of moving data into it will disproportionately increase traffic to CXL memory. This imbalance prevents the system from fully leveraging available DRAM bandwidth during the movement phase, resulting in suboptimal data movement throughput.

To support this claim, we construct a mathematical performance model. Assume we have a CXL-equipped machine, of which the system-wise memory bandwidth maximized at an optimal DRAM-to-CXL page ratio of $\alpha : \beta$. Meanwhile, we configured an interleaved memory tier with an arbitrary DRAM-to-CXL page ratio of $x : y$, and we are now moving pages from CXL memory to this interleaved memory tier. Since the data movement is actually sequentially duplicating memory pages from the source to the destination, the memory traffic to the source memory address and the destination address is equivalent. From the perspective of memory traffic to the memory devices, as the destination is interleaved across DRAM and CXL memory, the total memory traffic therefore leans towards CXL memory. Consequently, the memory traffic to CXL memory becomes $x + 2y$ while the memory traffic to DRAM is just x. As x and y represent percentages, we have $y = 100 - x$ for the data movement phase. Thus, we can model the ratio of CXL memory to DRAM page accesses as a function of x:

$$T(x) = \frac{x + 2y}{x} = \frac{x + 2 \times (100 - x)}{x} = \frac{200}{x} - 1\,. \tag{1}$$

We plot the function in Figure 2(a) and observe that, during data movement, the overall memory traffic ratio of CXL-to-DRAM approaches but never reaches the optimal system-level ratio (i.e., $\frac{\beta}{\alpha}$) that offers the maximum bandwidth, even as the DRAM page ratio in the interleaved memory tier approaches 100%. To validate this behavior, we conduct a microbenchmark and present the result in Figure 2(b), which shows that data movement throughput increases monotonically as the DRAM portion in the destination interleaved memory tier increases. In contrast, moving pages to an interleaved tier at the optimal system-level ratio (i.e., $x = \alpha$ in Figure 2(b)) yields only moderate throughput. Since the near-future CXL memory technology is not expected to surpass DRAM in bandwidth, we conclude that moving data directly from CXL to DRAM achieves higher throughput than moving data to any interleaved memory tier.

3.2 An End-to-End Performance Model

We now present an end-to-end performance model for main-memory hash joins in CXL memory systems. Hash joins are generally considered blocking operators, and both PHJ and NPHJ exhibit this behavior by executing across distinct phases. Despite their differences, all execution phases of PHJ and NPHJ, excluding the PHJ join phase (explained in Section 3.3) , can be abstracted into a common procedure: a data movement phase followed by query execution (see Figure 3). This procedure model is not limited to hash joins and is applicable to other bandwidth-intensive operations that involve data movement.

Suppose we are executing a bandwidth-intensive operation, where the corresponding data pages initially reside in CXL memory to conserve limited DRAM capacity. Since interleaving DRAM and CXL memory at a system-specific optimal ratio yields the highest overall bandwidth, it is desirable to execute the operation within this interleaved memory tier.

However, as shown in Section 3.1, relocating the entire dataset into the optimally interleaved memory tier results in lower throughput than moving the same data directly to DRAM. We therefore propose an alternative approach: instead of moving the entire set of data pages, we relocate only a portion of data to DRAM.

This strategy provides three-fold benefits: (1) Moving only a portion of the data reduces the total number of data pages that must be transferred, thereby lowering the overhead of the data movement phase. (2) The data placed in DRAM contributes to DRAM-side traffic during the subsequent query execution. By adjusting the portion of data moved, we can approximate or even match the system-optimal interleaving ratio, enabling near-maximum bandwidth for execution. (3) This partial movement approach introduces flexibility, allowing us to balance the cost of data movement with the performance gains from high bandwidth query execution. As a result, we can achieve improved end-to-end performance by tuning the movement ratio accordingly.

We now propose a performance model to analyze the above bandwidth-intensive procedure. Suppose a query execution phase achieves a throughput of r tuples per second when executed entirely in CXL memory. The corresponding normalized cost can be expressed as:

$$\text{cost}_1 = \frac{1}{r}. \tag{2}$$

Next, consider a scenario where a fraction x of the data pages is moved from CXL memory to DRAM, and the system supports a data movement throughput of p tuples per second. In addition, assume that the subsequent query execution phase achieves a combined throughput of q tuples per second, when accessing x fraction of the data from DRAM and the remaining $1-x$ from CXL memory, we can derive the corresponding normalized cost as: $\frac{x}{p} + \frac{1}{q}$.

Recall that p denotes the data movement throughput from CXL memory to DRAM, which is a system-specific parameter and is independent of the data movement fraction x. We thus define $p = kr$ to relate it to r, where k is a system-dependent coefficient.

For q, since it directly depends on the fraction x of data pages moving to DRAM, we express q as $q = f(x)r$. Existing studies have demonstrated that the system's aggregate bandwidth increases monotonically as the DRAM portion grows from 0 to the optimal ratio α, and decreases monotonically beyond α [8, 24, 26]. This suggests that choosing $x > \alpha$ is not meaningful, as it incurs higher data movement cost while offering less bandwidth than moving exactly α fraction. Therefore, we restrict the DRAM portion x to the range $[0, \alpha]$.

To model $f(x)$, we use a linear approximation $f(x) = mx + 1$, where m is a system-specific coefficient. Accordingly, the normalized cost $\frac{x}{p} + \frac{1}{q}$ becomes:

$$cost_2 = \frac{x}{p} + \frac{1}{q} = \frac{x}{kr} + \frac{1}{f(x)r} = \frac{x}{kr} + \frac{1}{(mx+1)r}. \tag{3}$$

Although the linear function is a simplification, it aligns with trends observed in bandwidth scaling behavior reported in prior work [8, 13, 16, 20, 24, 26]. While the actual relationship may not be linear, as shown in Section 4, this linear model offers sufficiently accurate performance predictions for practical purposes.

To justify the data movement strategy, $cost_2 < cost_1$ must hold. We define the following function to represent the $\frac{\text{cost}_1}{cost_2}$ ratio:

$$g(x) = \frac{\frac{1}{r}}{\frac{x}{p} + \frac{1}{q}} = \frac{kf(x)}{f(x)x + k} = \frac{kmx + k}{mx^2 + x + k}, \tag{4}$$

where a larger value of $g(x)$ indicates greater benefit from data movement. In order to identify the best value of x, we take the derivative of $g(x)$:

$$g'(x) = \frac{k\left(mk - m^2x^2 - 2mx - 1\right)}{\left(mx^2 + x + k\right)^2}, \tag{5}$$

and derive that the maximum value of $g(x)$ is achieved when $x = \frac{\sqrt{mk}-1}{m}$. Note that this result is valid only when $x \le \alpha$, as our model assumes the valid domain of x is within $[0, \alpha]$. If $x > \alpha$, we cap x at α to avoid excessive data movement and to prevent bandwidth degradation caused by an inferior memory access ratio.

Substituting the optimal x back into Equation 4, we derive the expected performance gain, $\frac{mk}{2\sqrt{mk}-1}$, which is greater than 1 if $mk \ge 1$, indicating a beneficial effect from data movement. If $mk < 1$, the optimal x becomes negative, suggesting that no data movement should occur and the system should proceed directly to in-CXL query execution. Overall, the maximum expected performance gain is expressed as,

$$\phi(m,k) = \begin{cases} \dfrac{mk}{2\sqrt{mk}-1}, & mk \ge 1, \\ 1, & mk < 1. \end{cases} \tag{6}$$

which captures the upper bound of the achievable performance gain from partial data movement under our model.

3.3 Hash Join Using the End-to-End Model

We now apply the end-to-end performance model to main-memory hash joins. For PHJ, the model accelerates the partition phase by judiciously moving a portion of the input tables equal to $\frac{\sqrt{mk}-1}{m}$ to DRAM, thereby allowing the operation to leverage increased aggregate bandwidth. This DRAM portion value is suggested from our performance model, and does not need to follow the optimal interleaving ratio for peak system bandwidth, which may incur unnecessary data movement overhead. Meanwhile, the destination for partitioned output should be the interleaved memory tier configured with the optimal system-level interleaving ratio, for the purpose of having highest memory bandwidth in materialization. For the PHJ join phase, the model is not applicable, as it performs cache-level join by a single direct scan over cache-sized partitions, which already reside in the interleaved memory tier and has no need of additional data movement. Since the partition phase dominates the execution time of PHJ [2, 5, 21], the proposed procedure can yield substantial performance improvements.

For NPHJ, however, our approach can be applied to both the build and probe phases, and we can estimate the performance gain from Equation 6 directly.

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setup

We conduct evaluation on a genuine CXL-equipped platform compliant with the CXL 1.1 specification [23, 27]. The host machine is configured with 32GB of DDR5 DRAM, offering a peak bandwidth of 30.01GB/s. The attached CXL type-3 memory provides 64GB of capacity and achieves 20.95GB/s bandwidth over PCIe 5.0. Following previous studies [8, 13, 20, 24, 26], we configure DRAM and CXL memory interleaving at various ratios. We find that a 3:2 DRAM-to-CXL interleaving ratio yields the maximum achievable aggregate bandwidth in our system, reaching 47.50GB/s.

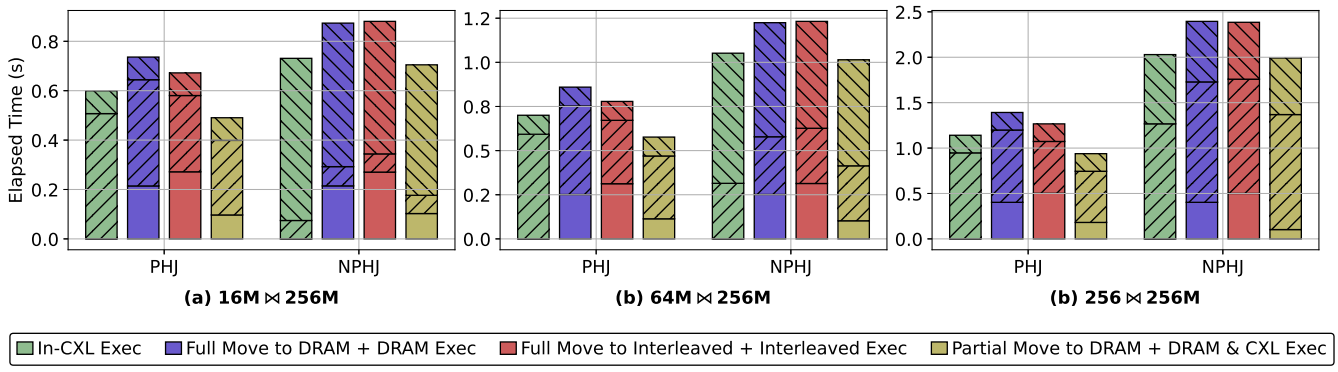

Figure 4: Hash join performance comparison across varying input sizes. The stacked bars show the runtime breakdown, with segments (bottom to top) representing memory movement (if applicable), partition/build, and join/probe phases, distinguished by solid fill, forward slash (//), and backslash (\\) hatching patterns, respectively.

Table 1: Measured parameters and expected throughput gains on our platform.

Algorithm	k	m	$x^* = \frac{\sqrt{mk}-1}{m}$ †	$g(x)\|_{x=x^*}$ ‡
PHJ	2.3534	1.0977	0.5528	1.1703
NPHJ	3.7879	0.3641	0.5104	1.0214

† x^* denotes the optimal fraction of data to be moved from CXL memory to DRAM.
‡ $g(x)|_{x=x^*}$ represents the expected throughput improvement when using the optimal setting $x = x^*$.

We adopt representative implementations of main-memory hash joins and evaluate them on the canonical synthetic equi-join benchmark [2–4, 7, 21]. The benchmark consists of input relations with cardinalities of 16M, 64M, and 256M, with a fixed tuple size of 16 bytes. In our experiments, we fix the probe side cardinality at 256M while varying the build side cardinality (16M, 64M, and 256M). This setup allows us to evaluate performance across varying build-to-probe ratios, thereby assessing the effectiveness and robustness of our approach.

Throughout evaluation, the input data is initially stored in CXL memory. We organize our study by algorithm and evaluate the following four execution strategies: (1) executing the join directly in CXL memory, (2) fully moving the data to DRAM followed by join processing in DRAM, (3) fully moving the data to an interleaved memory tier configured with the system optimal DRAM-to-CXL ratio, followed by join processing in that tier, and (4) partially moving an optimal fraction of the data, as guided by our performance model (Section 3.2), and performing the join by accessing DRAM and CXL memory in parallel. In all strategies, we place intermediate results in the interleaved memory tier configured at the optimal DRAM-to-CXL ratio to enable faster processing [4]. Moreover, in order to apply the performance model, we first profile the system to extract the necessary throughput parameters, from which we derive the values of k and m, calculate the optimal data fraction x^* to be moved to DRAM, and estimate the corresponding expected performance gain, as listed in Table 1.

[4]Intermediate results refer to the PHJ cache-sized partitions and the NPHJ hash table, which require materialization in memory and therefore do not conflict with our performance model.

4.2 Evaluation Results

Figure 4 presents the performance of PHJ and NPHJ under four execution strategies. In general, PHJ outperforms NPHJ by effectively mitigating cache miss penalties, consistent with observations in prior work [2, 3, 21]. Moreover, when benchmarked against competing methodologies, our proposed strategy exhibits the best performance, outperforming alternatives across all investigated algorithms and build-to-probe cardinality ratios. This is because it leverages the performance model to judiciously move an optimal portion of data to DRAM, enabling high bandwidth during subsequent query execution while avoiding superfluous data movement.

Across both join variants, our approach yields 22.20% and 3.59% runtime reduction for PHJ and NPHJ, respectively, compared to the second-best strategy (i.e., in-CXL join processing). These improvements closely match the expected gains reported in Table 1, thereby validating the accuracy of our model. Furthermore, the observed performance improvements exhibit stability across all evaluated workloads, irrespective of the build-to-probe cardinality ratio. This consistency strongly indicates that our proposed model possesses sufficient robustness for a diverse range of bandwidth-intensive relational workloads.

Compared to PHJ, the performance gain for NPHJ is inherently lower, primarily due to the smaller m value in Equation 4. This limited runtime reduction mainly stems from the probe phase, where memory traffic is majorly dominated by random accesses to the hash table. Because the hash table already resides in the interleaved memory tier with the optimal DRAM-to-CXL ratio after the build phase, the choice of the probe-side data source has minimal impact on performance. As a consequence, the benefits of our approach are more constrained.

Excluding our proposed scheme, both PHJ and NPHJ perform better with direct in-CXL processing than relocating data entirely to a faster memory tier for in-place processing, a result that validates the benefits of minimizing data movement. Regarding subsequent execution phases, we observe negligible performance variation

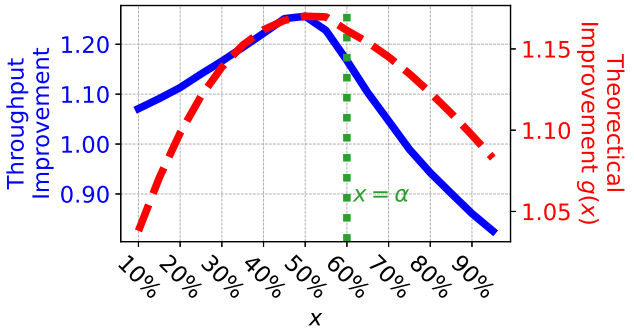

Figure 5: Performance comparison with respect to x, denoting the portion of data moved from CXL memory to DRAM. The blue and red curves represent measured throughput gain and model-estimated gain $g(x)$, respectively. The vertical dotted line (green) marks the upper bound of the x domain defined by the performance model.

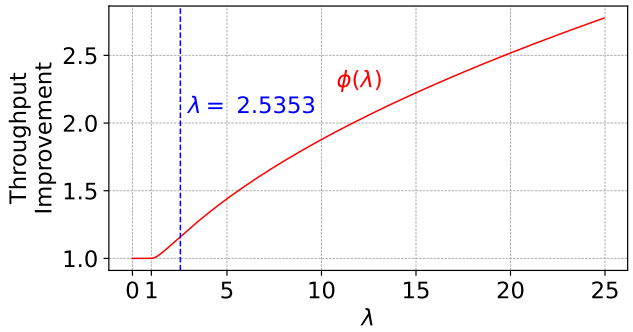

Figure 6: Projected performance improvement from our model across different values of λ. The blue vertical dashed line indicates our system configuration point at $\lambda = 2.5353$ (with $k = 2.3534$, $m = 1.0977$).

across strategies. This uniformity arises because all strategies place intermediate data, whether PHJ partitions or NPHJ hash tables, in the interleaved memory tier, thereby ensuring high bandwidth for the remainder of the query. When comparing full data migration strategies, we find that replicating data to DRAM incurs lower movement costs than relocating it entirely to the interleaved tier, corroborating our analysis in Section 3.1. Additionally, the performance divergence between these strategies underscores the critical importance of accounting for data movement overhead when designing join processing algorithms for CXL systems.

4.3 Sensitivity Study of the Performance Model

We now evaluate the sensitivity of the performance model. Using partitioning as a case study, we vary the x fraction of data moved from CXL memory to DRAM and measure the corresponding throughput improvement. We also plot the model-predicted performance gain $g(x)$ in Figure 5 for comparison.

We observe that the optimal data movement x ratio predicted by our model (0.5528) is close to the empirically determined optimal ratio (0.4972), and the estimated performance gain (17.03%) is reasonably aligned with the observed maximum improvement (24.20%) [5]. The actual optimal x ratio being slightly lower than the model prediction is due to the sublinear bandwidth improvement rate near the system-level optimal interleaving ratio α (detailed in Section 3.2 where $f(x) = mx + 1$), which allows similar bandwidth with less data movement.

For values of x that deviate significantly from the optimal point (0.5528), the predicted gain also diverges from the measured gain; however, the overall trend remains consistent. Importantly, the model suggests an x value smaller than α, validating its practical utility. In summary, the proposed model provides effective guidance for determining the optimal data movement fraction and achieves high accuracy in doing so.

[5]These gains are for the partitioning phase only and are therefore higher than those reported in Table 1, as the proposed approach is not applied to the join phase.

4.4 Future Trends And Beyond

As CXL technology is still in its nascent stage, we project the normalized performance improvement for future CXL systems based on Equation 6. Letting $\lambda = mk$, we plot the expected performance gain as a function of λ in Figure 6. We observe that as λ increases, the predicted gain from our approach grows monotonically. This trend is particularly insightful, as the industry roadmap points toward higher bandwidth for both CXL and PCIe interfaces [10, 23, 28, 34]. Meanwhile, we anticipate that CXL memory bandwidth will remain lower than that of DRAM, implying that coefficients k and m will likely stay above 1, given that the parallel use of DRAM and CXL memory yields higher aggregate bandwidth than relying on CXL memory alone. As a result, the proposed partial data movement strategy will continue to offer benefits in next-generation CXL-equipped platforms.

Additionally, since CXL technology fundamentally represents memory expansion via interconnects, a natural question arises: does our proposed approach extend to similar technologies relying on interconnects? We posit that our methodology applies to any interconnect that delivers additional memory bandwidth independent of local DDR channels, including NVIDIA NVLink and NUMA interconnect technologies [19, 20, 24, 29, 33]. In such architectures, our approach can judiciously determine the optimal portion of data to migrate to local DRAM, effectively balancing data movement costs against bandwidth gains. Moreover, given the constraints of the DRAM scaling wall, this supplemental bandwidth will play an increasingly pronounced role in holistic system performance. Consequently, our approach serves as a vital blueprint for optimizing applications with high bandwidth demands across emerging heterogeneous memory hierarchies.

5 CONCLUSION

In this paper, we revisit main-memory hash joins in CXL-enabled systems. We propose a simple yet generic performance model that balances the cost of data movement against the benefit of increased system bandwidth. Guided by this model, we advocate a new execution strategy: judiciously moving a portion of data from CXL memory to DRAM prior to join processing, which enables higher

bandwidth by accessing both memory types in parallel. We validate our approach through experimental evaluation, and the observed performance improvements confirm the accuracy and effectiveness of the proposed model.

Looking ahead, we plan to extend this framework to other bandwidth-intensive query operations and explore its applicability to multi-join query processing in CXL-equipped systems. Furthermore, we aim to validate its generalization to diverse bandwidth-critical workloads atop alternative memory expansion technologies.

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).

REFERENCES

[1] 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 (2024), 3827–3840.

[2] 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.

[3] 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.

[4] 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.

[5] Peter A. Boncz, Stefan Manegold, and Martin L. Kersten. 1999. Database Architecture Optimized for the New Bottleneck: Memory Access. In *VLDB*. Morgan Kaufmann, 54–65.

[6] Yunyan Guo and Guoliang Li. 2024. A CXL- Powered Database System: Opportunities and Challenges. In *ICDE*. IEEE, 5593–5604.

[7] 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.

[8] 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 *VLDB Workshops*. VLDB.org.

[9] Yibo Huang, Newton Ni, Vijay Chidambaram, Emmett Witchel, and Dixin Tang. 2025. Pasha: An efficient, scalable database architecture for cxl pods. In *CIDR*. www.cidrdb.org.

[10] Astera Labs Inc. 2025. Leo CXL® Smart Memory Controllers. https://www.asteralabs.com/products/leo-cxl-smart-memory-controllers/

[11] 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.

[12] 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.

[13] Georgiy Lebedev, Hamish Nicholson, Musa Ünal, Sanidhya Kashyap, and Anastasia Ailamaki. 2025. Demystifying CXL Memory Bandwidth Expansion for Analytical Workloads. In *VLDB Workshops*. VLDB.org.

[14] 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.

[15] Sangjin Lee, Alberto Lerner, Philippe Bonnet, and Philippe Cudré-Mauroux. 2024. Database Kernels: Seamless Integration of Database Systems and Fast Storage via CXL. In *CIDR*. www.cidrdb.org.

[16] Alberto Lerner and Gustavo Alonso. 2024. CXL and the Return of Scale-Up Database Engines. *Proc. VLDB Endow.* 17, 10 (2024), 2568–2575.

[17] 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.

[18] 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 *ASPLOS (2)*. ACM, 1203–1217.

[19] MemVerge. accessed in 2024. Introducing Weighted Interleaving in Linux for Enhanced Memory Bandwidth Management. https://memverge.ai/introducing-weighted-interleaving-in-linux-for-enhanced-memory-bandwidth-management/

[20] Gregory Price. accessed in 2024. [PATCH v2 3/3] mm/mempolicy: introduce MPOL_WEIGHTED_INTERLEAVE for weighted interleaving. https://lore.kernel.org/lkml/20240119175730.15484-4-gregory.price@memverge.com/

[21] Stefan Schuh, Xiao Chen, and Jens Dittrich. 2016. An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory. In *SIGMOD Conference*. ACM, 1961–1976.

[22] 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. *CoRR* abs/2412.12491 (2024).

[23] Debendra Das Sharma, Robert Blankenship, and Daniel S. Berger. 2024. An Introduction to the Compute Express Link (CXL) Interconnect. *ACM Comput. Surv.* 56, 11 (2024), 290:1–290:37.

[24] 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.

[25] Yupeng Tang, Runxiang Cheng, Ping Zhou, Tongping Liu, Fei Liu, Wei Tang, Kyoungryun Bae, Jianjun Chen, Wu Xiang, and Rui Shi. 2024. Exploring cxl-based kv cache storage for llm serving. In *Workshop on ML for Systems at NeurIPS*.

[26] Yupeng Tang, Ping Zhou, Wenhui Zhang, Henry Hu, Qirui Yang, Hao Xiang, Tongping Liu, Jiaxin Shan, Ruoyun Huang, Cheng Zhao, Cheng Chen, Hui Zhang, Fei Liu, Shuai Zhang, Xiaoning Ding, and Jianjun Chen. 2024. Exploring Performance and Cost Optimization with ASIC-Based CXL Memory. In *EuroSys*. ACM, 818–833.

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

[28] Micron Technology. 2023. Memory Scaling is Dead, Long Live Memory Scaling. https://sg.micron.com/content/dam/micron/global/public/products/white-paper/cxl-impact-dram-bit-growth-white-paper.pdf

[29] Markus Velten, Robert Schöne, Thomas Ilsche, and Daniel Hackenberg. 2022. Memory Performance of AMD EPYC Rome and Intel Cascade Lake SP Server Processors. In *ICPE*. ACM, 165–175.

[30] 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.

[31] Zhao Wang, Yiqi Chen, Cong Li, Yijin Guan, Dimin Niu, Tianchan Guan, Zhaoyang Du, Xingda Wei, and Guangyu Sun. 2025. CTXNL: A Software-Hardware Co-designed Solution for Efficient CXL-Based Transaction Processing. In *ASPLOS (2)*. ACM, 192–209.

[32] Marcel Weisgut, Daniel Ritter, Pınar Tözün, Lawrence Benson, and Tilmann Rabl. 2025. CXL Memory Performance for In-Memory Data Processing. *Proc. VLDB Endow.* 18, 9 (2025), 3119–3133.

[33] Felix Werner, Marcel Weisgut, and Tilmann Rabl. 2025. Towards Memory Disaggregation via NVLink C2C: Benchmarking CPU-Requested GPU Memory Access. In *HCDS*. ACM, 8–14.

[34] Yuhong Zhong, Daniel S. Berger, Carl A. 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 *OSDI*. USENIX Association, 37–56.