Computational CXL-Memory Solution for Accelerating Memory-Intensive Applications

Joonseop Sim, Soohong Ahn, Taeyoung Ahn, Seungyong Lee, Myunghyun Rhee, Jooyoung Kim, Kwangsik Shin, Donguk Moon, Euiseok Kim, and Kyoung Park

Abstract—CXL interface is the up-to-date technology that enables effective memory expansion by providing a memory-sharing protocol in configuring heterogeneous devices. However, its limited physical bandwidth can be a significant bottleneck for emerging data-intensive applications. In this work, we propose a novel CXL-based memory disaggregation architecture with a real-world prototype demonstration, which overcomes the bandwidth limitation of the CXL interface using near-data processing. The experimental results demonstrate that our design achieves up to 1.9× better performance/power efficiency than the existing CPU system.

Index Terms—Compute express link (CXL), near-data-processing (NDP)

1 Introduction

The data explosion in emerging applications such as big data and AI requires high bandwidth memory with large capacity, demanding an innovative architectural shift [1], [2], [3]. The number of servers and memory devices has kept increasing to meet the overwhelming memory requirement; however, it is not a scalable solution due to cost and resource inefficiency [4]. In addition, since the conventional system has memory devices dedicated to a processor, it poses the issue of a resource imbalance in responding to the needs of various applications [5].

'Memory disaggregation' is a promising architectural solution that decouples memory from compute nodes [6], [7]. It allows the system designers to expand additional memory capacity independent of each server flexibly while meeting the memory requirements of user applications [8]. For example, the server under high memory pressure uses the far memory from other nodes in the disaggregated group. This approach can consequently manage resources more efficiently than the conventional dedicated CPU/memory architecture.

Recent efforts on CXL (Compute Express Link) are a key enabler in accelerating the architecture shift for the memory disaggregation. CXL is an industry-supported cache-coherent interconnect (CCI) for various processors to efficiently expand memory capacity with a memory-semantic protocol [9]. Unlike the existing DDR interface entirely dependent on the host CPU, the memory connected with CXL allows the handshaking communication to include additional value (e.g., data processing engine) into the memory [10].

However, there is a critical technical challenge in deploying the CXL memory – the limited interface bandwidth compared to the host DDR memory[11]. Comparing the bandwidth per capacity ratio of DDR and CXL memory, DDR4 and DDR5 have $0.4s^{-1}$(= $25.6GB/s/64GB$) and $0.6s^{-1}$(= $38.4GB/s/64GB$), respectively, whereas CXL memory has a very limited bandwidth of 0.0625 s^{-1}(= $32GB/s$(@PCIe5.0×8) / $512GB$). In many memory-intensive applications, e.g., KNN (K-nearest neighbor search), the target application of our system, the bandwidth constraint can be a critical factor that limits the system performance.

In this paper, we propose "*CMS: Computational CXL-Memory Solution for Memory-intensive Applications*". CMS offers a fast, scalable, and low-cost add-on solution to the conventional CPU scale-out system, improving the performance and power efficiency of memory-intensive applications. Our CMS exploited the CXL interconnect to expand the memory capacity for modern applications that require large amounts of data. To address the bandwidth challenge, we employed a novel NDP (Near Data Processing) logic that maximizes the internal bandwidth. Our contributions are summarized as follows:

- We designed a CXL-based memory expansion prototype with real-world deployment, which supports load/store accessibility with 512GB capacity.
- We designed an NDP core inside the CXL memory, eliminating the performance bottleneck by reducing data movement through the CXL interface.
- We propose a load balancer that fully utilizes the internal bandwidth of CMS by effectively interleaving memory channels with a performance-optimized MAC operator that hides an accumulator latency.

2 Related Work

Prior work on disaggregated memory [5], [12], [13], [14] exposed capacity as a memory pool, allowing it to be shared by multiple machines. However, their work relied on RDMA over InfiniBand or Ethernet, which have limited latency with a high overhead associated with host involvement. In contrast, our work utilizes the emerging CXL protocol for the memory disaggregation solution, enabling memory sharing (or pooling) with high performance while reducing software stack complexity.

The works in [15], [16] proposed a memory expansion system of IMDBMS (In-memory database management system) using a CXL memory device. The author's group successfully demonstrated a prototype of CXL type 3 memory expansion devices in E3.S form factor using CXL.mem and CXL.io commands. However, the CXL-attached memory suffers the bandwidth due to its physical limitation, as discussed in Section 1. Our design overcomes this limitation by adopting an NDP logic to fully utilize the disaggregated memory's internal bandwidth.

There have also been many proposals for NDP technologies to address the data movement issue between processor and memory. For example, the work in [17], [18] offloads the scan operation of the in-memory database system and the embedding operation in the recommendation system, respectively, both of which are limited by memory bandwidth. They place their processing units into the buffer in the DIMM module, increasing the internal bandwidth compared to the DIMM interface by enabling parallelism between ranks within the DIMM. However, the RCD (Register Clock Driver), which acts as a buffer for the RDIMM (Registered DIMM), is located on the rank-shared bus and cannot perform exact parallel operations between ranks independently [19]. In contrast, our work supports scalable internal bandwidth since it can increase the bandwidth in proportion to the number of modules/DIMMs in the memory nodes.

3 CMS Architecture

3.1 Overall Architecture

Fig. 1 shows the overall system, including our CMS prototype. It consists of a host CPU, DDR memory, and the CMS prototype

• *The authors are with SK Hynix, 17336 Icheon, Gyonggi, South Korea. E-mail: {joonseop.sim, soohong.ahn, taeyoung.ahn, seungyong.lee, myunghyun.rhee, jooyoung2.kim, kwangsik.shin, donguk.moon, euiseok.kim, kyoung.park}@sk.com.*

Manuscript received 31 October 2022; accepted 27 November 2022. Date of publication 5 December 2022; date of current version 11 January 2023.
(Corresponding authors: Joonseop Sim.)
Digital Object Identifier no. 10.1109/LCA.2022.3226482

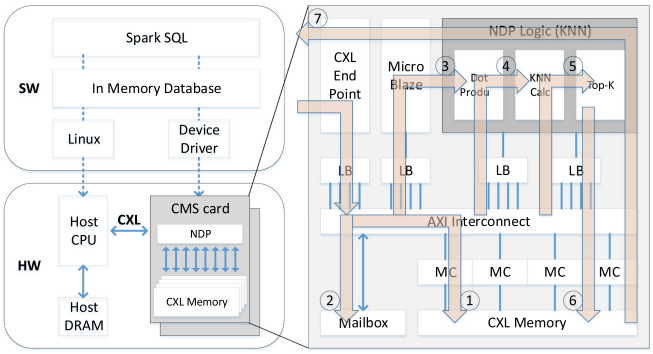

Fig. 1. CMS architecture and operational flow.

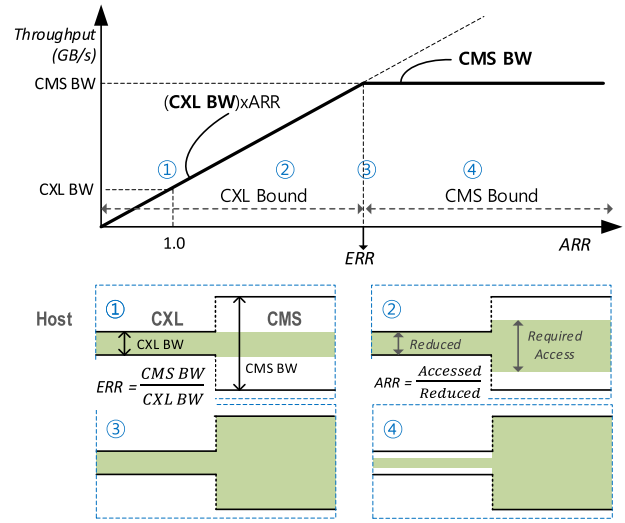

Fig. 2. Performance boundness of CXL-enabled CMS.

connected through the CXL interface. The CMS prototype contains a CXL controller, internal DDR memory, and an NDP engine that allows the CXL memory to fully utilize the bandwidth despite the limited bandwidth of the CXL.

Fig. 1 shows a detailed block diagram of our CMS card. We prototyped the CMS card using Xilinx Alveo U250. We synthesized the offloading logic implemented in RTL and MicroBlaze of Xilinx, a soft processor for command parsing, with the CXL 2.0 IP. Xilinx mailbox IP is used for bidirectional inter-processor communication. The operation flow of CMS is as follows. Note that we implemented KNN as an initial sample, but the offloading logic can be flexibly extended for other applications. The dataset for the KNN operation has initially been loaded into CXL memory. Once a KNN searching request is issued, it sends the query to CXL memory ① with its address to the mailbox ②. Then, Micro Blaze parses the query ③ and performs the KNN operation while reading the query and data ④ ⑤. The result is written to the predetermined address in memory ⑥ so that the host can read it back ⑦.

3.2 Throughput Optimization Components

For many memory-intensive workloads, memory bandwidth is critical in determining the system performance [20]. As shown in Fig. 2, the factors limiting the performance may vary over the amount of data reduction ratio (RR) during the execution of the target application. In this paper, we defined the RR in two different ways: *Application RR (ARR)* and *Effective RR (ERR)*. The *ARR* is a software parameter determined by the application algorithm, defined as the ratio of accessed data size to reduced. The *ERR* is the reduction capability that hardware can support, defined as the ratio of *CMS BW* (bandwidth) to *CXL BW*. In the region where $ARR < ERR$ (① and ② in Fig. 2), the effective throughput can be represented as $CXL\ BW \times ARR$, which means the performance is limited to the CXL bandwidth (**CXL-Bound**). In the region where ARR > ERR (④ in Fig. 2), on the other hand, the amount of data processed by CMS is high, and the data size going through CXL is low. Thus, the performance is limited to the CMS bandwidth (**CMS-Bound**). For many cases of memory-intensive workloads, it is clear that our system operates in the CMS-bound region since *ARR* is larger than the *ERR* (e.g., 4.75 of *ERR versus*. 80000 of *ARR* from our KNN experimental setup in Section 4). In this case, raising the upper bound of region ④ by maximizing the utilization of CMS BW and approaching ERR to ARR is critical for performance. To this end, the following subsections propose two key components of our CMS to improve the CMS BW utilization to provide high ERR.

3.2.1 Load Balancer

Integrating DDR memory with an NDP logic using the CXL interface requires the protocol conversions of CXL-AXI-DDR interfaces. Here, we found performance degradation points due to the interleaving methods, especially when loading contiguous data into multiple channels of DDR memory through the CXL interface. First, in the case of no interleaving applied, it fills continuous data in one channel and then the other channels sequentially. In this case, NDP can use only single-channel bandwidth, which makes only $1/n$ (=number of memory channels) of the total memory BW utilized. Second, in the case of using interleaving by changing the AXI address mapping, interleaving less than 4KB is not allowed due to the address boundary constraint of the AXI specification. As a result, NDP still utilizes only 1/n of bandwidth at a given time. Third, software at the host CPU can perform interleaving. In this case, the programmer should manually rearrange the data at the user space to store contiguous memory addresses of table data in an interleaved fashion across memory channels. Also, it may pose high software overhead, e.g., *memcpy*.

To address this issue, we designed a novel hardware logic, *load balancer (LB)*, which *host-transparently* distributed data across channels in a 64-byte memory access unit. As shown in Fig. 3 (Down), our LB remaps the contiguous address space per-channel to across-channels. It maximizes memory bandwidth utilization by interleaving each channel in units of DDR access granularity. Since we implemented LB in hardware logic, it relieves the burden of the host CPU in memory mapping, having better performance than

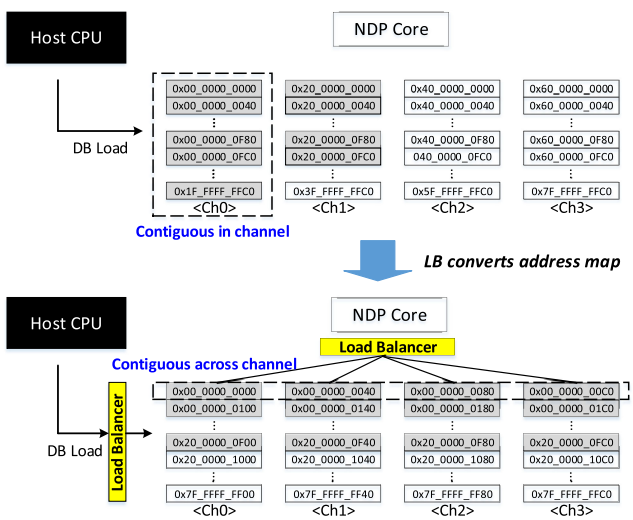

Fig. 3. Effect of load balancer (LB).

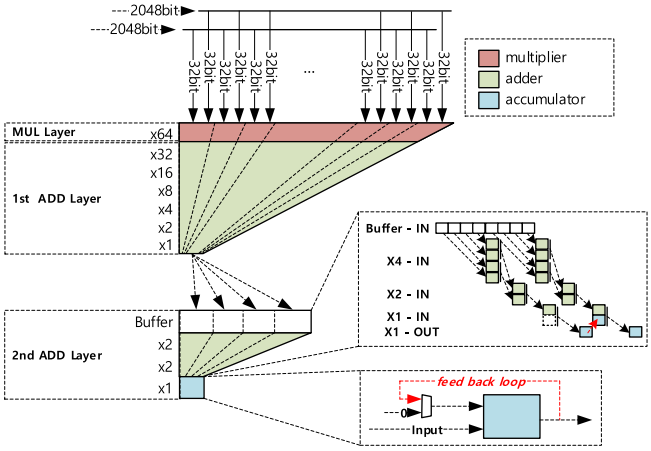

Fig. 4. MAC operation flow of proposed CMS.

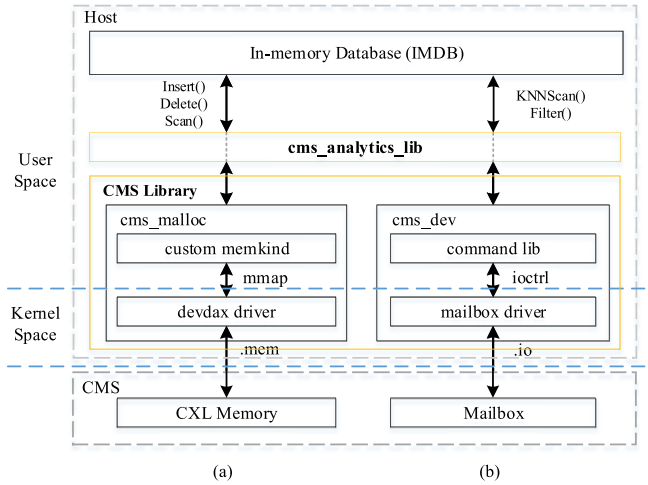

Fig. 5. CMS software library for (a) memory allocation path, and (b) mailbox communication path.

the data management involving software. We discuss our experimental results on the LB effect in Section 4.2.

3.2.2 MAC Block Optimization

Since 2048-bit data is seamlessly input to NDP Logic by the previously described LB logic, the MAC operator of the next stage needs an architecture that can process 64x FP32 data (= 2048 bits) without performance degradation. However, the typical MAC accumulator has a critical performance degradation factor due to the inherent latency of the feedback loop structure.

Our proposed MAC architecture effectively hides the latency of the accumulator to prevent performance degradation. Fig. 4 presents the MAC operation flow of our CMS. In our current prototype, the MAC operator supports 64 FP32 execution. It performs the computation with three main steps: MUL layer, 1st ADD layer, and 2nd ADD layer. The MUL layer multiplies 64 operations and 64 data in parallel. Then, the two ADD layers perform the cascaded addition to obtain the aggregated results of 64 executions.

During the design of our prototype, we found that an undesired latency delay called *throughput bubble* may occur due to the feedback loop block of the final stage when obtaining the result of the MAC operation through a single cascaded sequence. We addressed this issue with an optimized accumulation block by positioning a 4-depth buffer between the 1st and 2nd ADD layers. As shown in Fig. 4, our design buffers the result of the 1st ADD layer, and the 2nd ADD layer performs the addition in units of x4 to get the final accumulated value. It successfully hides the operation time of the feedback loop in the last stage during the buffering time, so it can operate seamlessly without blocking the input.

3.3 Software Architecture

We built Apache Arrow [21] based software API, named *cms_analytics_lib*, enabling the elastic applicability of our CMS design to general data analysis platforms. As shown in Fig. 5, our API defines *Insert*, *Delete*, *Scan*, and *KNNScan* functions to communicate with the in-memory database. The *cms_analytics_lib* has *cms_malloc* and *cms_dev* stack as lower layers to control the CXL memory and NDP unit, respectively. The *cms_malloc* stack allocates and frees memory, managing the related metadata in the CMS memory with the CXL.mem protocol. The *cms_dev* stack controls the NDP logic while reading and writing the mailbox of the CMS via the CXL.io protocol. We implemented the read operation for the mailbox based on polling instead of the interrupt to achieve higher performance.

4 Evaluation

4.1 Experimental Setup

Fig. 6 shows our prototype configuration. We prototyped the CMS card using Xilinx Alveo U250 FPGA with four channels of DDR memory (DDR4/2400Mbps/128GB). We synthesized the KNN operation logic implemented in RTL and MicroBlaze of Xilinx for command parsing. To evaluate the effectiveness of the proposed CMS, we built three different design configurations: 1) `CPU_only`, 2) `CME (CXL Memory Expansion)`, and 3) `CMS`. `CPU_only` is the conventional system case consisting of a CPU (Intel's Future Xeon Processor) and two channels of DDR memory (DDR5/4800Mbps/64GB). We measure the power consumption of the evaluated systems using VTS PM [22]. We experimented with the KNN benchmarks [23] where a feature vector is a 256-dimensional vector of FP32-type features. For the KNN queries, each system compared the input feature vectors with various ranges from million database features and extracted the most similar 25 entries.

4.2 Experimental Results and Analysis

Fig. 7a shows the execution times of our CMS and two reference systems, `CPU_only` and `CME`, normalized to `CPU_only` for different DB sizes at K=25. The results show that CMS outperforms `CPU_only` and `CME` by 3.23× and 3.1×, respectively, for 2M of the database size. As discussed in Section 3.2, since data processing does not occur in the CXL memory (RR=1) in the case of `CME`, the CXL bandwidth entirely limits the system performance. However, our CMS can improve the performance even in this case. It is because the NDP on CMS accesses and processes massive data on behalf of the hosts, reducing the data size communicated through

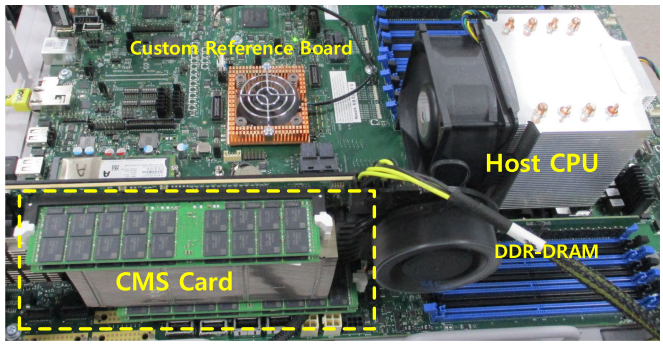

Fig. 6. FPGA prototype of proposed CMS card.

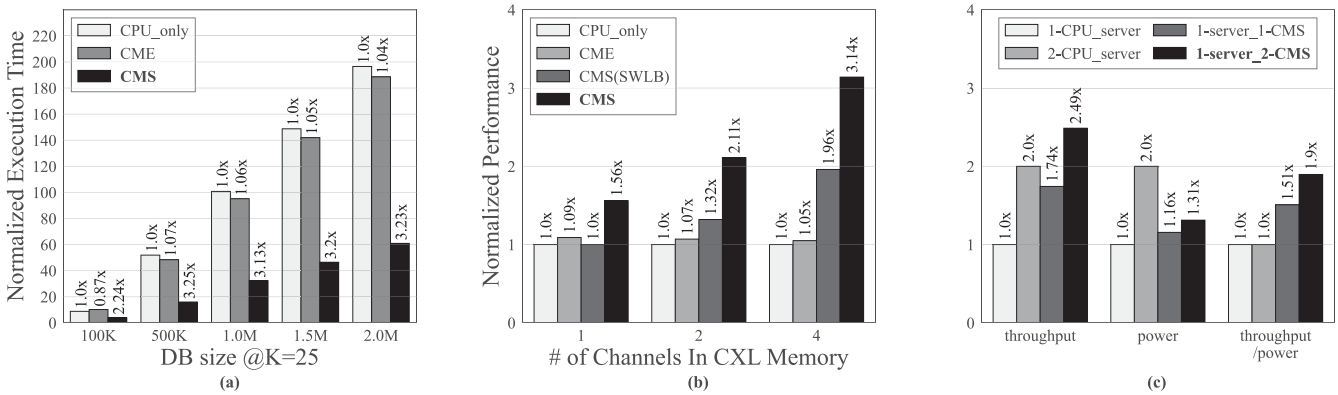

Fig. 7. Performance and power consumption between CPU_only versus CME versus. CMS: (a) Execution time for different DB sizes (b) performance for different channels (c) power efficiency.

CXL significantly, i.e., returning only the computation results with the small size.

We next verify the CMS scalability. Fig. 7b shows how the performance changes over the increased channel numbers of the CXL memory. In the case of `CME`, even if the number of channels of the CXL memory increases from 1 CH (channel) to 4 CH, there is almost no performance improvement effect. In contrast, the performance on the proposed CMS proportionally increases as the number of channels (=bandwidth) of the CXL memory increases. It implies that the effective throughput proportionally increases, showing that CMS using the NDP can fully utilize internal memory bandwidth despite CXL's limited bandwidth.

Fig. 7b also shows the impact of the LB component on the system performance. While CMS includes our proposed LB logic described in Section 3.2 by default, `CMS (SWLB)` refers to a case in which the host performs interleaving between the CXL memory channels on the software instead. We observe that the software-based channel interleaving can burden the host, resulting in 1.6× performance degradation.

Fig. 7c shows the power efficiency of the proposed CMS (Scale-up) with a cluster of CPU servers (Scale-out). The result shows that CMS offers significant at-scale power savings. For example, a single CPU server with two CMS cards consumes 34% less power and shows 90% better throughput/power efficiency than a cluster of two single CPU servers. Thus, we conclude that CMS is a promising solution with TCO (*Total Cost of Ownership*) competitiveness compared to existing servers in the latest application market that pressures memory expansion.

5 Conclusion

In this work, we proposed an NDP-enabled CXL memory for memory-intensive applications with real-world prototype demonstration. Our design addresses the issue of the data movement bottleneck in CXL by adopting an optimized NDP core and supplying a larger memory node capacity. The experimental results show that our design achieves up to 3.2 times improvement in execution time and 34% energy saving compared to the conventional CPU system.

References

[1] C. Wang et al., "Panthera: Holistic memory management for big data processing over hybrid memories," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2019, pp. 347–362.

[2] S. Shukla et al., "A scalable multi-teraops core for ai training and inference," *IEEE Solid-State Circuits Lett.*, vol. 1, no. 12, pp. 217–220, Dec. 2018.

[3] R. Godse et al., "Memory technology enabling the next artificial intelligence revolution," in *Proc. IEEE Nanotechnol. Symp.*, 2018, pp. 1–4.

[4] "5 simple ways to avoid energy waste in your data center," 2022. [Online]. Available: https://www.energystar.gov/products/identify_and_remove_unused_servers

[5] H. Li et al., "First-generation memory disaggregation for cloud platforms," 2022, *arXiv:2203.00241*.

[6] Z. Guo et al., "Clio: A hardware-software co-designed disaggregated memory system," in *Proc. 27th ACM Int. Conf. Archit. Support Program. Lang. Operat. Syst.*, 2022, pp. 417–433.

[7] D. Gouk, S. Lee, M. Kwon, and M. Jung, "Direct access,{High-Performance } memory disaggregation with {DirectCXL}," in *Proc. USENIX Annu. Tech. Conf.*, 2022, pp.1 287–294.

[8] L. Liu et al., "Memory disaggregation: Research problems and opportunities," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1664–1673.

[9] CXL, 2022. [Online]. Available: https://www.computeexpresslink.org/

[10] D. D. Sharma, "A low latency approach to delivering alternate protocols with coherency and memory semantics using PCI express® 6.0 PHY at 64.0 GT/s," in *Proc. IEEE Symp. High-Perform. Interconnects*, 2021, pp. 35–42.

[11] H. A. Maruf et al., "TPP: Transparent page placement for CXL-enabled tiered memory," 2022, *arXiv:2206.02878*.

[12] I. Calciu et al., "Rethinking software runtimes for disaggregated memory," in *Proc. 26th ACM Int. Conf. Archit. Support Lang. Oper. Syst.*, pp. 79–92, 2021.

[13] Y. Gao et al., "When cloud storage meets {RDMA}," in *Proc. 18th USENIX Symp. Networked Syst. Des. Implementation*, 2021, pp. 519–533.

[14] H. A. Maruf et al., "Memtrade: A disaggregated-memory marketplace for public clouds," 2021, *arXiv:2108.06893*.

[15] D. Lee et al., "Optimizing data movement with near-memory acceleration of in-memory DBMS," in *Proc. Int. Conf. Extending Database Technol.*, 2020, pp. 371–374.

[16] M. Ahn et al., "Enabling CXL memory expansion for in-memory database management systems," in *Data Management on New Hardware*, New York, NY, USA: ACM, 2022.

[17] D. Lee et al., "Improving in-memory database operations with acceleration DIMM (AxDIMM)," in *Data Management on New Hardware*. New York, NY, USA: ACM, 2022.

[18] L. Ke et al., "Near-memory processing in action: Accelerating personalized recommendation with AxDIMM," *IEEE Micro*, vol. 1, no. 1, pp. 116–127, Jan./Feb. 2022.

[19] RCD, 2022. [Online]. Available: https://www.rambus.com/blogs/get-ready-for-ddr5-dimm-chipsets/

[20] J. H. Kim et al., "Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond," in *Proc. IEEE Hot Chips 33 Symp.*, 2021, pp. 1–16.

[21] Apache arrow, 2022. [Online]. Available: https://arrow.apache.org/

[22] VTS PM user guide, 2022. [Online]. Available: http://www.senanetworks.co.kr/download/manual/manual_VTSPM_kr-v1.0.2.pdf

[23] KNN benchmark, 2022. [Online]. Available: https://github.com/mnms/ltdb-knn-kernel-bench