An Examination of CXL Memory Use Cases for In-Memory Database Management Systems using SAP HANA

Minseon Ahn
minseon.ahn@sap.com
SAP Labs Korea

Thomas Willhalm
thomas.willhalm@intel.com
Intel Deutschland GmbH

Norman May
norman.may@sap.com
SAP SE

Donghun Lee
dong.hun.lee@sap.com
SAP Labs Korea

Suprasad Mutalik Desai
suprasad.desai@intel.com
Intel Technology India Pvt. Ltd.

Daniel Booss
daniel.booss@sap.com
SAP SE

Jungmin Kim
jimmy.kim@sap.com
SAP Labs Korea

Navneet Singh
navneet.singh@intel.com
Intel Technology India Pvt. Ltd.

Daniel Ritter
daniel.ritter@sap.com
SAP SE

Oliver Rebholz
oliver.rebholz@sap.com
SAP SE

ABSTRACT

CXL-based disaggregated memory systems offer options to expand the memory beyond the limits of a single server via cache-coherent memory expansion cards or memory pools. Especially, In-Memory Database Management Systems (IMDBMSs) can benefit from alleviating two critical constraints: (1) limited memory capacity in a server and (2) long restart time during failover to reload data to memory. However, the usage and effectiveness of CXL memory in enterprise-scale IMDBMSs has yet to be validated.

In this work—for the first time—we investigate dynamic memory expansion employing commercial CXL memory devices for IMDBMSs. Our detailed performance analysis reveals that the performance impact of higher latency and lower memory bandwidth impact depends on the memory access patterns of data structures (cf. (1)). Additionally, we present the feasibility of CXL shared memory between servers to improve restart times during failover (cf. (2)).

Our evaluation shows the effectiveness of CXL memory integrated into the SAP HANA Cloud IMDBMS. OLTP workloads have a negligible performance degradation while OLAP workloads have a wide range of performance degradation. CXL shared memory shows a 40% reduction of the restart time for TPC-H SF10 and 84% potential reduction for TPC-H SF100.

PVLDB Reference Format:
Minseon Ahn, Thomas Willhalm, Norman May, Donghun Lee, Suprasad Mutalik Desai, Daniel Booss, Jungmin Kim, Navneet Singh, Daniel Ritter, and Oliver Rebholz. An Examination of CXL Memory Use Cases for In-Memory Database Management Systems using SAP HANA. PVLDB, 17(12): 3827 - 3840, 2024.
doi:10.14778/3685800.3685809

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685809

1 INTRODUCTION

As memory devices become one of the most expensive components in cloud data centers and stranded memory—parts of its DRAM being unused—becomes the major reason of inefficiencies in the cloud [25], numerous research are dedicated to investigating disaggregated memory systems [1, 16, 51, 53]. With the emergence of CXL technology, disaggregated memory systems have the great potential to relieve the constraints of limited memory space in a single server, enabling efficient memory resource sharing within a rack, offering advantages such as enhanced memory utilization, reduced fragmentation, and lower operational costs [52]. Furthermore, it holds particular significance for In-Memory Database Management Systems (IMDBMSs) by alleviating constraints associated with limited memory space in a single server, extending beyond the capacity of a single board to encompass an entire server rack [24]. While there is extensive research on this area like [1, 16, 51, 53], to the best of our knowledge, none of the previous work investigated the actual performance effects employing commercial-grade CXL memory on enterprise-level IMDBMSs.

In our previous work [22], we addressed a few elastic use cases of CXL memory in SAP HANA. Using the two features i) moving main table data to CXL memory and ii) allocating operational heap memory in CXL memory, we investigated the performance impact of CXL memory. We reported no performance degradation with Online Transaction Processing (OLTP) workloads even with longer latency of our CXL 1.1 compatible FPGA prototypes. However, a wide range of performance degradation was noted across Online Analytical Processing (OLAP) workloads, dependent on memory access patterns. The associated experiments were constrained by limitations inherent in our early FPGA-based prototypes.

In this study, we explore two operational use cases of CXL memory to mitigate two fundamental architectural constraints within IMDBMSs by building upon our previous research, i.e., dynamic memory expansion and failover with fast restart. The inherent limitation of memory capacity due to a restricted number of slots for memory DIMMs in a server poses a critical resource constraint in IMDBMSs. To alleviate this restriction, we investigate dynamic memory expansion using CXL memory. Additionally, the time required to initialize an IMDBMS during server restarts or takeovers in emergency, attributed to the data reloading of necessary tables into memory, represents a significant operational challenge. To mitigate this issue, we propose a solution for failover with fast restart by leveraging CXL shared memory among multiple servers.

Dynamic memory expansion While emerged CXL technology easily supports dynamic memory expansion, we need further investigation on which data should move to CXL memory in IMDBMSs. We select three dominant data structures from our real-world enterprise IMDBMS, SAP HANA: i) table data, ii) operational data, iii) temporary tables, and conduct intensive performance experiments employing commercial CXL memory devices. Our experiments with OLTP and OLAP workloads using a new SAP HANA prototype, reveal diverse performance effects when placing the different data structures in CXL memory. Our analysis of profiles reveals that their performance depends on the memory access patterns, such as sequential or random accesses.

Failover with fast restart CXL 3.0 [9] introduces memory sharing among multiple hosts. This advancement allows multiple hosts to collaboratively access and share data within a CXL shared memory. We implement CXL shared memory supporting CXL 3.0 and build a two-host high availability system to test failover with fast restart. We show the feasibility of failover with fast restart by reusing the table data in CXL shared memory, significantly reducing the server restart time and improving overall system availability.

Contributions and key insights In summary, this work makes the following main contributions:

- Specification of CXL memory use cases, i.e., dynamic memory expansion and failover with fast restart leveraging CXL to alleviate two fundamental constraints of IMDBMSs.
- Experimental investigation of performance effects, when employing CXL 2.0 compatible commercial CXL memory devices in a real enterprise IMDBMS using various workloads, such as OLTP and OLAP, including a real-world scenario.
- Insights into a wide range of performance degradation on dynamic memory expansion according to the data structures and their memory access patterns.
- Implementation of CXL shared memory and demonstration of its feasibility to enable the failover with fast restart between two servers, resolving slow restart to load the data to memory on failover and improving overall system availability.

Our extensive examination of the first commercial CXL memory devices that we integrate in the enterprise-scale SAP HANA Cloud IMDBMS focuses on various workloads for the memory expansion. We report no significant performance degradation for TPC-C [48] with 100 warehouses, but a wide range of degradation for TPC-DS [49] SF100 due to limited bandwidth and long latency—depending on the type of placed data—for current CXL devices. Temporary table allocation in CXL memory was tested through PaPM [41] and ProcBench [17], showing degradation up to 13% for PaPM and nearly none for ProcBench. The failover with fast restart with CXL shared memory, saves 85% preloading time with total reductions of the restart time by 40% for TPC-H [50] scale factors 10. It is estimated to save 95% preloading time and 84% restart time for TPC-H scale factor 100.

In addition, we find that a dynamic memory expansion via CXL helps reducing the amount of host memory in each server / virtual machine, and thus improving the overall memory utilization, allowing for the usage of smaller, cheaper DIMMs, and reducing the need for overprovisioning.

Outline The remainder of this paper is organized as follows: Section 2 introduces CXL memory and SAP HANA. Section 3 discusses use cases of CXL memory while Section 4 describes the details of the commercial CXL memory devices and our FPGA-based CXL shared memory. We conduct the performance evaluation and share its analysis in Section 5 and discuss our insight in Section 6. We give an overview of related work in Section 7. Finally, we conclude our work in Section 8.

2 BACKGROUND

In this section, we clarify terminology and introduce relevant concepts of SAP HANA, an IMDBMS, which we use in our experiments.

2.1 Disaggregated memory vs. CXL memory

Disaggregated memory is a new architecture expanding the memory hierarchy, for example, to a remote memory provided by separate memory blades. It offers several advantages, such as higher memory utilization, reduced memory fragmentation, lower operational costs and lower total cost ownership utilizing cheaper memory devices [6, 24, 26, 52]. Disaggregated memory is being incarnated as a high speed cache-coherent interconnect is available, such as CXL [8].

The Compute Express Link (CXL), with its evolution from version 1.0 to 3.1, plays an important role in incarnating disaggregated memory systems and augmenting the capabilities of IMDBMS [22]. CXL 1.0 and 1.1 enable memory expansion in a single server [10, 32] alleviating a constraint of the limited memory capacity within a single server. CXL 2.0 allocates memory to hosts on an "as-needed" basis, optimizing utilization and enabling the creation of an independent disaggregated memory pool. Each server is assigned a portion of memory space in the pool. When IMDBMSs allocate data in this memory pool, the data remains accessible even after an IMDBMS instance restarts. CXL 3.0 and 3.1 [9] extend these capabilities, introducing more flexible memory sharing and access modes within a group of CXL devices. This expansion allows complex connection topologies and fabrics, facilitating multiple instances of IMDBMS to share data in the disaggregated memory pool.

Previous work used different notations to refer to CXL-attached memory, e.g., remote memory, far memory, or disaggregated memory [24]. In this work, we adopt the term '*CXL memory*' as we deal with the expanded memory via CXL technology in addition to the '*native memory*', where the size is limited by the number of DIMM slots in a server. CXL memory includes the expanded memory by directly attached CXL memory expanders [32] in a server, called as '*direct attached CXL memory*', the allocated memory into a server within a disaggregated memory pool system, called as '*pooled CXL memory*', and finally the memory regions which are allocated in a disaggregated memory pool and can be shared among multiple servers, called as '*shared CXL memory*' [7]. Accesses to these additional memory spaces occur via the CXL layer, potentially incurring additional latency through PCIe connections and, further, through CXL switches. Furthermore, the limited bandwidth and long latency constrained by the specifications of each system may reduce the performance compared to native memory. Investigating this performance impact on IMDBMS is one of the main topics of this study.

2.2 SAP HANA In-memory database platform

As a full-fledged enterprise-class in-memory database platform, SAP HANA [12, 13] has revolutionized the management of both transactional and analytical workloads. SAP HANA, a leading Hybrid Transaction/Analytical Processing (HTAP) system [38, 42] uniquely supports both OLTP and OLAP workloads within a single system offering low latency and high throughput. This unified approach not only simplifies the overall system design but also significantly reduces the total cost of ownership (TCO) [2, 34, 35, 37].

SAP HANA uses a compressed, columnar storage layout for both fast read accesses and a low memory footprint. The columnar data is stored in the read-optimized 'main storage' and maintains a separate 'delta storage' for optimized writes. The main storage contains table data and retrievals of these table data consume higher memory bandwidth, especially for OLAP workloads [21]. Fortunately, accesses to the table data are mostly sequential [19], which enables efficient prefetching hiding the access latency. The delta storage is write-optimized columnar storage to save newly inserted or modified data. It is periodically merged with the main storage [12, 31]. In addition, a designated portion of memory is allocated for intermediate data during query processing or other database operations. This allocated space is referred to as '*operational memory*'. Within this category, a temporary table represents a specialized form of operational memory, storing intermediate results in a columnar structure while executing programs or procedures. In real-world scenarios, temporary tables are often linked to out-of-memory situations resulting from large intermediate results. Consequently, we distinguish temporary tables from other operational memory areas and conduct an in-depth investigation into their performance behavior when utilizing CXL memory. Section 3 dives deeper into this topic detailing the issues that may arise when using disaggregated memory. It also summarizes previous efforts to address these challenges leveraging CXL technology.

3 CXL MEMORY USE CASES

IMDBMSs are designed to minimize latency overhead caused by memory accesses and stalled CPU cores. To this end, they keep the working set of data permanently in memory and use NUMA-aware data organization and task scheduling as well as vectorized processing on columnar tables to fully utilize the capabilities of modern hardware. Relying on data being loaded into the main memory makes the restricted memory size a critical resource constraint due to a limited number of slots for memory devices (DIMMs). Additionally, IMDBMSs encounter another constraint during scenarios like DBMS restarts or server takeovers in emergencies, requiring long server restart times to reload data to memory. Upcoming disaggregated memory systems enabled by CXL are poised to alleviate these constraints. This section addresses the existing efforts to surmount these limitations and explores how CXL can enhance these two fundamental constraints.

3.1 Dynamic memory expansion to CXL memory

One of the main constraints of IMDBMSs revolves around the limited memory space in a single server, leading to several potential issues. For example, when sizing their systems for some DBMS workloads, customers risk to either overprovision the memory capacity of their systems when they want to avoid out-of-memory situations, or choose a tight memory capacity resulting in the need to add memory resources as their datasets grow or workload gets more demanding. Dynamic memory expansion will resolve both the high TCO issue by overprovisioning and the challenge to increase the memory space as the data or workload grows.

One approach to extend the memory space of a single server would be to explore utilizing memory in other servers, like Remote Direct Memory Access (RDMA) [15]. However, implementing RDMA necessitates modifying the application source code and introduces extra communication overhead via the network system. In contrast, CXL enables dynamic memory expansion, offering superior latency and throughput compared to RDMA [15]. We have three options to make CXL memory exposed to applications. First, CXL memory is recognized as a memory only NUMA node by default when a CXL memory device is attached. To directly allocate within the CXL memory, applications use the mbind() system call [47] with the NUMA node number of the CXL memory device. Otherwise, this memory only NUMA node will not be used until the local NUMA node is fully used. Second, the additional memory space of the CXL device could be uniformly integrated with the native memory via heterogeneous interleaving [4]. New memory allocation will be done uniformly without any consideration of the native or CXL memory. Third, CXL memory can be exposed as a DAX [14] device by setting its address range within the CXL memory devices.

From our prior research [22], we learned that limited bandwidth and long latency of CXL memory devices can affect the overall data access performance according to their access patterns. Therefore, we need to decide the memory placement per data structure within our IMDBMS and comprehend the performance implications.

As mentioned in Section 2, the whole memory space used in SAP HANA is divided into the three parts – main storage, delta storage, and operational memory. The main storage in SAP HANA is the data storage to store compressed, columnar tables. It is optimized for fast read accesses and a small memory footprint. Our previous work showed endurable performance impact when moving the main storage to CXL memory due to its sequential access patterns because prefetching hides the long latency of CXL memory. We revisit the performance evaluation using the commercial CXL memory against both OLTP and OLAP workloads. The delta storage maintains a separate storage for optimized writes [12, 44] to store the updates of columnar tables, of which size is usually much smaller than the main storage. As it is quite latency sensitive, it is highly recommended to keep it in the local host memory, not in the CXL memory. Thus, it is not included in this investigation.

Operational memory refers to the allocated heap memory reserved to store operational data and intermediate results during query processing. Our prior research indicated a significant performance impact when allocating operational heap memory to CXL memory, particularly due to a higher portion of latency-sensitive random accesses. We reassess and validate these findings in 5, incorporating the use of commercial CXL memory devices. Operational memory encompasses all temporarily data used by the HANA Execution Engine (HEX) [43], excluding the main storage and the delta storage. Here we distinguish temporary tables from operational memory objects. Temporary tables store the intermediate results in a columnar table format during the execution of a procedure. The interim content of these tables is subsequently transferred to the next step within the procedure. The substantial memory space consumed by these tables can lead to operational challenges like out-of-memory situations.

Many real-world systems have reported memory shortages caused by large temporary table usages. For instance, planning processes within SAP Analytics Cloud (SAC) applications generate numerous one-time temporary tables during multiple rounds of its simulation against their data cubes, often adjusting simulation parameters. The financial services performance management system also reports analogous issues. Our investigation utilizes a streamlined version of the financial performance management system. This system focuses on expense evaluation and optimal distribution of the expenses to individual cost centers. We evaluate its performance impact when allocating the temporary tables into CXL memory.

Enabling dynamic memory expansion to CXL memory holds great potential for advancing meaningful use cases within large-scale IMDBMSs. For example, during application system upgrades, it's often necessary to modify the metadata of the base tables. One effective strategy involves temporarily loading the main tables into CXL memory, creating a bridge system that mimics the original setup. This bridge system contains all the required data from the production system, allowing users to seamlessly continue their work. This approach enables near-zero downtime upgrades, ensuring a smooth transition during the application system improvement process. In addition, dynamic memory expansion will empower a more flexible system configuration, ensuring an agile and efficient allocation of resources, adapting to changing computational needs and optimizing the overall system utilization in the Cloud.

3.2 Failover with fast restart using CXL memory

The initialization of an IMDBMS during restarts or takeovers takes a long time to reload the necessary tables into the memory. Addressing the issue is crucial to enhance system efficiency and meet the requirement of service level agreement (SLA) on system availability. In addressing the challenge of slow restarts or takeovers, several approaches have been investigated to mitigate the issue.

Persistent Memory. Utilizing persistent memory storage allows IMDBMSs to retain data in a non-volatile state. Its most important characteristics are that it's byte addressable like DRAM. If persistent memory storage is enabled in SAP HANA [3], data is still written to the data volumes in the persistent storage. After a system restart, the main data fragments saved in persistent storage are still available in memory. Thus, it can reuse the data instead of reloading [39].

CXL memory will be provided as a form of memory pool with CXL 3.0 specification in the near future. If we assume the memory pooling system has its own independent power domain, its behavior would be nearly the same as persistent memory. The persistent memory feature in SAP HANA includes cacheline writebacks and store fences during writes to provide consistency and persistence. This feature is also essential in the CXL memory pool. Thus, we use this persistence memory feature. We expect that this implementation would be independent from the CXL memory medium whether it is volatile DRAM or non-volatile PMEM. Thus, this approach can easily exploit CXL memory.

Fast restart using tmpfs. SAP HANA provides a fast restart option [40], which makes it possible to reuse main data fragments after an SAP HANA service restart without the need to reload the main data from the persistent storage or disk. It leverages the persistent memory implementation in SAP HANA with the key performance benefit of greatly accelerated start-up time so that it quickly reuses the data and minimizes system downtime. In contrast to persistent memory which works with a DAX-enabled file system [14], the fast restart option stores main data fragments in tmpfs with the content in DRAM. It can grow and shrink dynamically and lives completely in the page cache keeping all files in virtual memory. However, the data in tmpfs is lost when the operating system needs any update/service patch or the server reboots.

Incremental Loading. Rather than loading all tables at once, adopting an incremental loading approach involves loading critical tables progressively. This allows the system to become operational faster while still loading less critical data in the background. SAP HANA also provides a preload option for entire column store tables or individual columns, which allows you to mark important data containers for in-memory preload after startup. As expected, the first query execution which accesses the tables not loaded into memory yet can be delayed until all the required tables are loaded into memory.

CXL 3.0. The introduction of peer-to-peer direct memory accesses in CXL 3.0, will enhance memory pooling capabilities, enabling multiple hosts to collaboratively share a memory space on a CXL 3.0 device. This allows a specific portion of memory to be attached to a particular server, and the ownership of the memory region within the memory pool can be dynamically modified among multiple servers. In the context of an IMDBMS, if we assume the memory pooling system has its own independent power domain, it would behave similarly to persistent memory storage. This means data remains intact during power cycles or system restarts of the DBMS servers, regardless of the storage medium, such as persistent memory, fast DDR5 DRAMs, or slower DDR4 DRAMs, used in the memory pool.

This work presents our experiments demonstrating that CXL 3.0 can ensure fast restarts during failover in an IMDBMS by effectively utilizing CXL shared memory between two servers. We leverage the SAP HANA persistent memory implementation and the columnar tables in the main storage are moved to CXL memory. When a failover is invoked, the standby server attaches the CXL shared memory during its restart and IMDBMS reads the tables without reloading the data into its local memory.

Beyond restart or failover, numerous similar use cases can emerge in cloud environments. For example, in scenarios like distributed query processing across multiple compute nodes, instances often require sharing intermediate results among nodes. A CXL-based memory pool proves advantageous in such situations, offering lower latency and improved bandwidth compared to network communication. This enhances overall efficiency in handling distributed processing tasks, such as result buffers and file caches, in the cloud.

4 IMPLEMENTATION OF USE CASES USING CXL MEMORY

This section describes the details of the CXL memory devices employed in SAP HANA. For the investigation of performance impact on dynamic memory expansion, CXL memory devices are directly attached to an IMDBMS server. For failover with fast restart, we develop our own CXL 3.0 featured prototype in response to the unavailability of commercially accessible memory devices supporting CXL 3.0.

4.1 Dynamic memory expansion via CXL

This section describes how to employ dynamic memory expansion in SAP HANA and how to modify SAP HANA to access the CXL memory. During our experiments, we employ engineering samples of Samsung CMM-D (CXL Memory Module-DRAM), cutting-edge commercial CXL memory devices supporting CXL 2.0 [32]. Each CXL memory device consists of a 256GB DDR5 DIMM supporting 5200 MT/s, an internal memory controller, an ASIC logic converting CXL protocols to DDR5, and a PCIe Gen5x8 interface. The overall access latency is about 2.3x of the local NUMA latency, while the FPGA implementation in our previous work [22] has about 4x latency. These devices are connected as CXL type 3 devices in the test machine. Each memory device is exposed as a memory-only NUMA node by default in the Linux Kernel 6.4.0.

To move the main storage to CXL memory, we leverage the persistent memory feature in SAP HANA [3]. After creating an `fsdax` [36] in a memory-only NUMA node and a working directory within the `fsdax`, this working directory is configured as the path to the main storage in SAP HANA. Once we change the memory storage preference, the main storage will be created in this path.

To allocate HEX heap memory and temporary tables in CXL memory, we modify the existing memory allocator in our SAP HANA prototype. When enabling it, we set the memory only NUMA node number to specify where to allocate the memory, instead of the local NUMA node number. Once a new query processing starts, SAP HANA uses the mbind() system call [47] with the NUMA node number to initiate the allocation and trigger a page fault.

4.2 FPGA-based emulation of CXL shared memory

This section introduces our implementation of our FPGA-based CXL shared memory prototype, which is used for our experiments on failover with fast restart in Section 5.3.

The CXL 3.0 specification released on August 2022, has additional features to support memory sharing of the same memory region among multiple hosts and enhanced coherency within the connected hosts. Memory sharing enables multiple servers to connect the same CXL memory region. We leverage the implementation of the persistent memory feature in SAP HANA to move the table data

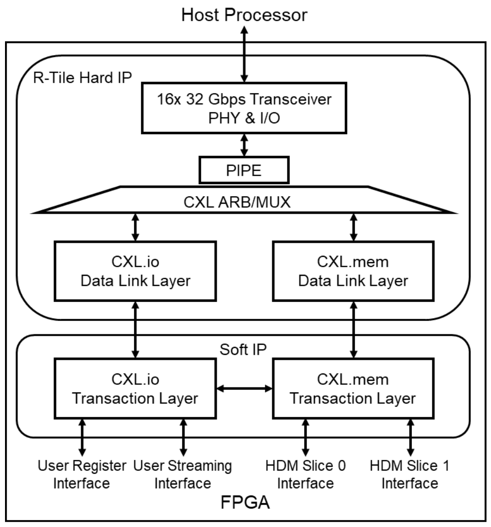


Figure 1: CXL shared memory architecture [22]

into CXL shared memory. When the instance restarts, it reads the table data right away in the CXL shared memory without reloading the data into memory. To enable failover with fast restart, all involved servers need to connect the same CXL memory region in the CXL shared memory. However, failover among multiple servers requires a more sophisticated ownership change on the specific memory area in CXL shared memory. An IMDBMS instance in a primary server has the ownership of the CXL memory to read and write the data in the CXL memory. When the primary server fails and cannot restart by itself, another instance in a standby server needs to take over the role of the instance in the primary server. After changing the ownership to the standby server, the standby server can take over the role and access the data in the CXL shared memory.

The CXL memory pool is realized using an Altera® Agilex™ AGI027 FPGA development board which supports PCIe Gen5 x16 CXL spec 1.1 connectivity. Figure 1 shows the overall diagram of the implementation. The R-Tile Altera® FPGA CXL IP implements the CXL data link layer management functions. Each R-tile is connected to a host with a PCIe Gen5x16 interface and supports up to a 64GB/s theoretical bandwidth. The Soft IP manages transaction layer functions. To support CXL type 3, it manages two transaction layers, CXL.mem and CXL.io. The CXL.mem transaction layer is linked to host-managed device memory (HDM) subsystems while the CXL.io transaction layer is connected to register interfaces. Our memory prototype implemented on the FPGA acts as a software development vehicle (SDV) for the emulating selected functionality of CXL 3.0. The Altera® development board has two interfaces to connect the memory to two servers, (1) the original PCIe edge finger interface supporting 16 PCIe/CXL lanes and (2) the additional interface using two MCIO cables. It allows both servers to read CXL memory at the same time. Our failover test scenario does not have simultaneous accesses from both servers as the main storage is read mostly. We use software-managed coherency in this experiment.

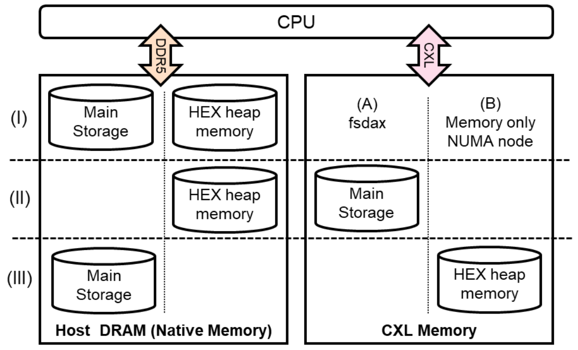


Figure 2: CXL memory configurations for main storage (Main FAR) and HEX heap memory (HEX FAR)

5 PERFORMANCE EVALUATION

In this section, we evaluate the performance impact of the CXL use cases introduced in Section 3. First, we examine the dynamic memory expansion on moving the main storage to CXL memory and allocating HEX heap memory in CXL memory. Analytical and transactional workloads are assessed using the TPC-DS and TPC-C benchmarks, respectively. Subsequently, we evaluate the allocation of temporary tables in CXL memory using procedural workloads, such as PaPM [41] and ProcBench [17]. Lastly, we explore failover with fast restart using our CXL implementation.

5.1 Memory expansion for main storage and operational data

5.1.1 System configuration. We set up the system with Intel 5th generation Xeon® processor code-named *Emerald Rapids*. The system is equipped with one processor with 56 physical cores using Hyper-Threading. The processor has one 128 GB DDR5 4800 MT/s DIMM per channel, totally 1024 GB in 8 channels.

To study the performance effects on two different CXL memory usage types, (i) moving the main storage and (ii) allocating HEX heap memory, we divide the CXL memory into two parts, (A) `fsdax` for the main storage and (B) memory only NUMA node for the HEX heap memory. Figure 2 shows the basic CXL memory configurations in this experiment. First, the baseline (I) has both the main storage and the HEX heap memory in native memory (host DRAM), which is the same as **Both DRAM** in [22]. Second, **Main FAR** (II) moves the main storage to the CXL memory and the HEX heap memory remains in the native memory, while **HEX FAR** (III) keeps the main storage in the native memory and allocates the HEX heap memory in the CXL memory.

To see the performance effect on CXL bandwidth, we use two different CXL memory setups, (1) single CXL with one CXL device and (2) double CXL with two CXL devices. First, in the configurations with a single CXL memory device, we install one 256 GB CXL device. The CXL memory is exposed as a single NUMA node with 256 GB at NUMA node 1. Second, in the configurations with two CXL memory devices, we install two CXL devices (256 GB per device), and activate CXL homogeneous interleaving upon them to double up the CXL bandwidth. Then, the CXL memory is exposed as a single NUMA node with 512 GB at NUMA node 1. In both setups, we create an `fsdax` as 50% of the CXL memory for moving the main storage. The other 50% remains in a NUMA node 1 for allocating HEX heap memory.

To combine CXL memory usage types and CXL memory setups, we use the following naming rule in the configurations. When **Main FAR** and **HEX FAR** are used in the single CXL setup, no postfix is attached in the configuration name. When **Main FAR** and **HEX FAR** are used in the double CXL setup, a postfix **x2** is attached in the configuration name. Therefore, we have five different configurations as follows. (1) **Baseline** means both the main storage and HEX heap memory remain in the native memory. (2) **Main FAR** puts the main storage in the single CXL configuration. (3) **Main FAR x2** puts the main storage in the double CXL configuration. (4) **HEX FAR** allocates HEX heap memory in the single CXL configuration. (5) **HEX FAR x2** allocates HEX heap memory in the double CXL configuration.

This experiment explores both a transactional (OLTP) and an analytical (OLAP) benchmark. One is TPC-C [48] with 100 warehouses for OLTP workloads and the other one is TPC-DS [49] with scale factor 100 for evaluating OLAP workloads. We measure the throughput of each benchmark test and the amount of CXL traffic with Intel® Performance Counter Monitor (PCM) [33]. Unlike the previous work [22], we use the officially released version of Intel PCM to monitor CXL traffic of type 3 devices in this experiment.

5.1.2 Evaluation results. First, we evaluate the TPC-C benchmark with 100 warehouses for OLTP workload evaluation. We measure the total number of transactions and CXL traffic including read/write traffic, while increasing the number of client threads within a single client process. Figure 3 shows the overall performance and CXL traffic in TPC-C. The throughput is normalized to the value at the 256-thread configuration of the baseline. The results show that the overall performance of the baseline, **Main FAR** and **HEX FAR** are the same and there is no performance degradation in CXL memory. It is well known that TPC-C has a lot of lock conflicts during transactions as mentioned in the previous work [22]. Our analysis with Intel® VTune™ Profiler confirms this high synchronization overhead too. Since highly contended OLTP workloads are sensitive to neither bandwidth nor latency of CXL memory, no performance degradation is observed in the TPC-C benchmark.

TPC-C measurements in both **Main FAR x2** and **HEX FAR x2** are excluded intentionally because the workload is not bound by the CXL memory bandwidth. In the previous work [22], the FPGA-based CXL prototype having smaller bandwidth and higher latency than the current CXL memory device also showed no significant performance degradation in TPC-C. Therefore, no meaningful performance difference from the baseline is naturally expected in the doubled CXL bandwidth.

Second, we execute the TPC-DS benchmark with scale factor 100 for OLAP workload evaluation. Figure 4 shows the overall performance and CXL traffic in TPC-DS. The throughput is normalized to the value at the 8-stream configuration of the baseline. We observe that **Main FAR** has 27% performance degradation with around 19 GB/s of CXL traffic due to the CXL bandwidth limitation. After doubling the CXL bandwidth by CXL interleaving, the performance degradation is reduced to 10% in **Main FAR x2** with around 26 GB/s of CXL traffic, which is smaller than the full bandwidth of

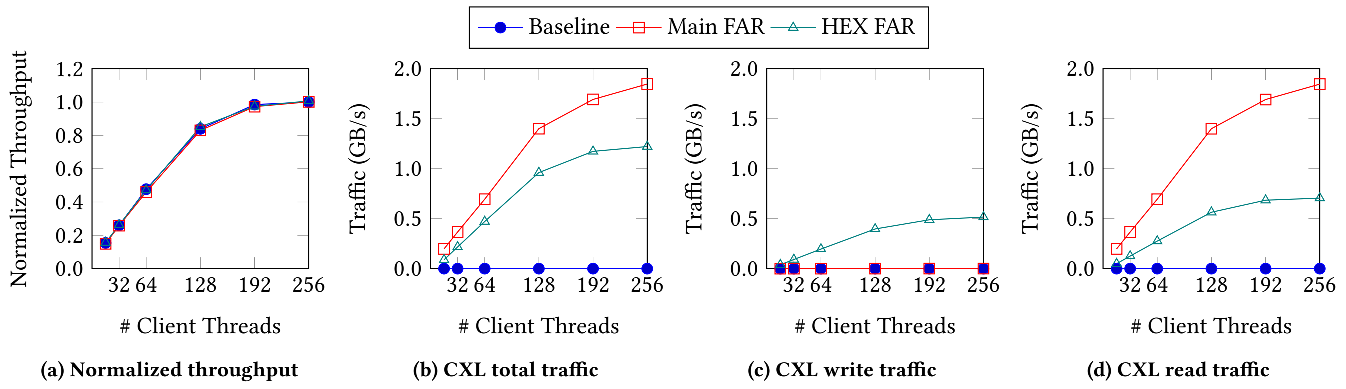


Figure 3: TPC-C throughput and CXL traffic

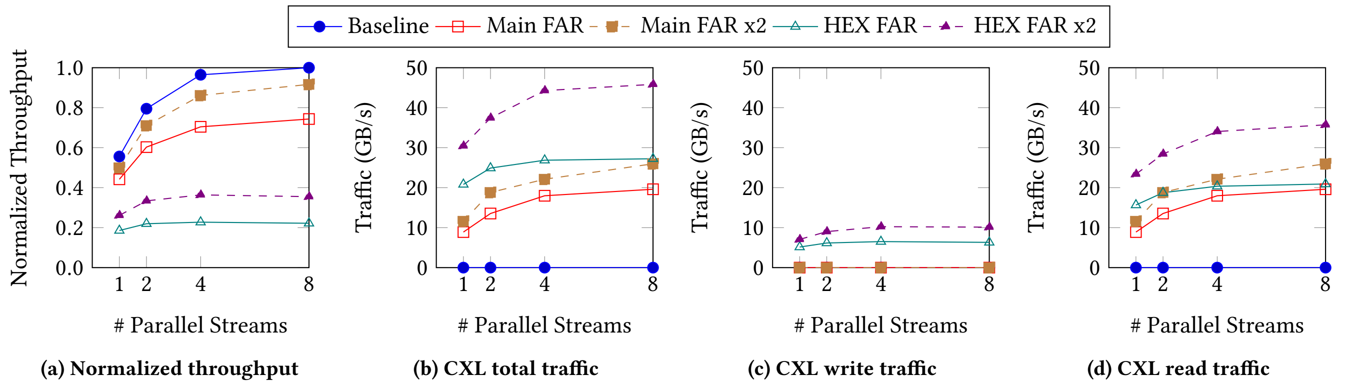


Figure 4: TPC-DS throughput and CXL traffic

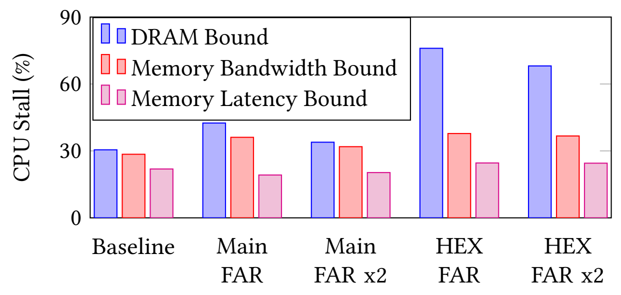


Figure 5: Microarchitecture analysis on 8-stream TPC-DS

CXL interleaving. About 10% of performance degradation is quite aligned with the emulation results in the previous work [22], where we observed about 10% performance degradation on moving the main storage into the remote NUMA node in a two-socket server. Figure 5 shows the percentage of the CPU time stalled by memory accesses during the execution of 8-stream TPC-DS according to the top-down microarchitectural analysis[55] using the Intel® VTune™ Profiler. It shows that **Main FAR x2** has lower memory bandwidth bound than **Main FAR** due to increased available bandwidth, while they have a similar amount of memory latency bound. Thus, doubling CXL bandwidth in **Main FAR x2** provides enough bandwidth for moving the main storage in this benchmark. Basically, sequential accesses have no performance degradation because prefetching hides the higher latency of sequential accesses as discussed in [2] if the CXL memory bandwidth is large enough to support all CXL traffic. However, random accesses to CXL memory may have a performance impact because of their higher latency. As discussed in [20], accessing the main storage has a small portion of random accesses to dictionaries in general. Thus, the performance degradation in **Main FAR x2** mainly comes from these random accesses.

Unlike **Main FAR**, **HEX FAR** has 78% performance degradation with around 27 GB/s of CXL traffic. Our analysis with Intel VTune Profiler shows that **HEX FAR** is much more bound by memory latency than **Main FAR**, which is primarily caused by random memory accesses of HEX heap memory. One more interesting observation is that **HEX FAR** has a larger amount of CXL traffic than **Main FAR** unlike our previous work [22]. Since the CXL memory device has 1.7x lower latency than the FPGA implementation in the previous work, it increases the amount of CXL traffic in **HEX FAR**. In addition, **HEX FAR** has CXL write traffic as well as CXL read traffic, while **Main FAR** has no CXL write traffic because it only reads the table data in the main storage. Thus, **HEX FAR** has more CXL traffic than **Main FAR** in the same CXL memory bandwidth limitation because it utilizes both the transmit data line and the

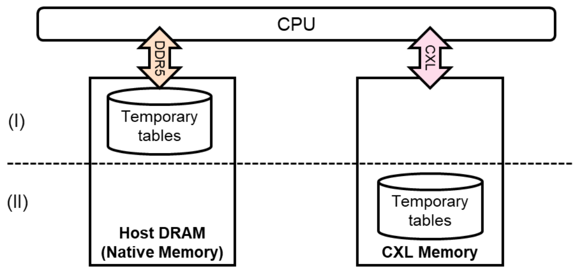


Figure 6: CXL memory configurations for temporary tables

Table 1: Temporary table usages in PaPM

Test procedure	CXL memory usage (MB)
(A)	1,335.329
(B)	628.359
(C)	600.617

Table 2: Temporary table usages in ProcBench

Procedure	CXL memory usage (MB)
custTotalLoss	3,031.582
unsatisfiedCustomersCat	1,510.766
unsatisfiedCustomerStore	2,165.340

receive data line in the PCI express bus. Our microarchitectural analysis also shows that **HEX FAR** is similarly bound by memory bandwidth with **Main FAR** due to the increased amount of CXL traffic, resulting in CXL traffic saturation in **HEX FAR**. After doubling up the CXL bandwidth by CXL interleaving, the performance degradation in **HEX FAR x2** is reduced to 65% with around 45 GB/s of CXL traffic. Increased CXL bandwidth also increases the overall throughput in **HEX FAR x2**. However, our Intel VTune analysis shows that **HEX FAR x2** has a similar amount of memory bandwidth bound with **HEX FAR**, indicating that doubled CXL bandwidth in **HEX FAR x2** is not enough to support all CXL traffic for allocating all HEX heap memory in CXL memory. In summary, **HEX FAR x2** requires more CXL bandwidth than **Main FAR x2** and the bandwidth requirement would be increased if the latency to the CXL memory device is improved.

In the evaluation with the TPC-DS benchmark, a wide range of performance degradation is observed. When moving the main storage to CXL, sequential accesses are dominant, but the bandwidth requirement is met by doubling CXL bandwidth. Thus, it has 10% performance degradation in **Main FAR x2**. When allocating HEX heap memory in CXL, random accesses are dominant and a larger amount of CXL traffic is observed due to improved latency in the CXL memory device. Consequently, it has 65% performance degradation in **HEX FAR x2**. It will be further discussed in Section 6.

5.2 Moving temporary tables to CXL memory

In this section, we present the performance evaluation of the CXL memory usage for temporary tables.

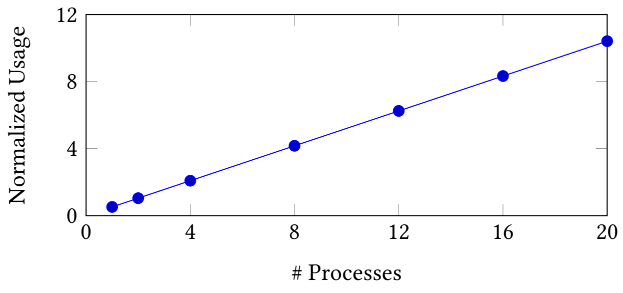


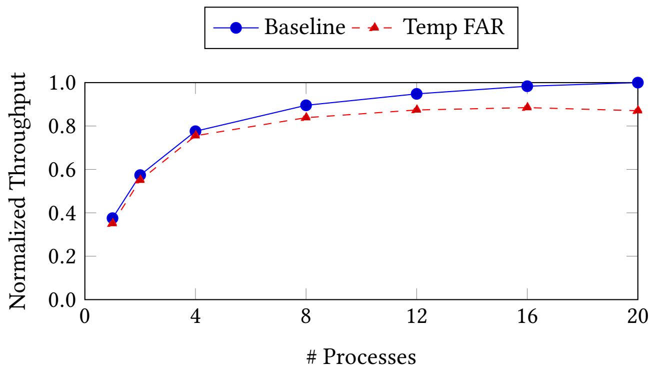
Figure 7: Normalized temporary table usage of PaPM

5.2.1 System configuration. Temporary tables are normally used in procedural workloads to keep the intermediate results and pass them to the next processing steps. As addressed in Section 3.1, numerous real-world systems suffer from memory shortages, primarily attributed to the extensive usage of large temporary tables. To investigate the performance effects on allocating the temporary tables in CXL memory, we evaluate two workloads, (1) SAP Profitability and Performance Management (PaPM) [41], a real-world scenario formerly known as Financial Services Performance Management, and (2) ProcBench [17], a publicly available benchmark test for procedural workloads.

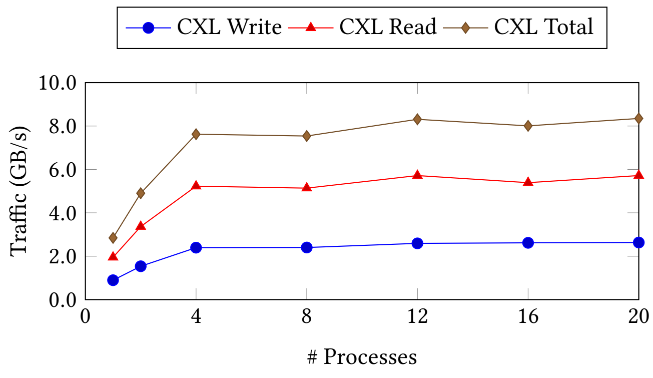
In this experiment, we use the same system as shown in Section 5.1. Our memory allocator for temporary tables has the option to utilize the designated memory only NUMA node for its temporary table placement in CXL memory like **HEX FAR**. Figure 6 shows two configurations used in this experiment. (I) **Baseline** allocates the temporary tables in the native memory (host DRAM). (II) **Temp FAR** allocates the temporary tables in the CXL memory. The main storage and the delta storage are located in the native memory as well as all the other heap memory except temporary tables.

PaPM [41] is a real-world scenario for financial services to effectively process the voluminous granular data using robust data modeling and calculation engines. This experiment uses a simpler version for expense evaluation and cost distribution to each cost center. Each process consists of three internal procedures using its own data set. The first procedure populates the required data into a base table, and the other two procedures simulate the expense evaluation and cost distribution using various simulation parameters. Table 1 shows its CXL memory usage of temporary tables for each procedure in a single process. Each process allocates its own temporary tables while executing three procedures in it. We measure the throughput while increasing the number of processes having the different user information. As shown in Fig. 7, the amount of temporary tables allocated in the CXL memory in 20 processes goes up to more than 10 times of its data size, which indicates that allocating temporary tables may cause out-of-memory situations due to the limited native memory.

ProcBench [17] is an open benchmark for procedural workloads, based on TPC-DS [49] and its history tables. We select the scale factor 100 for TPC-DS data in this experiment. To evaluate the performance impact, we select 3 procedures showing higher temporary table usages more than 1 GB in SAP HANA. Table 2 shows

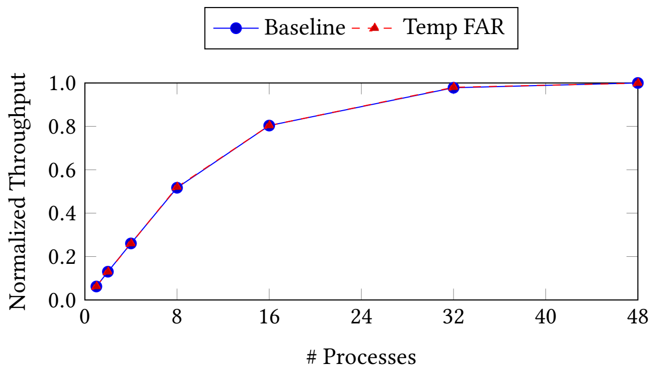


(a) Normalized throughput

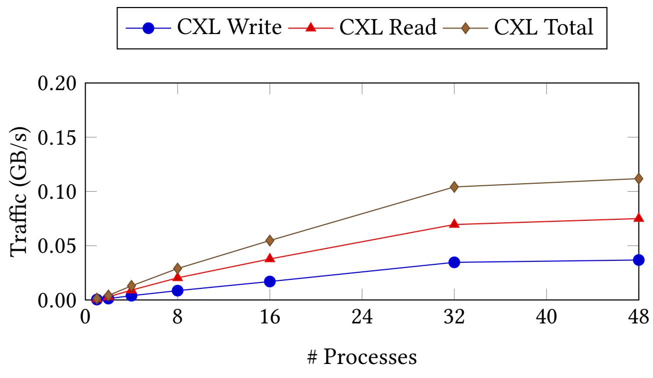


(b) CXL traffic in Temp FAR

Figure 8: PaPM throughput and CXL traffic



(a) Normalized throughput



(b) CXL traffic in Temp FAR

Figure 9: ProcBench throughput and CXL traffic

the selected procedures and their temporary table usages. For the throughput test of this benchmark, we create streams by randomly permuting the selected 3 procedures.

5.2.2 Evaluation results. Figure 8 shows the overall performance and CXL traffic in PaPM. The throughput is normalized to the value at the 20-process configuration of the baseline. The workload is data-intensive and allocates large temporary tables. However, overall throughput is limited by the mixed transactions consisting of insert and select queries against the same tables, showing less than 50% of CPU usage in the baseline. It is aligned with our Intel VTune analysis where intensive transactional interference is observed within SAP HANA. When we move temporary tables to CXL memory, 4% ∼ 13% performance degradation is observed while the amount of CXL traffic is saturated at around 8 GB/s in CXL total traffic, which is much less than available bandwidth. Our Intel VTune analysis shows that the temporary table accesses in the CXL memory contain a certain portion of random accesses sensitive to longer latency of the CXL memory. Therefore, the transactional interference limits the overall CPU usage, saturating performance throughput and the amount of CXL traffic, while a small portion of random accesses to temporary tables contributes to about 10% of performance loss.

The overall performance and CXL traffic in ProcBench is shown in Fig. 9. The throughput is normalized to the value at the 20 parallel stream configuration of the baseline. Unlike PaPM, there is no performance degradation in ProcBench. We observe a significant amount of temporary table usage in the CXL memory, but the selected procedures repeatedly allocate and deallocate a small temporary table for each iteration in their loop body. Because of their long execution time, the total amount of CXL traffic is around 0.11 GB/s, much smaller than the offered CXL bandwidth unlike PaPM. Thus, we could not find any significant performance impact in these selected procedures of ProcBench.

In summary, the performance effects on allocating temporary tables in the CXL memory would mostly depend on the amount of CXL traffic and the transactional interference within the corresponding workloads. Additionally, random accesses to the CXL memory are more bound by memory latency, resulting in more contribution to the performance degradation.

5.3 Failover with fast restart

5.3.1 System configuration. In this experiment, we use two servers with Intel 4th generation Xeon® processor Platinum 8468H codenamed *Sapphire Rapids*. Each server is equipped with two processors

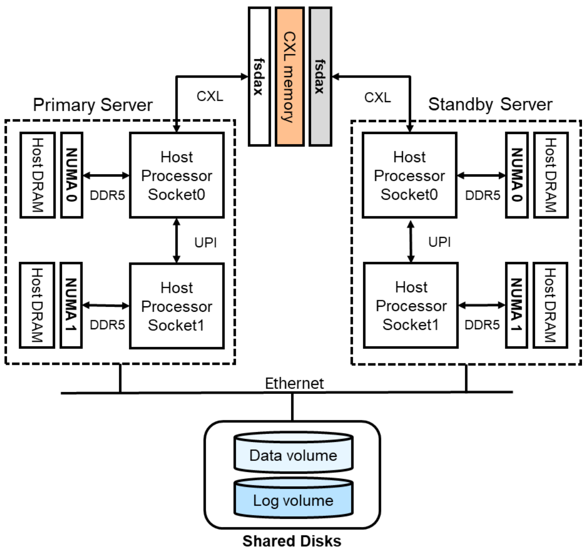


Figure 10: System overview for failover with fast restart

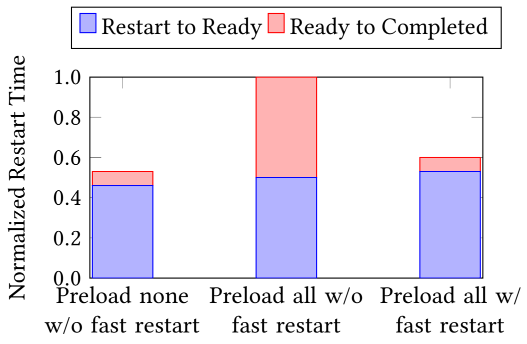


Figure 11: Normalized failover restart time

with a base frequency of 2.0 GHz and 48 cores each plus Hyper-Threading. Each processor has 128 GB in 8 channels, one 16 GB DDR5 4800 MT/s DIMM per channel. When the standby server takes over operations from the primary server, it needs to access the same database volumes. Thus, another file server for distributed file systems is needed to share database volumes, including data volumes and log volumes, between two servers. Figure 10 shows the detailed view of the experimental system. Since it is symmetric between two servers, fast restart is applicable to both i) failover to the standby server and ii) failback to the primary server.

The experimental CXL shared memory is implemented in the Altera® Agilex™ AGI027 FPGA development board which supports PCIe Gen5 x16 CXL spec 1.1 connectivity. The FPGA board hosts 2x 8GB on-board DDR4 1333 MHz memory, which is exposed to the servers as memory. It is connected to the first server by directly inserting it in a PCIe slot of the primary server with 16 PCIe lanes and connected to the standby server through two MCIO cables. In each server, we create an `fsdax` for the whole memory in the FPGA to store the table data in the main storage. In this experiment, we use TPC-H [50] with scale factor 10 due to the limited capacity of the CXL memory in our FPGA implementation.

To activate the fast restart, the data should be located in the `fsdax` instead of the native memory. Since the FPGA does not fully provide coherency on writes between servers due to the resource limitation in the FPGA, we mount the CXL memory only in one server at a time to avoid any access violation. Thus, we manually mount and unmount the `fsdax` of CXL memory during the transitions of both failover and failback. For each server, we mount the fsdax before starting the database instance and unmount it after stopping the instance.

Our scenario used in this experiment has three steps to conduct successful fast restart during failover and failback. (A) In the initial preparation step, we set up a two-host high availability system with the two servers as shown in Fig. 10. After setting up the mount path of the `fsdax` in each instance, we shut down the both database instances before mounting the `fsdax` to avoid any access violation. To load the data into the shared CXL memory, we mount the `fsdax` and start the instance in the primary server. Then, we move the main storage for table data into the `fsdax`. Since we are not considering the delta storage in this work, we execute the delta merge and savepoint to make the delta storage empty and the main storage up-to-date. (B) In the failover step, we manually shut down the database instance and unmount the `fsdax` in the primary server. Then, we mount the `fsdax` and start the instance in the standby server. (C) In the failback step, we manually shut down the database instance and unmount the `fsdax` in the standby server. Then, we mount the `fsdax` and start the instance in the primary server. After the failover and failback, we confirm the results of the whole TPC-H queries to check the consistency of the database. During the restart, each database instance records the timestamps in the trace log (1) when the instance begins the restart (Restart), (2) when the instance completes its initialization and takeover (Ready), and (3) when data preload is completed (Completed). To measure the elapsed time, we collect these timestamps and calculate the elapsed time. In this experiment, we consider only the main storage. In many real-world scenarios, the size of the delta storage is much smaller than that of the main storage because the delta storage is periodically merged into the main storage. Thus, failover spends most of time in recovering the main storage, and the performance impact on the delta storage during failover is negligible.

5.3.2 Experimental results. Figure 11 shows the normalized restart time during failover to the standby server. In the first configuration, fast restart is not activated and none of the table data is set to preload during the restart. Then, it spends most of its time to initialize the database instance during the restart. In the second configuration, fast restart is not activated, but all the data is set to preload during the restart. Since the table data is not set to place in the CXL shared memory, it takes nearly half of the restart time to preload the table data from the data volumes to the local DRAM. In this experiment, a gigabit Ethernet is used between servers because of the limitation of our test machines. The third configuration activates fast restart and all the data is set to preload during the restart. It has the table data in the `fsdax` of the CXL shared memory before failover. Thus, it saves 85% of data preloading time, totally reducing 40% of the restart time, compared to the second configuration because there is no need to load the data again. Compared to the first configuration, it takes only 7% longer time during the restart because of the consistency check of the table data in the CXL shared memory. However, the first configuration could take more time for the initial query. Since the data are not yet loaded in the memory, it needs to load the table data during its first execution, resulting in higher latency and no benefit on the short restart time.

Our experiment shows 40% reduction in the restart time because of the small data size (4.7 GB) of TPC-H scale factor 10. The larger the database size is, the longer time it takes to preload all the table data into the native memory. Thus, fast restart can achieve more gains as the database size is increased. For example, it is estimated to save 95% of data preloading time in TPC-H scale factor 100, reducing around 84% of the restart time in the emulation with the same distributed file systems. In summary, failover with fast restart using the CXL shared memory can reduce a significant amount of the restart time during failover, improving overall system availability.

We are expecting further performance enhancement in failover with fast restart with ASIC-based commercial CXL memory because the CXL memory used in Section 5.1 yields 1.6x lower latency and 2.6x larger bandwidth (1.3x per device) than our FPGA implementation. Our previous work [22] showed that the FPGA-based CXL memory has around 40% performance degradation in TPC-H SF10 when moving the main storage. However, the commercial CXL memory achieves up to 10% performance degradation in TPC-DS SF100 in the same use case.

6 DISCUSSION

To alleviate the constraint of limited memory capacity in a server, we evaluated dynamic memory expansion leveraging CXL memory. We analyzed the performance impact according to memory access patterns against the main storage, operational memory, and temporary tables used in SAP HANA, and shared our insights on which data structures should move to CXL memory. Each data structure showed a different performance impact according to workloads.

Performance impact for OLTP and OLAP There was no negative impact for OLTP workloads due to transactional interference and small CXL traffic. It is well known as the inherent characteristics of OLTP workloads, with regardless of the data structures. However, we observed a wide range of performance degradation for OLAP workloads due to the limited bandwidth and long latency of CXL memory. Accessing table data showed less performance degradation because sequential accesses are dominant and prefetching hides the long latency of the CXL memory device. Its performance is limited by available bandwidth of CXL memory. Accessing operational data showed more performance degradation because random accesses are dominant and they are sensitive to the latency.

We showed the different performance impact according to the workloads and data objects within an IMDBMS. Based on the performance impact guided by our experimental results, we can move the data objects having less performance impact first (like the main storage) to CXL memory and estimate the possible performance impact of the selective placement of these database objects. Even though SAP HANA can place table data in CXL memory per column, we may further investigate the performance impact by moving more fine-grained data objects, such as dictionaries or value arrays, to a remote memory as a next study, as we did in our previous work [20].

Not only bandwidth but also latency The performance impact is significantly reduced with improved latency and bandwidth. Performance degradation from sequential accesses is easily improved by increasing the offered bandwidth of the memory devices. Performance degradation from random accesses cannot be simply improved by reducing the access latency because the improved latency also increases the amount of traffic resulting in the performance saturation due to the bandwidth limitation in the memory device. More importantly, most data structures in database management systems have both sequential accesses and random accesses [28]. Therefore, it is imperative to improve both latency and bandwidth of CXL memory devices.

Overall TCO reduction Dynamic memory expansion via CXL memory also contributes to several operational benefits in IMDBMSs. It reduces the amount of host memory consumption in each server or each virtual machine, which alleviates the stranded memory issue improving the overall system utilization. In addition, it will contribute to system-wide lower TCO by relieving the overprovisioning issue and accepting smaller and cheaper DIMMs.

Fast restart at the industrial scale As a resolution of the long restart time to reload data to memory in IMDBMSs, we employ CXL shared memory. Section 5.3 showed that CXL 3.0 enables failover with fast restart by sharing the table data in the CXL memory with the independent power domain. It was impossible with the conventional systems with local DRAMs and persistent memory because they are a part of a host and their contents cannot be accessed once the host suffers from any failure. As one of the representative use cases of CXL 3.0, we presented failover with fast restart among multiple servers in a complex IMDBMS, reducing the restart time during failover or failback and improving overall system availability. In this work, we implemented software-managed coherency to test our failover scenario. If the enhanced coherency is fully provided, failover and failback could be automatically conveyed without any human involvement. This part remains as future work.

7 RELATED WORK

Several innovative approaches utilizing Compute Express Link (CXL) technology have emerged regarding simple memory expansion. Al Maruf et al. [29] highlighted the potential of Transparent Page Placement (TPP) for CXL-enabled tiered memory. Despite promising performance rivaling ideal systems, the technology needs further refinement to manage less frequently accessed 'cold' page data. Subsequently, Sun et al. [46] present an in-depth research study comparing the performance of actual CXL memory and its emulated counterpart. While their findings prompt reconsideration of previous assessments of emulated CXL, they also signal that the real-world performance of their proposed CXL-memory-aware dynamic page allocation technology remains uncertain.

Similarly, Benson et al. [5] indicate the need for future systems to grasp various memory performance aspects for optimal overall performance. However, their study is limited by the absence of tests involving actual CXL devices. Lastly, Park et al. [32] present a CXL memory expander offering high-bandwidth access to remote memory resources, yet its practical scalability in complex systems calls for further empirical validation.

In the domain of CXL memory pools, numerous studies have explored the potential of CXL technology. Aguilera et al. [1] make a compelling case for memory disaggregation, though they identify software compatibility hurdles to be addressed. Approaching from a different angle, Gouk et al. [16] trace the development of DirectCXL, a system enabling direct access between the host processor complex and remote memory resources via CXL's memory protocol. Nevertheless, this system's feasibility and scalability call for further verification. Li et al. [25] introduces Pond, a CXL-based full-stack memory pool suitable for cloud deployment; however, challenges persist in predicting memory latency and large-scale resource management. Similarly, Wahlgren et al. [51] evaluate CXL-enabled memory pooling for high-performance computing systems, but their study needs empirical research using real CXL memory hardware. Lastly, Yang et al. [54] assessed a CXL-enabled hybrid memory pool, and while they showed promising results, further testing is needed to confirm their consistency and long-term effectiveness.

Some studies offer insights on adapting to this new paradigm in the context of DBMS with disaggregated memory. Notably, Lerner and Alonso [24] discuss the potential impact of the CXL specification on database engines and data processing systems. However, they note that integrating existing systems with CXL's capabilities remains challenging. In a similar context, Wang et al. [52] highlight the potential of memory disaggregation to influence distributed shared-memory databases positively. However, implementing such systems presents challenges, particularly with DSM layer design, multi-node concurrency control, buffer management, and index design.

Several studies highlight different findings when discussing accelerators with global cache coherency via CXL. Dally et al. [11] explore the promise of domain-specific hardware accelerators and the efficiency of parallelism through specialization. However, meeting the demands of reduced memory bandwidth requires significant algorithmic changes. Meanwhile, Zhao et al. [57] stress the need for comprehensive research on DSI (Data Storage and Ingestion) pipeline scaling for large-scale ML training infrastructures. Sim et al. [45] propose an innovative solution using computational CXL memory to accelerate memory-intensive applications, but its scalability and adaptability still require verification. Lastly, Kwon et al. [18] propose a fault-tolerant training method utilizing persistent memory disaggregation over CXL technology, whose broad-spectrum adaptability needs further exploration.

In the realm of near-data processing, several studies have explored various techniques and technologies. Lee et al. [23] introduce a modular Solid-State Drive (SSD) architecture and the concept of Database Kernels (DBKs) using CXL technology, but practical implementation of DBKs requires more research. Meanwhile, Maschi and Alonso [30] conducted an in-depth study on using FPGA-based accelerators on commercial search engines, revealing that the imbalance between system CPUs limits improvements and could make deployments economically less attractive.

Lastly, research papers focused on GPU functionalities and data management [27, 56] have shed light on promising co-processing strategies for extensive data management on GPUs and techniques to enhance data warehousing queries' performance on GPU systems. However, they underline the reliance on fast interconnects and suggest that simply investing in superior GPU hardware may not significantly improve query performance.

8 CONCLUSION

We presented the two use cases of CXL memory for IMDBMSs to alleviate two fundamental constraints of IMDBMSs. First, we evaluate dynamic memory expansion using the commercial CXL memory devices as the solution for the limited memory space in a server. Performance measurements using various workloads showed a wide range of performance degradation according to the data structures and memory access patterns. No performance impact was observed with OLTP workloads. However, OLAP workloads have performance degradation. Moving table data to CXL memory shows less performance degradation than allocating operational data. Second, we showed that CXL 3.0 enables fast restart during failover or failback between two servers, reducing the long restart time and improving overall system availability. We presented our implementation details to incarnate CXL shared memory, and confirmed that memory sharing among multiple hosts allow IMDBMSs to share the data in CXL memory. In conclusion, we emphasize that CXL is not only a straightforward memory disaggregation technology but also plays a crucial role in reducing total cost of ownership and enabling fast restarts in many use cases.

IMDBMSs deployed in the cloud have more requirements to share the data among multiple servers or instances, such as result buffers and file caches. As future work, we aim to explore the additional use cases using CXL shared memory. Moreover, we expect that CXL switches could play an important role in the flexible disaggregated memory system. Investigation of the performance impact on the CXL switches is a good candidate for our future work. In addition, investigation on hardware-managed tiered memory techniques, such as Intel Flat Memory Mode [58], is one of our crucial future work to automatically place the data according to their temperature, reducing the overall access latency and improving throughput in dynamic memory expansion.

ACKNOWLEDGEMENT

The authors would like to thank Hojun Shim, Seungwoo Lim, Kyumin Park, Youjin Jang, and Byeonghun Hwang at Samsung Electronics for providing us commercialized CXL memory devices. We also thank Mehul Wagle, Sergej Hardock, and Jinsub Ahn at SAP for their dedicated supports in SAP HANA.

REFERENCES

[1] Marcos K Aguilera, Emmanuel Amaro, Nadav Amit, Erika Hunhoff, Anil Yelam, and Gerd Zellweger. 2023. Memory disaggregation: Why now and what are the challenges. *ACM SIGOPS Operating Systems Review* 57, 1 (2023), 38–46.
[2] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna T. Malladi, and Yang-Seok Ki. 2022. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *DaMoN*. ACM, 8:1–8:5. https://doi.org/10.1145/3533737.3535090
[3] Mihnea Andrei, Christian Lemke, Günter Radestock, et al. 2017. SAP HANA adoption of non-volatile memory. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1754–1765. https://doi.org/10.14778/3137765.3137780
[4] AsteraLabs. 2024. *Breaking Through the Memory Wall*. https://www.asteralabs.com/general/breaking-through-the-memory-wall/
[5] Lawrence Benson, Marcel Weisgut, and Tilmann Rabl. 2023. What We Can Learn from Persistent Memory for CXL. *BTW 2023* (2023).

[6] Daniel S. Berger, Daniel Ernst, Huaicheng Li, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Lisa Hsu, Ishwar Agarwal, Mark D. Hill, and Ricardo Bianchini. 2023. Design Tradeoffs in CXL-Based Memory Pools for Public Cloud Platforms. *IEEE Micro* 43, 2 (2023), 30–38. https://doi.org/10.1109/MM.2023.3241586

[7] Prakash Chauhan and Mahesh Wagh. 2022. *CXL Memory Challenges.* https://hc34.hotchips.org/assets/program/tutorials/CXL/Hot%20Chips%202022%20CXL%20MemoryChallenges.pdf

[8] Compute Express Link Consortium. 2019. *CXL.* https://www.computeexpresslink.org/

[9] CXL. 2024. CXL 3.1 Specification. https://computeexpresslink.org/cxl-specification/

[10] CXL. 2024. Past CXL Specifications. https://computeexpresslink.org/past-cxl-specifications/

[11] William J Dally, Yatish Turakhia, and Song Han. 2020. Domain-specific hardware accelerators. *Commun. ACM* 63, 7 (2020), 48–57.

[12] Franz Faerber, Alfons Kemper, Per-Åke Larson, Justin J. Levandoski, Thomas Neumann, and Andrew Pavlo. 2017. Main Memory Database Systems. *Found. Trends Databases* 8, 1-2 (2017), 1–130. https://doi.org/10.1561/1900000058

[13] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. 2012. The SAP HANA Database–An Architecture Overview. *IEEE Data Eng. Bull.* 35, 1 (2012), 28–33.

[14] The LINUX Foundation. 2021. *Direct Access for files.* https://www.kernel.org/doc/Documentation/filesystems/dax.txt

[15] Andreas Geyer, Daniel Ritter, Dong Hun Lee, Minseon Ahn, Johannes Pietrzyk, Alexander Krause, Dirk Habich, and Wolfgang Lehner. 2023. Working with Disaggregated Systems. What are the Challenges and Opportunities of RDMA and CXL?. In *BTW (LNI, Vol. P-331).* Gesellschaft für Informatik e.V., 751–755. https://doi.org/10.18420/BTW2023-47

[16] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct access, High-Performance memory disaggregation with DirectCXL. In *USENIX ATC.* 287–294. https://www.usenix.org/conference/atc22/presentation/gouk

[17] Surabhi Gupta and Karthik Ramachandra. 2021. Procedural Extensions of SQL: Understanding their usage in the wild. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1378–1391.

[18] Miryeong Kwon, Junhyeok Jang, Hanjin Choi, Sangwon Lee, and Myoungsoo Jung. 2023. Failure Tolerant Training With Persistent Memory Disaggregation Over CXL. *IEEE Micro* 43, 2 (2023), 66–75.

[19] Robert Lasch, Suleyman S. Demirsoy, Norman May, Veeraraghavan Ramamurthy, Christian Färber, and Kai-Uwe Sattler. 2020. Accelerating Re-Pair Compression Using FPGAs. In *Proceedings of the 16th International Workshop on Data Management on New Hardware* (Portland, Oregon) *(DaMoN '20).* Association for Computing Machinery, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/3399666.3399931

[20] Robert Lasch, Thomas Legler, Norman May, Bernhard Scheirle, and Kai-Uwe Sattler. 2022. Cost modelling for optimal data placement in heterogeneous main memory. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2867–2880. https://www.vldb.org/pvldb/vol15/p2867-lasch.pdf

[21] Donghun Lee, Andrew Chang, Minseon Ahn, Jongmin Gim, Jungmin Kim, Jaemin Jung, Kang-Woo Choi, Vincent Pham, Oliver Rebholz, Krishna Malladi, et al. 2020. Optimizing Data Movement with Near-Memory Acceleration of In-memory DBMS.. In *EDBT.* 371–374.

[22] 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 (DaMoN '23).* ACM, 35—-43. https://doi.org/10.1145/3592980.3595311

[23] 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.*

[24] Alberto Lerner and Gustavo Alonso. 2024. CXL and the Return of Scale-Up Database Engines. arXiv:2401.01150 [cs.DB]

[25] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 2023. Pond: Cxl-based memory pooling systems for cloud platforms. In *ASPLOS.* ACM, 574–587. https://doi.org/10.1145/3575693.3578835

[26] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F Wenisch. 2012. System-level implications of disaggregated memory. In *IEEE International Symposium on High-Performance Comp Architecture.* IEEE, 1–12.

[27] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Pump up the volume: Processing large data on GPUs with fast interconnects. In *SIGMOD.* 1633–1649.

[28] Stefan Manegold, Peter Boncz, and Martin L Kersten. 2002. Generic database cost models for hierarchical memory systems. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases.* Elsevier, 191–202.

[29] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit O. Kanaujia, and Prakash Chauhan. 2022. TPP: Transparent Page Placement for CXL-Enabled Tiered Memory. *CoRR* abs/2206.02878 (2022). https://doi.org/10.48550/arXiv.2206.02878 arXiv:2206.02878

[30] Fabio Maschi and Gustavo Alonso. 2023. The Difficult Balance Between Modern Hardware and Conventional CPUs. In *Proceedings of the 19th International Workshop on Data Management on New Hardware.* ACM, Seattle WA USA, 53–62. https://doi.org/10.1145/3592980.3595314

[31] J. McGlone, P. Palazzari, and J. B. Leclere. 2018. Accelerating Key In-memory Database Functionality with FPGA Technology. In *ReConFig.* 1–8. https://doi.org/10.1109/RECONFIG.2018.8641722

[32] S. J. Park, H. Kim, K.-S. Kim, J. So, J. Ahn, W.-J. Lee, D. Kim, Y.-J. Kim, J. Seok, J.-G. Lee, H.-Y. Ryu, C. Y. Lee, J. Prout, K.-C. Ryoo, S.-J. Han, M.-K. Kook, J. S. Choi, J. Gim, Y. S. Ki, S. Ryu, C. Park, D.-G. Lee, J. Cho, H. Song, and J. Y. Lee. 2022. Scaling of Memory Performance and Capacity with CXL Memory Expander. In *2022 IEEE Hot Chips 34 Symposium (HCS).* 1–27. https://doi.org/10.1109/HCS55958.2022.9895633

[33] Intel® PCM. 2023. *Intel® Performance Counter Monitor.* https://github.com/intel/pcm

[34] Hasso Plattner. 2009. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD.* ACM, 1–2. https://doi.org/10.1145/1559845.1559846

[35] Hasso Plattner. 2014. The impact of columnar in-memory databases on enterprise systems: implications of eliminating transaction-maintained aggregates. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1722–1729.

[36] PMDK. 2004. *Persistent Memory Development Kit.* https://pmem.io/pmdk/

[37] Iraklis Psaroudakis, Florian Wolf, Norman May, Thomas Neumann, Alexander Boehm, Anastasia Ailamaki, and Kai-Uwe Sattler. 2015. Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. *Performance Characterization And Benchmarking: Traditional To Big Data* 8904 (2015), 16. 97–112. https://doi.org/10.1007/978-3-319-15350-6_7

[38] SAP. 2012. *translytical-data-platforms-forrester-wave-sap-a-leader.* https://news.sap.com/2022/12/translytical-data-platforms-forrester-wave-sap-a-leader/

[39] SAP. 2024. *Persistent Memory in SAP HANA.* https://help.sap.com/docs/SAP_HANA_PLATFORM/6b94445c94ae495c83a19646e7c3fd56/1f61b13e096d4ef98e62c676debf117e.html

[40] SAP. 2024. *SAP HANA Fast Restart Option.* https://help.sap.com/docs/SAP_HANA_PLATFORM/6b94445c94ae495c83a19646e7c3fd56/ce158d28135147f099b761f8b1ee43fc.html

[41] SAP. 2024. *SAP Profitability and Performance Management.* https://www.sap.com/products/financial-management/profitability-and-performance-management.html

[42] SAP. 2024. *Why SAP data and analytics?* https://www.sap.com/products/technology-platform/analytics.html

[43] Reza Sherkat, Colin Florendo, Mihnea Andrei, Rolando Blanco, Adrian Dragusanu, Amit Pathak, Pushkar Khadilkar, Neeraj Kulkarni, Christian Lemke, Sebastian Seifert, Sarika Iyer, Sasikanth Gottapu, Robert Schulze, Chaitanya Gottipati, Nirvik Basak, Yanhong Wang, Vivek Kandiyanallur, Santosh Pendap, Dheren Gala, Rajesh Almeida, and Prasanta Ghosh. 2019. Native store extension for SAP HANA. *Proc. VLDB Endow.* 12, 12 (aug 2019), 2047–2058. https://doi.org/10.14778/3352063.3352123

[44] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. 2012. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In *SIGMOD.* ACM, 731–742. https://doi.org/10.1145/2213836.2213946

[45] Joonseop Sim, Soohong Ahn, Taeyoung Ahn, Seungyong Lee, Myunghyun Rhee, Jooyoung Kim, Kwangsik Shin, Donguk Moon, Euiseok Kim, and Kyoung Park. 2023. Computational CXL-Memory Solution for Accelerating Memory-Intensive Applications. *IEEE Comput. Archit. Lett.* 22, 1 (2023), 5–8. https://doi.org/10.1109/LCA.2022.3226482

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

[47] Inc. the Linux Kernel Organization. 2024. *NUMA memory policy.* https://www.kernel.org/doc/Documentation/vm/numa_memory_policy.txt

[48] TPC-C. 2023. *TPC-C.* https://www.tpc.org/tpcc/

[49] TPC-DS. 2023. *TPC-DS.* https://www.tpc.org/tpcds/

[50] TPC-H. 2023. *TPC-H.* https://www.tpc.org/tpch/

[51] Jacob Wahlgren, Maya Gokhale, and Ivy B Peng. 2022. Evaluating Emerging CXL-enabled Memory Pooling for HPC Systems. In *2022 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC).* IEEE, 11–20.

[52] Ruihong Wang, Jianguo Wang, Stratos Idreos, M. Tamer Özsu, and Walid G. Aref. 2022. The case for distributed shared-memory databases with RDMA-enabled memory disaggregation. *Proc. VLDB Endow.* 16, 1 (sep 2022), 15–22. https://doi.org/10.14778/3561261.3561263

,

[53] Marcel Weisgut, Daniel Ritter, Martin Boissier, and Michael Perscheid. 2022. Separated Allocator Metadata in Disaggregated In-Memory Databases: Friend or Foe?. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS Workshops 2022, Lyon, France, May 30 - June 3, 2022*. IEEE, 1202–1208. https://doi.org/10.1109/IPDPSW55747.2022.00207

[54] Qirui Yang, Runyu Jin, Bridget Davis, Devasena Inupakutika, and Ming Zhao. 2022. Performance Evaluation on CXL-enabled Hybrid Memory Pool. In *NAS*. IEEE, 1–5. https://doi.org/10.1109/NAS55553.2022.9925356

[55] 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)*. IEEE, 35–44.

[56] Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2013. The Yin and Yang of processing data warehousing queries on GPU devices. *Proceedings of the VLDB Endowment* 6, 10 (Aug. 2013), 817–828. https://doi.org/10.14778/2536206.2536210

[57] Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, et al. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: Industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 1042–1057.

[58] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, 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 *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 37–56. https://www.usenix.org/conference/osdi24/presentation/zhong-yuhong