CXL Memory as Persistent Memory for Disaggregated HPC: A Practical Approach

Yehonatan Fridman
Ben-Gurion University, NRCN, Israel
fridyeh@post.bgu.ac.il

Suprasad Mutalik Desai, Navneet Singh
Intel, India
{suprasad.desai,navneet.singh}@intel.com

Thomas Willhalm
Intel, Germany
thomas.willhalm@intel.com

Gal Oren
Technion, NRCN, Israel
galoren@cs.technion.ac.il

ABSTRACT

In the landscape of High-Performance Computing (HPC), the quest for efficient and scalable memory solutions remains paramount. The advent of Compute Express Link (CXL) introduces a promising avenue with its potential to function as a Persistent Memory (PMem) solution in the context of disaggregated HPC systems. This paper presents a comprehensive exploration of CXL memory's viability as a candidate for PMem, supported by physical experiments conducted on cutting-edge multi-NUMA nodes equipped with CXL-attached memory prototypes. Our study not only benchmarks the performance of CXL memory but also illustrates the seamless transition from traditional PMem programming models to CXL, reinforcing its practicality.

To substantiate our claims, we establish a tangible CXL prototype using an FPGA card embodying CXL 1.1/2.0 compliant endpoint designs (Intel FPGA CXL IP). Performance evaluations, executed through the STREAM and STREAM-PMem benchmarks, showcase CXL memory's ability to mirror PMem characteristics in *App-Direct* and *Memory Mode* while achieving impressive bandwidth metrics with Intel 4th generation Xeon (Sapphire Rapids) processors.

The results elucidate the feasibility of CXL memory as a persistent memory solution, outperforming previously established benchmarks. In contrast to published DCPMM results, our CXL-DDR4 memory module offers comparable bandwidth to local DDR4 memory configurations, albeit with a moderate decrease in performance. The modified STREAM-PMem application underscores the ease of transitioning programming models from PMem to CXL, thus underscoring the practicality of adopting CXL memory.

The sources of this work are available at: https://github.com/Scientific-Computing-Lab-NRCN/STREAMer.

KEYWORDS

CXL, Memory disaggregation, Persistent Memory (PMem), Intel Optane DCPMM, HPC, STREAM, STREAM-PMem, STREAMer

1 INTRODUCTION

1.1 Current HPC Memory Solutions Limitations

As the era of Exa-Scale computing unfolds, the demand for analyzing, manipulating, and storing massive amounts of data intensifies [60]. Exascale systems are designed to meet these demands and enable the execution of a broad spectrum of computations, ranging from loosely to tightly coupled tasks, including CFD simulations and deep learning optimizations [11]. Memory and storage resources play a crucial role in the performance and scalability of these computations [32]. Memory factors such as capacity, latency, and bandwidth are responsible for successfully handling extensive tasks and delivering data to processing units promptly [45]. In scientific computing, storage devices hold significance for preserving diagnostics throughout computations [37]. Notably, the growing frequency of failures in exascale machines emphasizes the significance of storing vast data volumes to support recovery and bolster fault tolerance [8, 14].

However, the traditional memory and storage hierarchy in HPC systems reveals notable gaps that impose critical constraints on scientific computations [32]. From the vantage point of memory architecture, DRAM has inherent limitations of bandwidth and capacity that impact performance and prevent the processing of large-scale problems [56, 58]. From the storage perspective, traditional devices (such as HDDs and SSDs) provide large capacities but exhibit very slow access times, leading to significant overheads for I/O-bound applications and fault tolerance mechanisms [37]. These gaps and limitations of traditional hardware highlight the ongoing endeavors to expand the memory-storage hierarchy and develop novel memory architectures and solutions. A notable example is Non-Volatile RAM [59, 71] (on which we elaborate in subsection 1.2).

While High-Bandwidth Memory (HBM) [28] has been introduced as a solution to enhance memory performance, it doesn't entirely alleviate the problem [29]. HBM memory modules are stacked vertically, allowing for higher memory bandwidth due to their increased parallelism. However, even with HBM, the memory capacity remains limited compared to conventional DDR (Double Data Rate) memory modules [67]. This limitation can still lead to constraints in memory-intensive applications that require larger memory spaces [44]. Moreover, while HBM addresses the bandwidth issue to some extent, it doesn't eliminate the underlying problem of memory hierarchy [44]. The processor still needs to access different memory levels, and the latency of transferring data between these levels can impact performance [44]. HBM improves bandwidth between the processor and certain memory modules, but the need to access different levels of memory introduces latency that can affect the execution of various tasks [44].

In general, it is possible to proclaim that the conventional approach of locating memory modules directly on the board poses significant challenges in the context of HPC systems [10]. This arrangement restricts memory bandwidth due to the limited number of connections between the processor and these modules [10]. As

a result, the data transfer rate between the processor and board-mounted memory becomes a bottleneck, hindering the overall performance of the system [10].

For an increase of memory capacity outside of the node, advanced communication technologies such as the Remote Direct Memory Access (RDMA) based Message Passing Interface (MPI) optimize inter-node communication [35]. However, these sophisticated frameworks are not devoid of challenges [17]: MPI, a cornerstone for distributed computing communication, contends with latency and overhead issues during message transmission, disproportionately affecting efficiency for applications requiring frequent communication. Furthermore, the management complexity escalates with the cluster's scale due to heightened contention for network resources among a larger node count [7].

1.2 Persistent Memory in HPC

A proposed solution aimed at bridging the gap between memory and storage is Persistent Memory (PMem) [33, 49]. PMem implementations such as BBU (battery backed up) DIMM or Non-Volatile RAM (NVRAM) aim to deliver rapid byte-addressable data access alongside persistent data retention across power cycles. PMem technologies establish a new tier within the memory-storage hierarchy by combining memory and storage characteristics [59, 71]. Basic solutions include battery-backed DRAM and have been accessible from diverse vendors over a significant timeframe, representing an established concept [30, 39, 50, 63]. However, these solutions face challenges due to limited scalability and potential data loss risks. The reliance on batteries introduces concerns regarding power failures, leading to potential data corruption or loss if batteries deplete. Moreover, the approach's scalability is hampered by the need for individual batteries for each module, impacting cost-effectiveness and overall system performance.

Yet, in recent years new PMem technologies have emerged, with 3D-Xpoint [19] being the main technology and Intel Optane DCPMM [21, 72] the prominent product on the market. These modern PMem technologies offer byte-addressable memory in larger capacities compared to DRAM while maintaining comparable access times [27]. Moreover, as these technologies are non-volatile in nature, they enable data retrieval even in instances of power failures. Moreover, PMem offers two configuration options based on these characteristics: (1) It can be utilized as main memory expansion, providing additional volatile memory, and (2) it can serve as a persistent memory pool that can be accessed by applications via a PMem-aware file system [71] or be managed and accessed directly by applications [27]. To simplify and streamline PMem programming and management, the Persistent Memory Development Kit (PMDK) was created [64].

During recent years, PMem has gained significant traction in HPC applications [15, 48, 55, 62], with two direct use cases of PMem for scientific applications that require no (or minimal) changes to applications. The first use-case involves PMem as memory expansion to support the execution of large scientific problems [48]. The second use case involves leveraging PMem as a fast storage device accessed by a PMem-aware file system (mainly based on the POSIX API), primarily for application diagnostics and checkpoint restart (C/R) mechanisms [38], but also for increasing the performance and inherent fault tolerance of scientific applications [14].

In addition to the direct use cases of PMem in scientific applications, various frameworks and algorithms were developed to access and manage data structures on PMem [4]. Among these are primary methods that are built on top of the PMDK library [14, 31]. For example, persistent memory object storage frameworks such as MOSIQS [31] and the NVM-ESR recovery model for exact state reconstruction of linear iterative solvers using PMem [14].

Nevertheless, as HPC workloads advance, computing units evolve, and onboard processing elements increase, the demand for heightened memory bandwidth becomes essential [32]. Existing PMem solutions demonstrate notable shortcomings in meeting these requirements, showing limitations in scalability beyond a certain threshold [15]. Specifically, PMem devices exhibit limited bandwidth. For instance, the bandwidth of Optane DCPMM for reading and writing is multi-factor lower than that of DRAM [27]. This, in part, is connected with the hybrid and in-between properties of a PMem module [18], as schematically described in Table 1.

Adding to these challenges, a significant limitation arises from the physical attachment of most PMem devices, like Optane DCPMM, to the CPU board through memory DIMMs. This configuration restricts the potential for memory expansion, as PMem contends for DIMM slots alongside conventional DRAM cards, presenting a bottleneck to achieving optimal memory configurations [51, 69]. The HPC community as a whole — both the super and cloud computing [61] — recognizes the drawbacks associated with tight integrating memory and compute resources, particularly in relation to capacity, bandwidth, elasticity, and overall system utilization [10, 57]. PMem technologies that are tightly coupled with the CPU inherit these limitations. Now, as prominent PMem technologies are phased out (Optane DCPMM, for example, as announced in 2022 [20, 22]), there is an active and prominent pursuit for the adoption of novel memory solutions in particular, and a strive to achieve more disaggregated computing in general [36].

1.3 Dissagregated Memory with CXL

The emergence of discrete memory nodes housing DRAM and network interface controllers (NICs) is anticipated to revolutionize conventional memory paradigms, facilitating distributed and shared memory access and reshaping HPC landscapes [10]. This shift aligns with the concept of disaggregation, where compute resources and memory units are decoupled for optimized resource utilization, scalability, and adaptability.

The concept of memory disaggregation has been facilitated recently by the development of advanced interconnect technologies, exemplified by Compute Express Link (CXL) [66]. CXL is an open standard to support cache-coherent interconnect between a variety of devices [66]. After its introduction in 2019, the standard has evolved and continues to be enhanced. CXL 1.1 defines the protocol for three major device types [66]: Accelerators with cache-only (type 1), cache with attached memory (type 2), and memory expansion (type 3). CXL 2.0 expands the specification – among other capabilities – to memory pools using CXL switches on a device level. CXL 3.0 introduces fabric capabilities and management,

Property	As a main memory extension	As a direct access to persistent memory
Volatility	Volatile in memory extension mode	Non-volatile in direct access mode
Access	Cache-coherent memory expansion	Transactional byte-addressable object store
Capacity	Higher than main memory volume	Lower than storage volume
Cost	Cheaper than the main memory	More expansive than storage
Performance	Several factors below main memory bandwidth	High bandwidth compared to storage

Table 1: Properties of PMem modules, either as a memory extension (*Memory Mode)* or as a direct access PMem (*App-Direct*).

improved memory sharing and pooling with dynamic capacity capability, enhanced coherency, and peer-to-peer communication. Bandwidth-wise, CXL 1.1 and 2.0 employ PCIe 5.0, achieving 32 GT/s for transfers up to 64 GB/s in each direction via a 16-lane link. On the other hand, CXL 3.0 utilizes PCIe 6.0, doubling the speed to 64 GT/s, supporting 128 GB/s bi-directional communication via an x16 link.

Since the market of CXL memory modules is emerging, several vendors have announced products using the CXL protocol. For example, Samsung [52] and SK Hynix [53] introduce CXL DDR5 modules, AsteraLabs [3] announced a CXL memory accelerator, and Montage Technology [68] will offer a CXL memory expander controller.

Leveraging CXL, memory nodes will be interconnected through high-speed links, enabling adaptive memory provisioning to compute nodes in real time [70]. The practice of intra-rack disaggregation holds the potential to effectively address the memory demands of applications while concurrently ensuring an adequate supply of efficient remote memory bandwidth [46, 47]. Figure 1 demonstrates the expected phase change from the processor's point of view, from previous years' DDR4+PMem memory access, equipped with NVMe SSDs via the PCIe Gen4, to the upcoming future of DDR5 local memory equipped with local or remote NVMe SSDs and CXL memory for memory expansion or persistency over the new generations of PCIe.

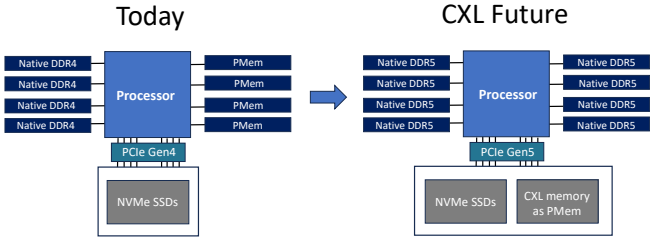


Figure 1: The migration from PMem as hardware to CXL memory as PMem in future systems.

Nevertheless, while the concept of memory disaggregation with technologies like CXL holds significant promise, it is important to acknowledge that there are still challenges and considerations that need to be addressed [2, 16]; challenges and considerations that resemble the ones of persistent memory integration in HPC [5]. For example, software and programming models need to evolve to take advantage of disaggregated memory fully; Applications and algorithms must be designed or adapted to work seamlessly across distributed memory nodes; and efficient data placement and movement strategies are crucial to minimize the impact of network latencies and ensure that data-intensive workloads can effectively utilize CXL-based disaggregated memory resources, especially when cache-coherence or direct access is enabled. Notwithstanding, when comparing CXL memory aspects to the ones of PMem as non-volatile RAM (NVRAM), in general, it can be observed (Table 2) that from the disaggregated HPC usage perspective, there should be a prevalence to CXL over NVRAM considering bandwidth, data transfer, and scalability, but also considering memory coherency, integration, pooling and sharing.

1.4 Contribution

In this work, based on actual physical experiments with multi-NUMA nodes and multi-core high-performance SOTA hardware (subsection 2.1) and CXL-remote memory (subsection 2.2), we claim that it is not only possible to exemplify most persistent memory modules characteristics (as described in Table 1) with CXL memory fully but also that in terms of performance, we can achieve much better bandwidth than previously published Optane DCPMM ones (such in [26], which, for a single Optane DCPMM, discovers that its max read bandwidth is 6.6 GB/s, whereas its max write bandwidth is 2.3 GB/s). In fact, we show (Figure 4) that by approaching our CXL-DDR4 memory module – much cheaper than DDR5 – we achieve comparable results to the local DDR4 module and exhibit performance degradation of only about 60% in bandwidth in comparison to local DDR5 module access (noting that DDR4 has about 50% bandwidth of DDR5). Our tests were made in multiple configurations (subsection 3.2) in relation to the memory distance from the working threads using the well-known STREAM benchmark (subsection 3.1).

In order to demonstrate the non-volatile properties of the memory as PMem, the CXL memory was located outside of the node, in an FPGA device (subsection 2.2), potentially backed by battery, like previous battery-backed DIMMs. As many nodes can approach the device, the battery-backed consideration is no longer considered by us as a major overhead since it will be applied only once for the memory modules and not in each compute node.

Moreover, besides the cache-coherent performance benchmarks with STREAM [40, 41], we retested the memory bandwidth in an equivalent of the *App-Direct* approach with a modified STREAM application, named STREAM-PMem [12] when all of the main arrays were allocated as a PMDK's *pmemobj* and manipulated accordingly [65]. *pmemobj* provides an assurance that the condition of objects will remain internally consistent regardless of when the program concludes. Additionally, it offers a *transaction* function that can encompass various modifications made to persistent objects. This function ensures that either all of the modifications are successfully applied or none of them take effect.

Aspect	CXL Memory	NVRAM
Bandwidth & Data Transfer	Significantly higher bandwidth enabling fast data transfers between processors and memory devices.	Non-volatile storage with potential data transfer rate limitations due to underlying interface and technology.
Memory Coherency	Provides memory-coherent links, ensuring consistent data across different memory tiers.	Requires additional mechanisms for memory coherency, except with local RAM, when integrated with other memory technologies.
Heterogeneous Memory Integration	Allows seamless integration of various memory technologies within a unified architecture.	Effective for extending memory capacity, but integration may require additional considerations due to unique characteristics.
Memory Pooling and Sharing	Facilitates memory pooling and sharing, enabling efficient resource utilization and dynamic allocation based on workload requirements.	Extends memory capacity, but inherent flexibility in memory sharing and pooling may be limited.
Industry Standardization	Open industry standard supported by major technology players, ensuring compatibility, interoperability, and broader adoption.	Solutions may vary, potentially leading to compatibility challenges and limited integration options.
Scalability	Architecture designed for scalability with multiple lanes and protocols, catering to evolving data center needs.	Scalability may be constrained by underlying technology characteristics, such as DIMM count and RAM/NVRAM tradeoff.
Relevance to HPC	Higher bandwidth, memory coherency, and memory pooling capabilities enhance HPC workload performance. Standardization compatibility in heterogeneous environments and scalability cater to evolving demands.	Offers non-volatility but is constrained by limitations in bandwidth, coherency management, and scalability, affecting its applicability to complex HPC memory needs.

Table 2: General comparison between common aspects of CXL memory and NVRAM for disaggregated HPC.

We stress that as our CXL memory module is located outside of the node and can be backed by a battery, the ability to transactionally and directly access the memory, exactly as previously done with Optane DCPMM, while achieving even better performances, is a key to our practical approach, which consider CXL memory as a persistent memory for the future of disaggregated HPC.

Finally, we open-sourced the entire benchmarking methodology as an easy-to-use and automated tool named STREAMer for future CXL memory device evaluations for HPC purposes.

2 PHYSICAL EXPERIMENTAL SETUP

2.1 HPC hardware

Our HPC hardware experimental environment is based on 2 setups:

(1) Node equipped with two Intel 4^{th} generation Xeon (Sapphire Rapids) processors with a base frequency of 2.1GHz and 48 cores each, plus Hyper-Threading. BIOS was updated to support only 10 cores per socket. Each processor has one memory DIMM (64GB DDR5 4800MHz DIMM). The system is equipped with a CXL prototype device, implemented as DDR4 memory on a PCIe-attached FPGA (see Figure 2).

(2) Node equipped with two Intel Xeon Gold 5215 processors with a base frequency of 2.5GHz and 10 cores each, plus Hyper-Threading. Each processor has total 96GB DRAM in 6 channels, 16GB DDR4 2666MHz DIMM per channel. (see Figure 3).

2.2 CXL prototype

We provide an in-depth overview of our CXL prototype's implementation on an FPGA card [25]. Figure 2 and Figure 4 grant a more detailed view into the implementation of our CXL memory pool on the FPGA card (while Figure 3 show the reference system, without any CXL attachment, with DDR4 main memory). The prototype aims to harness the capabilities of the R-Tile Intel FPGA IP for CXL, encompassing critical functionalities for CXL link establishment and transaction layer management. This comprehensive solution facilitates the construction of FPGA-based CXL 1.1/2.0 compliant endpoint designs, including Type 1, Type 2, and Type 3 configurations. It's built upon a previously proven prototype, with necessary slight modifications for PMem activity [34].

The architecture of our CXL implementation revolves around a synergistic pairing of protocol Soft IP within the FPGA main fabric die and the Hard IP counterpart, the R-Tile. This cohesive arrangement ensures effective management of CXL link functions, which are pivotal for seamless communication. Specifically, the R-Tile interfaces with a CPU host via a PCIe Gen5x16 connection, delivering a theoretical bandwidth of up to 64GB/s. As a key facet of our implementation, the FPGA device is duly enumerated as a CXL endpoint within the host system.

Complementing this link management, the Soft IP assumes the mantle of transaction layer functions, vital for the successful execution of different CXL endpoint types. For Type 3 configurations, the CXL.mem transaction layer adeptly handles incoming CXL.mem requests originating from the CPU host. It orchestrates the generation of host-managed device memory (HDM) requests directed toward an HDM subsystem. Simultaneously, the CXL.io transaction layer undertakes the responsibility of processing CXL.io requests. These requests encompass both configuration and memory space inquiries initiated from the CPU host, seamlessly forwarding them to their designated control and status registers. A noteworthy augmentation is the User Streaming Interface, offering a conduit for custom CXL.io features that can be seamlessly integrated into the user design.

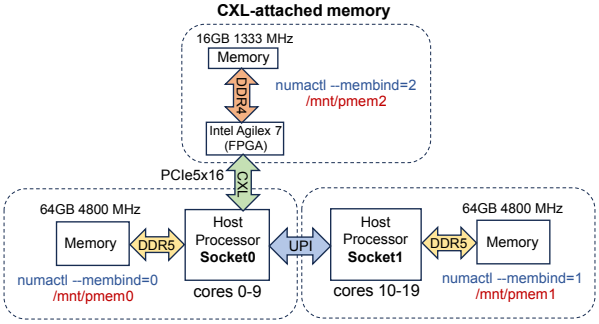


Figure 2: Setup #1 with DDR5 on-node memory and DDR4 CXL-attached memory.

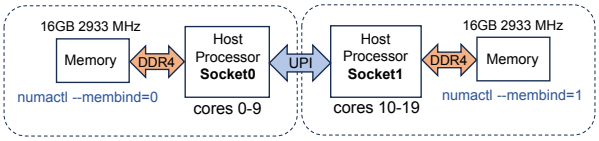


Figure 3: Setup #2 with DDR4 on-node memory.

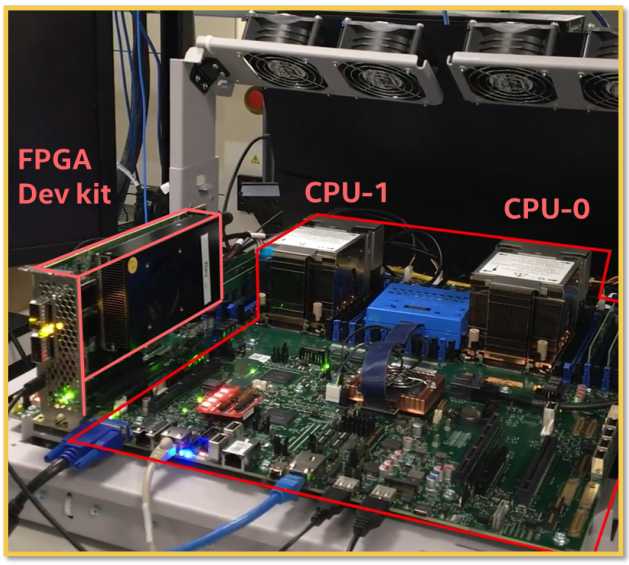


Figure 4: Overview of CXL IP for Intel® Agilex® 7 I-Series FPGA [24], demonstrated in Figure 2 (setup #1).

Integral to our FPGA card is the inclusion of two onboard DDR4 memory modules, each boasting a capacity of 8GB and operating at a clock frequency of 1333 MHz. These modules are accessible from the host system as conventional memory resources. It is imperative to highlight a distinctive attribute of this prototype configuration: the CXL link facilitates access to an identical memory volume. In essence, this means that the same far memory segment can be made available to two distinct NUMA nodes, eliminating any concerns of address overlap. However, due to the absence of a unified cache-coherent domain, the onus of maintaining coherency between the two NUMA nodes assigned to the shared far memory rests with the applications leveraging this configuration.

Notably, the bandwidth attainable from this prototype configuration is subject to current implementation constraints and does not reflect an intrinsic limitation of the CXL standard. Potential avenues for enhancing bandwidth include several considerations. First, transitioning to a higher-speed FPGA, supporting DDR4 speeds of 3200 Mbps or even embracing the capabilities of DDR5 at 5600 Mbps, could appreciably enhance throughput. Additionally, scaling the resources allocated to the CXL IP by increasing the number of slices is a viable strategy. Furthermore, expanding the FPGA's capacity to accommodate multiple independent DDR channels, possibly transitioning from one channel to four, holds promise in augmenting the prototype's bandwidth potential.

In our discussion, the fact that the CXL memory device is DDR4 and not DDR5 is key, as usually, PMem is slower and cheaper than the main memory. By using DDR4 CXL memory and not DDR5, while main memory is DDR5, we keep on this important relation.

3 PERFORMANCE EVALUATION

3.1 STREAM and STREAM-PMem Benchmarks

The STREAM benchmark [42] is a synthetic benchmark program that measures sustainable memory bandwidth for simple vector kernels in high-performance computers. STREAM was developed as a proxy for the basic computational kernels in scientific computations [43] and includes Copy, Scale, Sum, and Triad kernels. STREAM has a dedicated version to benchmark PMem modules by allocating and accessing PMem via PMDK (STREAM-PMem [12]).

The excerpt presented in Listing 1 constitutes a portion of the initial codebase that has since been extracted from the current version of the code.

Listing 1: Original STREAM benchmark code at line 175-181.

```
#ifndef STREAM_TYPE
#define STREAM_TYPE double
#endif
static STREAM_TYPE a[STREAM_ARRAY_SIZE+OFFSET],
                   b[STREAM_ARRAY_SIZE+OFFSET],
                   c[STREAM_ARRAY_SIZE+OFFSET];
```

The content represented in Listing 1 has been substituted in STREAM-PMem [12] with the code demonstrated in Listing 2. The code commences by accessing the memory pool. Furthermore, a function named *initiate* is employed to initialize the three arrays. Following this initialization, the code proceeds to execute the remaining segments of the STREAM benchmark code, mirroring the structure of the original STREAM benchmark code.

Listing 2: Code that has replaced original code.

```
PMEMobjpool *pop;
POBJ_LAYOUT_BEGIN(array);
POBJ_LAYOUT_TOID(array, double);
POBJ_LAYOUT_END(array); //Declearing the arrays
TOID(double) a, b, c;
void initiate() { //Initiating the arrays.
  POBJ_ALLOC(pop, &a, double,
       (STREAM_ARRAY_SIZE+OFFSET)*sizeof(STREAM_TYPE),
       NULL, NULL); //Same for b and c.
int main(){
```

```
    const char path[] = ".../pool.obj";
    pop = pmemobj_create(path, LAYOUT_NAME, 10737418240,
          0666);
    if (pop == NULL)
       pop = pmemobj_open(path, LAYOUT_NAME);
    if (pop == NULL) {
       perror(path);
       exit(1); }
    initiate();
    //The rest of the STREAM benchmark after this.
 }
```

In this work, we employ STREAM in those two versions to showcase the shift from PMem to CXL. Throughout this demonstration, we illustrate how programs designed for PMem can seamlessly operate on CXL-enabled devices. Furthermore, we provide performance assessments to anticipate the impact of CXL on performance in relation to local RAM (DDR4 and DDR5) and local PMem-like devices (emulation of remote sockets either for memory expansion or as a direct access device, as done in [6, 13]).

In contrast to previous research that primarily emphasizes demonstrating the use of CXL memory for in-memory database queries or file system operations [1, 34], STREAM memory access involves accessing and manipulating large arrays, making it particularly applicable and significant for scientific computations in HPC systems. Moreover, STREAM is implemented with OpenMP threads, which is the common shared-memory paradigm in scientific computing for parallelism [9].

3.2 Test Configurations

The methodology of this work is to employ STREAM and STREAM-PMem in various CPU and memory configurations, taking into account the availability of DRAM and CXL memory available on the HPC setups, as will be described next. The results presented in Figure 5, Figure 6, Figure 7, Figure 8 refer to STREAM executions with 100M array elements for Scale, Add, Copy, and Triad operations correspondingly. For each STREAM method, the results of our tests are presented in 2 classes (and a total of 5 groups), divided conceptually for unique comparisons. We sub-divide those 5 groups into two classes. The first class (Class 1, (a)-(c)) refers to the equivalent of the *App-Direct* mode in PMem in which we directly access the local or remote memory (either in the alternative socket or in the CXL memory), and the second class (Class 2, (a)-(b)) refers to the *Memory Mode* in PMem, in which we increase the available memory using other CC-NUMA nodes:

Class 1 — *App-Direct*:

(a) **Local memory access as PMem:** Configurations within this group involve accessing local memory (on-socket memory) in *App-Direct* mode (thus benchmarking STREAM-PMem).

(b) **Remote memory access as PMem:** Configurations within this group involve computing cores on a single socket that access remote memory in *App-Direct* mode (thus benchmarking STREAM-PMem). The term "remote memory" in this context encompasses both CXL-attached memory and on-node memory accessed from the alternative CPU socket (i.e., memory accessed through the UPI).

(c) **Remote memory as PMem (thread affinity):** Configurations within this group involve computing cores in both sockets that access remote memory in *App-Direct* mode (thus benchmarking STREAM-PMem) using two distinct thread affinity methods: *close* and *spread*. The *close* method populates an entire socket first and then adds cores from the second socket. The *spread* method, on the opposite, adds cores alternately from both sockets.

Class 2 — *Memory Mode*:

(a) **Remote CC-NUMA:** Configurations within this group involve computing cores on a single socket that access remote memory as CC-NUMA.

(b) **Remote CC-NUMA (all cores):** Configurations within this group involve cores on both CPU sockets accessing remote memory as CC-NUMA. This includes configurations where both sockets operate and access memory on one of them since these workloads include remote accesses.

For better clarity, the data flow for each test configuration is demonstrated in Figure 9. Each row in Figure 9 contains the data flow examinations of the test groups of the two classes. Thus, in each of our test groups, for each of the STREAM operations (Figures 5, 6, 7, 8), the way to understand each trend, and its correspondence to the relevant dataflow, is given in the trend itself by a combination of three: symbol, color and memory annotation. The symbol is used to distinguish between accessing on-node DDR4 (▲), on-node DDR5 (●) or CXL-attached DDR4 (×). The color implies the active compute cores —- either in socket0, socket1, or both. The annotations *pmem*#{0, 1, 2} or *numa*#{0, 1, 2} accompanying each trend provide an explanation of the accesses memory location: 0 for socket0; 1 for socket1; and 2 for CXL memory. *numa* signifies STREAM accessing memory as NUMA memory expansion, while *pmem* represents STREAM-PMem accessing memory using PMDK.

4 RESULTS AND ANALYSIS

Figure 5, Figure 6, Figure 7 and Figure 8 present STREAM results for the Scale, Add, Copy, and Triad operations correspondingly, and for the test configurations defined in subsection 3.2 as will be described next. Figure 5a, Figure 6a, Figure 7a and Figure 8a through Figure 5e, Figure 6e, Figure 7e and Figure 8e present results for Class 1.(a) group though Class 2.(b) group correspondingly.

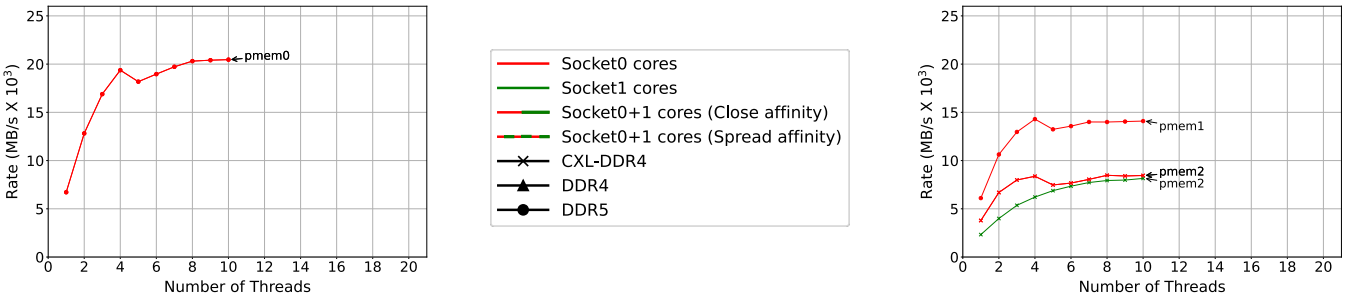
The results explain the costs associated with memory access across varied configurations distinguished by parameters such as memory type (on-node or CXL-attached), memory placement (local to the socket, on the alternate CPU socket, or the CXL-attached memory), access mode (*App-Direct* vs. *Memory Mode*), and thread affinity (Close or Spread).

Next, we will examine and analyze the achieved results in relation to the configuration classes and groups presented in subsection 3.2:

Class 1 — *App-Direct*:

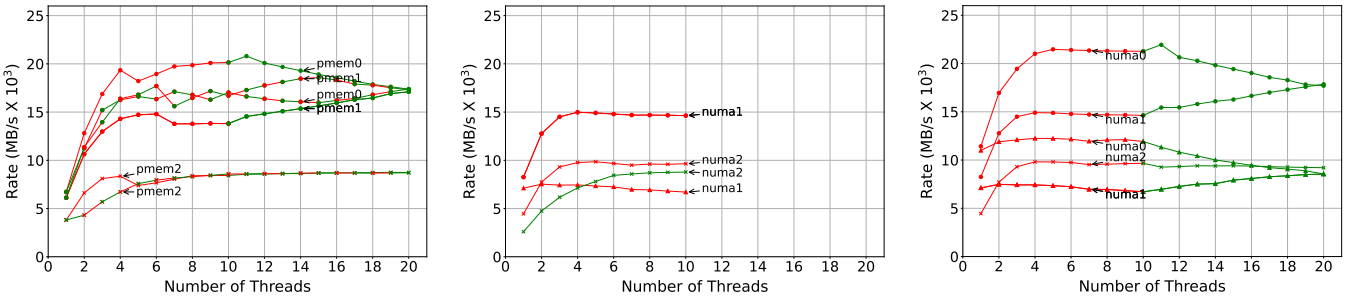
(a) **Local memory access as PMem:** It is possible to observe that among all of the STREAM actions, the *App-Direct* access using PMDK to the local DDR5 memory is saturated around 20-22 GB/s. This test is a reference for the remote access presented in the following group, either to a nearby remote socket or to the CXL memory (with PMDK).

(b) **Remote memory access as PMem:** *App-Direct* access to the emulated remote PMem (DDR5 on the alternate socket) results in a decrease of 30% (~15 GB/s) of performance on average for



(a) Class 1.a: Local memory access as PMem

(b) Class 1.b: Remote memory access as PMem

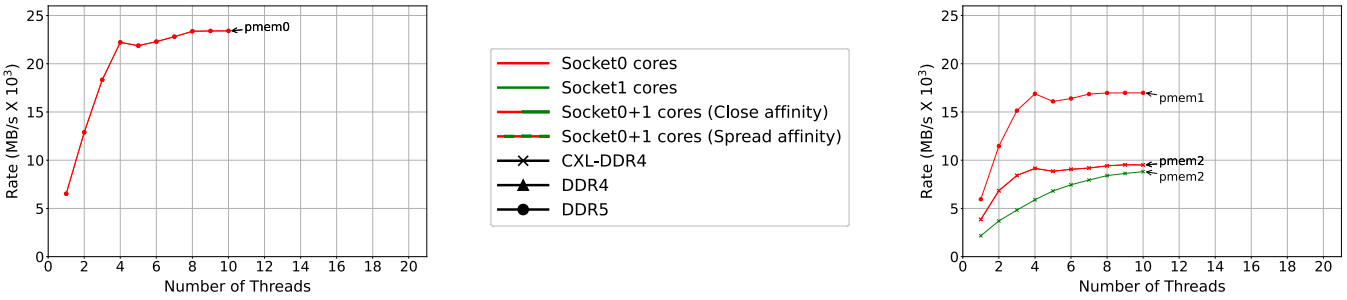


(c) Class 1.c: Remote memory as PMem (thread affinity)

(d) Class 2.a: Remote CC-NUMA

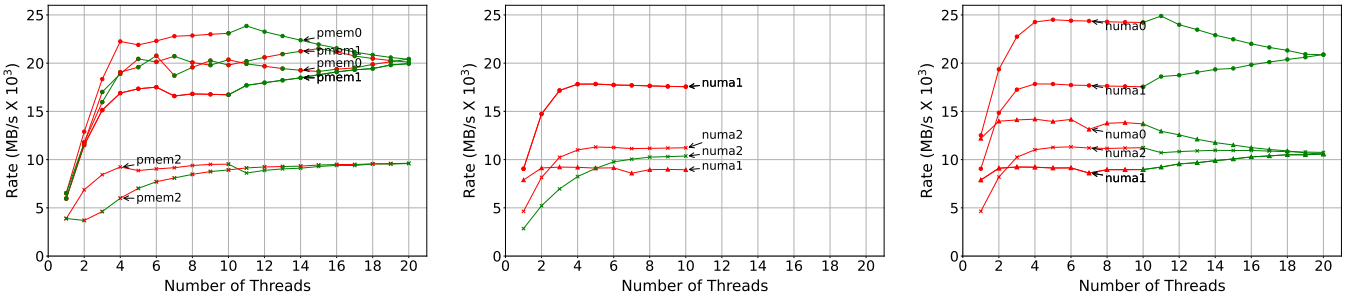
(e) Class 2.b: Remote CC-NUMA (all cores)

Figure 5: SCALE — Various STREAM test configurations. Refer to Section 3.2 for definition of test groups 1.(a), 1.(b), 1.(c), 2.(a), 2.(b) and legend clarifications.



(a) Class 1.a: Local memory access as PMem

(b) Class 1.b: Remote memory access as PMem

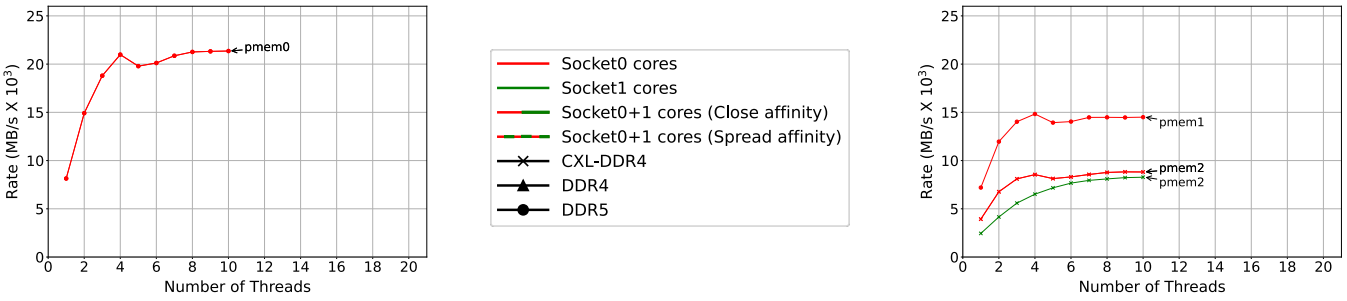


(c) Class 1.c: Remote memory as PMem (thread affinity)

(d) Class 2.a: Remote CC-NUMA

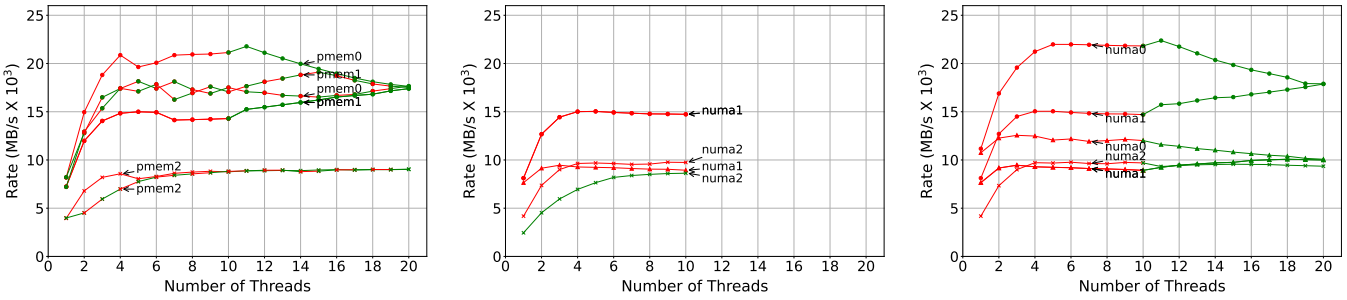
(e) Class 2.b: Remote CC-NUMA (all cores)

Figure 6: ADD — Various STREAM test configurations. Refer to Section 3.2 for definition of test groups 1.(a), 1.(b), 1.(c), 2.(a), 2.(b) and legend clarifications.



(a) Class 1.a: Local memory access as PMem

(b) Class 1.b: Remote memory access as PMem

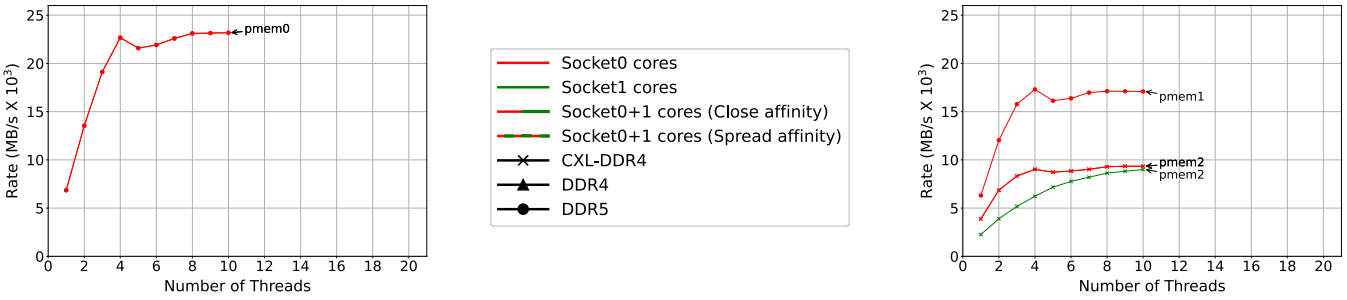


(c) Class 1.c: Remote memory as PMem (thread affinity)

(d) Class 2.a: Remote CC-NUMA

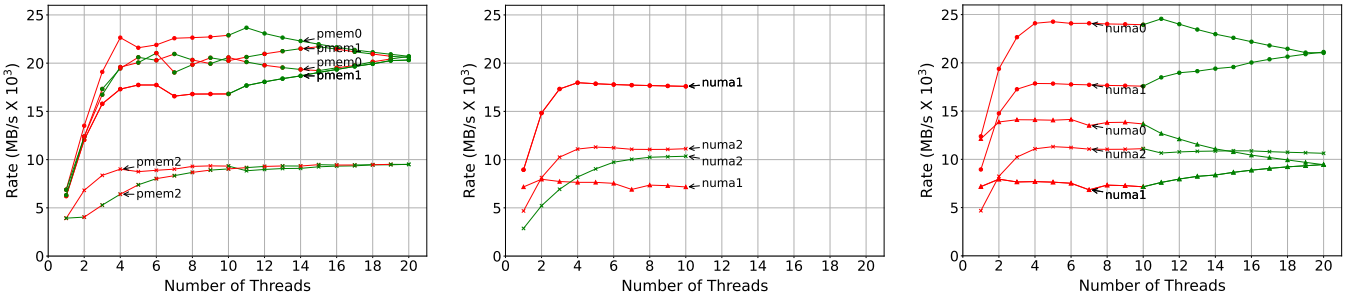
(e) Class 2.b: Remote CC-NUMA (all cores)

Figure 7: COPY — Various STREAM test configurations. Refer to Section 3.2 for definition of test groups 1.(a), 1.(b), 1.(c), 2.(a), 2.(b) and legend clarifications.



(a) Class 1.a: Local memory access as PMem

(b) Class 1.b: Remote memory access as PMem



(c) Class 1.c: Remote memory as PMem (thread affinity)

(d) Class 2.a: Remote CC-NUMA

(e) Class 2.b: Remote CC-NUMA (all cores)

Figure 8: TRIAD — Various STREAM test configurations. Refer to Section 3.2 for definition of test groups 1.(a), 1.(b), 1.(c), 2.(a), 2.(b) and legend clarifications.

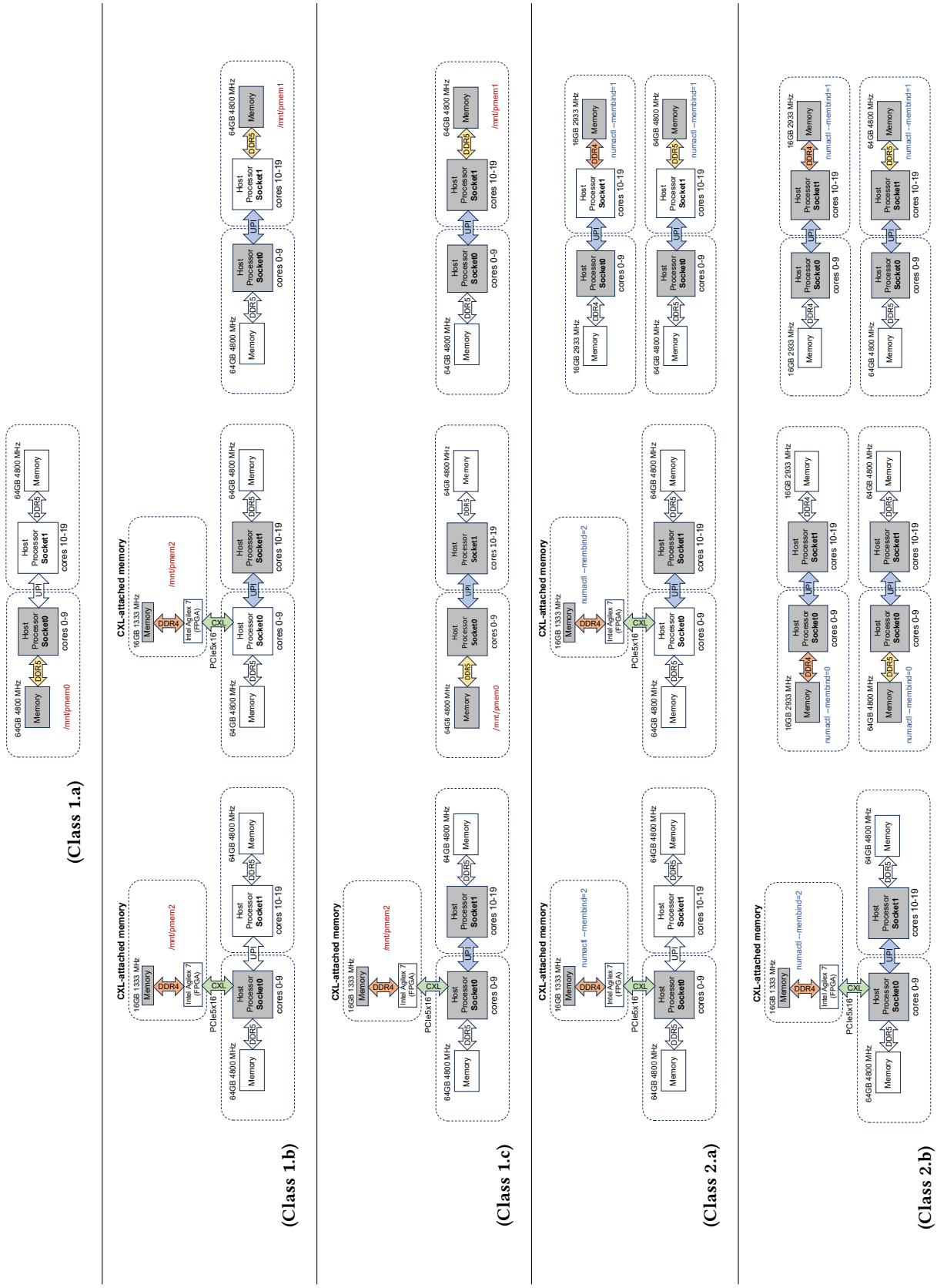


Figure 9: Data flow demonstrations for the two classes (*App Direct* and *Memory Mode*). Each test group is evaluated in corresponding to subfigures of Figure 5, Figure 6, Figure 7, Figure 8. Each row corresponds to a test group.

all STREAM operations, in comparison to local *App-Direct* access. In the case of *App-Direct* access to remote CXL memory (DDR4), we experience 50% decrease in performance in comparison to the emulated PMem on DDR5. However, we note that DDR5 inherently has about 50% higher bandwidth than DDR4, meaning that the rest of the overhead – about 2-3 GB/s loss in bandwidth – can be attributed to the CXL fabric.

(c) **Remote memory as PMem (thread affinity):** As observed in previous groups, local *App-Direct* accesses result in higher bandwidth than remote accesses. In the case of *close* thread affinity, after populating the entire socket, adding remote accesses of compute cores to the workload negatively impacts the bandwidth, whereas adding local accesses contributes positively. With *spread* affinity, the performance demonstrates an average between local and remote accesses due to the inclusion of alternating accesses. Eventually, when both sockets are operating with the entire core count, the results converge for on-node DDR5 and remote CXL memory, separately. Notably, accessing remote CXL memory (DDR4) leads to a 50% observed degradation compared to on-node DDR5.

Class 2 — *Memory Mode*:

(a) **Remote CC-NUMA:** Evaluating DDR4 CC-NUMA, whether on the remote socket or CXL-attached memory, yields comparable figures (with average gaps of up to 2-5 GB/s). However, following a small number of threads, a slight advantage is observed for accessing CXL memory. This advantage can be attributed to the larger caches in Setup #1 utilizing CXL (Shappire Rapids), as opposed to Setup #2 (Xeon Gold) with on-node DDR4 (subsection 2.1). This indicates that the CXL fabric overhead is constrained by the performance reduction when transitioning back from Sapphire Rapids to Xeon Gold. Moreover, the gap between the *CC-NUMA* to DDR5 and DDR4 (on-node or CXL-attached) stands on a factor of two, as already observed in 1.(b) and 1.(c). In addition, in comparison to the results of the *App-Direct* tests in 1.(b), it is observed that PMDK overheads over CC-NUMA are 10%-15% (in all STREAM methods).

(b) **Remote CC-NUMA (all cores):** The observed gap between DDR4 and DDR5 repeats here. Moreover, accessing on-node DDR4 using all cores converges to the same results as accessing DDR4 CXL memory.

To conclude, the analysis reveals that direct access to local DDR5 memory using PMDK saturates at 20-22 GB/s, while direct remote access to emulated PMem and CXL memory results in 30% and 50% performance decreases, respectively, with about 2-3 GB/s bandwidth loss attributed to CXL fabric. In terms of memory expansion, accessing remote DDR4 CC-NUMA and DDR4 CXL-attached memory exhibit similar performance gaps of 2-3 GB/s, while DDR5 CC-NUMA maintains an advantage gap of a factor of 1.5 compared to DDR4, and on-node DDR4 access converges with off-node DDR4 access under varying thread affinities.

5 CONCLUSIONS

In this study, we embarked on a comprehensive exploration of the potential of CXL memory as a promising candidate for serving as a persistent memory solution in the context of disaggregated HPC systems. By conducting physical experiments on state-of-the-art multi-NUMA nodes equipped with high-performance processors and CXL-attached memory prototypes, we have provided empirical evidence that supports the feasibility of using CXL memory to exhibit all the characteristics of persistent memory modules while achieving impressive performance metrics.

Our findings demonstrate that CXL memory has the capability to outperform previously published benchmarks for Optane DCPMM in terms of bandwidth. Specifically, by employing a CXL-DDR4 memory module, which is a cost-effective alternative to DDR5 memory, we achieved bandwidth results comparable to local DDR4 memory configurations, with only a marginal decrease of around 50% when compared to local DDR5 memory configurations. These results, attained across various memory distances from the working threads, were assessed through the well-established STREAM benchmark underscoring the reliability and versatility of CXL memory in the HPC landscape.

The shift from PMem to CXL was not only demonstrated through performance evaluations but was also highlighted through the modification of the STREAM application into STREAM-PMem. We showcased the seamless transition of programming models from PMem to CXL, leveraging the PMDK's *pmemobj* to ensure transactional integrity and consistency of operations on persistent objects. Furthermore, the ability to access CXL memory directly and transactionally, akin to Optane DCPMM, was underscored as a key advantage for practical implementation.

Our study extends beyond theoretical considerations by implementing a practical CXL prototype on an FPGA card. This prototype embodies CXL 1.1/2.0 compliant endpoint designs, demonstrating effective link establishment and transaction layer management through a combination of Soft and Hard IP components. The prototype's performance, while constrained by current implementation limitations, stands as a testament to the extensibility of this solution and offers a blueprint for potential enhancements, including higher-speed FPGAs and increased resources.

6 FUTURE WORK

While this study provides valuable insights into the feasibility and potential benefits of using CXL-enabled memory in HPC systems, several avenues for future research and exploration remain:

- **Scalability and Performance Optimization**: Further investigation is warranted to explore the scalability of CXL-enabled memory in larger HPC clusters, with more than one node accessing the CXL memory. Optimizing communication protocols and memory access patterns can help maximize memory disaggregation benefits.
- **Hybrid Architectures**: Combining different memory technologies, such as DDR, PMem, and CXL memory, in a hybrid memory architecture could offer a balanced solution that leverages the strengths of each technology. Also, the CXL memory could also use DDR5 and even Optane DCPMM, and as such, revisiting the results with those CXL memories would be beneficial.
- **Real-World Applications**: Extending the evaluation to real-world HPC applications beyond benchmarks can provide a clearer understanding of how CXL memory performs in practical scenarios.
- **Fault Tolerance and Reliability**: Investigating fault tolerance mechanisms and data reliability in the context of CXL-enabled memory is crucial, especially in large-scale distributed environments. Specifically, code systems that have previously been built upon PMDK and Optane DCPMM presence in the HPC system.

ACKNOWLEDGMENTS

This work was supported by Pazy grant 226/20, the Lynn and William Frankel Center for Computer Science, and Intel Corporation (oneAPI Center of Excellence program). Computational support was provided by the NegevHPC project [54] and Intel Developer Cloud [23]. The authors would like to thank Gabi Dadush, Israel Hen, and Emil Malka for their hardware support on NegevHPC. The authors also want to thank Jay Mahalingam and Guy Tamir of Intel for their great help in forming this collaboration.

REFERENCES

[1] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. 2022. Enabling CXL memory expansion for in-memory database management systems. In *Proceedings of the 18th International Workshop on Data Management on New Hardware*. 1–5.

[2] Hasan Al Maruf and Mosharaf Chowdhury. 2023. Memory Disaggregation: Open Challenges in the Era of CXL. In *Workshop on HotTopics in System Infrastructure*, Vol. 18.

[3] AsteraLabs. 2022. *CXL Memory Accelerators*. https://www.asteralabs.com/products/cxl-memory-platform/

[4] Alexandro Baldassin, Joao Barreto, Daniel Castro, and Paolo Romano. 2021. Persistent memory: A survey of programming support and implementations. *ACM Computing Surveys (CSUR)* 54, 7 (2021), 1–37.

[5] Lawrence Benson, Marcel Weisgut, and Tilmann Rabl. 2023. What We Can Learn from Persistent Memory for CXL. *BTW 2023* (2023).

[6] Lars Bergstrom. 2011. Measuring NUMA effects with the STREAM benchmark. *arXiv preprint arXiv:1103.3225* (2011).

[7] David E Bernholdt, Swen Boehm, George Bosilca, Manjunath Gorentla Venkata, Ryan E Grant, Thomas Naughton, Howard P Pritchard, Martin Schulz, and Geoffroy R Vallee. 2020. A survey of MPI usage in the US exascale computing project. *Concurrency and Computation: Practice and Experience* 32, 3 (2020), e4851.

[8] Andrés Bustos, Antonio Juan Rubio-Montero, Roberto Méndez, Sergio Rivera, Francisco González, Xandra Campo, Hernán Asorey, and Rafael Mayo-García. 2023. Response of HPC hardware to neutron radiation at the dawn of exascale. *The Journal of Supercomputing* (2023), 1–22.

[9] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An industry-standard API for shared-memory programming. *Computing in Science & Engineering* 1 (1998), 46–55.

[10] Nan Ding, Pieter Maris, Hai Ah Nam, Taylor Groves, Muaaz Gul Awan, LeAnn Lindsey, Christopher Daley, Oguz Selvitopi, Leonid Oliker, and Nicholas Wright. 2023. Evaluating the Potential of Disaggregated Memory Systems for HPC applications. *arXiv preprint arXiv:2306.04014* (2023).

[11] Thomas M Evans, Andrew Siegel, Erik W Draeger, Jack Deslippe, Marianne M Francois, Timothy C Germann, William E Hart, and Daniel F Martin. 2022. A survey of software implementations used by application codes in the Exascale Computing Project. *The International Journal of High Performance Computing Applications* 36, 1 (2022), 5–12.

[12] Svein Gunnar Fagerheim. 2021. *Benchmarking Persistent Memory with Respect to Performance and Programmability*. Master's thesis.

[13] Clément Foyer, Brice Goglin, and Andrès Rubio Proaño. 2023. A survey of software techniques to emulate heterogeneous memory systems in high-performance computing. *Parallel Comput.* (2023), 103023.

[14] Yehonatan Fridman, Yaniv Snir, Harel Levin, Danny Hendler, Hagit Attiya, and Gal Oren. 2022. Recovery of Distributed Iterative Solvers for Linear Systems Using Non-Volatile RAM. In *2022 IEEE/ACM 12th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. IEEE, 11–23.

[15] Yehonatan Fridman, Yaniv Snir, Matan Rusanovsky, Kfir Zvi, Harel Levin, Danny Hendler, Hagit Attiya, and Gal Oren. 2021. Assessing the use cases of persistent memory in high-performance scientific computing. In *2021 IEEE/ACM 11th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. IEEE, 11–20.

[16] 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? *BTW 2023* (2023).

[17] William Gropp. 2012. MPI 3 and beyond: why MPI is successful and what challenges it faces. In *European MPI Users' Group Meeting*. Springer, 1–9.

[18] Shashank Gugnani, Arjun Kashyap, and Xiaoyi Lu. 2020. Understanding the idiosyncrasies of real persistent memory. *Proceedings of the VLDB Endowment* 14, 4 (2020), 626–639.

[19] Frank T Hady, Annie Foong, Bryan Veal, and Dan Williams. 2017. Platform storage performance with 3D XPoint technology. *Proc. IEEE* 105, 9 (2017), 1822–1833.

[20] Jim Handy and Tom Coughlin. 2023. Optane's Dead: Now What? *Computer* 56, 3 (2023), 125–130.

[21] Takahiro Hirofuchi and Ryousei Takano. 2020. A prompt report on the performance of intel optane dc persistent memory module. *IEICE TRANSACTIONS on Information and Systems* 103, 5 (2020), 1168–1172.

[22] Intel. 2022. Migration from Direct-Attached Intel Optane Persistent Memory to CXL-Attached Memory. https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-11/optane-pmem-to-cxl-tech-brief.pdf. [Online].

[23] Intel. 2023. Intel Developer Cloud. https://www.intel.com/content/www/us/en/developer/tools/devcloud/overview.html. [Online].

[24] Intel. 2023. Intel® FPGA Compute Express Link (CXL) IP. https://www.intel.com/content/www/us/en/products/details/fpga/intellectual-property/interface-protocols/cxl-ip.html. [Online].

[25] Intel. 2023. Intel® FPGA Compute Express Link (CXL) IP. https://www.intel.com/content/www/us/en/products/details/fpga/intellectual-property/interface-protocols/cxl-ip.html. [Online].

[26] Joseph Izraelevitz, Hammurabi Mendes, and Michael L Scott. 2016. Linearizability of persistent memory objects under a full-system-crash failure model. In *International Symposium on Distributed Computing*. Springer, 313–327.

[27] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, et al. 2019. Basic performance measurements of the intel optane DC persistent memory module. *arXiv preprint arXiv:1903.05714* (2019).

[28] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. Hbm (high bandwidth memory) dram technology and architecture. In *2017 IEEE International Memory Workshop (IMW)*. IEEE, 1–4.

[29] Hongshin Jun, Sangkyun Nam, Hanho Jin, Jong-Chern Lee, Yong Jae Park, and Jae Jin Lee. 2016. High-bandwidth memory (HBM) test challenges and solutions. *IEEE Design & Test* 34, 1 (2016), 16–25.

[30] Rajat Kateja, Anirudh Badam, Sriram Govindan, Bikash Sharma, and Greg Ganger. 2017. Viyojit: Decoupling battery and DRAM capacities for battery-backed DRAM. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 613–626.

[31] Awais Khan, Hyogi Sim, Sudharshan S Vazhkudai, Jinsuk Ma, Myeong-Hoon Oh, and Youngjae Kim. 2020. Persistent memory object storage and indexing for scientific computing. In *2020 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 1–9.

[32] Peter M Kogge and William J Dally. 2022. Frontier vs the Exascale Report: Why so long? and Are We Really There Yet?. In *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 26–35.

[33] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2010. Phase change memory architecture and the quest for scalability. *Commun. ACM* 53, 7 (2010), 99–106.

[34] Donghun Lee, Thomas Willhalm, Minseon Ahn, Suprasad Mutalik Desai, Daniel Booss, Navneet Singh, Daniel Ritter, Jungmin Kim, and Oliver Rebholz. 2023. Elastic Use of Far Memory for In-Memory Database Management Systems. In *Proceedings of the 19th International Workshop on Data Management on New Hardware*. 35–43.

[35] Jiuxing Liu, Jiesheng Wu, Sushmitha P Kini, Pete Wyckoff, and Dhabaleswar K Panda. 2003. High performance RDMA-based MPI implementation over InfiniBand. In *Proceedings of the 17th annual international conference on Supercomputing*. 295–304.

[36] Ming Liu. 2023. Fabric-Centric Computing. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*. 118–126.

[37] Glenn K Lockwood, Damian Hazen, Quincey Koziol, R Shane Canon, Katie Antypas, Jan Balewski, Nicholas Balthaser, Wahid Bhimji, James Botts, Jeff Broughton, et al. 2023. Storage 2020: A vision for the future of hpc storage. (2023).

[38] Luke Logan, Jay Lofstead, Xian-He Sun, and Anthony Kougkas. 2023. An Evaluation of DAOS for Simulation and Deep Learning HPC Workloads. In *Proceedings of the 3rd Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*. 9–16.

[39] Krishna T Malladi, Manu Awasthi, and Hongzhong Zheng. 2016. DRAMPersist: Making DRAM Systems Persistent. In *Proceedings of the Second International Symposium on Memory Systems*. 94–95.

[40] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.

[41] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.

[42] John D. McCalpin. 1995. STREAM Benchmark. https://www.cs.virginia.edu/stream/. [Online].

[43] John D McCalpin. 1995. Stream benchmark. *Link: www. cs. virginia. edu/stream/ref. html# what* 22, 7 (1995).

[44] John D. McCalpin. 2023. Bandwidth Limits in the Intel Xeon Max (Sapphire Rapids with HBM) Processors. In *ISC 2023 IXPUG Workshop*. 1–24.

[45] Sally A McKee. 2004. Reflections on the memory wall. In *Proceedings of the 1st conference on Computing frontiers*. 162.

[46] George Michelogiannakis, Yehia Arafa, Brandon Cook, Liang Yuan Dai, Abdel Hameed Badawy, Madeleine Glick, Yuyang Wang, Keren Bergman, and John Shalf. 2023. Efficient Intra-Rack Resource Disaggregation for HPC Using Co-Packaged DWDM Photonics. *arXiv preprint arXiv:2301.03592* (2023).

[47] George Michelogiannakis, Benjamin Klenk, Brandon Cook, Min Yee Teh, Madeleine Glick, Larry Dennison, Keren Bergman, and John Shalf. 2022. A case for intra-rack resource disaggregation in HPC. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 2 (2022), 1–26.

[48] Vladimir Mironov, Igor Chernykh, Igor Kulikov, Alexander Moskovsky, Evgeny Epifanovsky, and Andrey Kudryavtsev. 2019. Performance evaluation of the intel optane dc memory with scientific benchmarks. In *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 1–6.

[49] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *2013 5th IEEE International Memory Workshop*. IEEE, 21–25.

[50] Dushyanth Narayanan and Orion Hodson. 2012. Whole-system persistence. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*. 401–410.

[51] Kevin Huang Nathan Pham. 2019. Analyzing the Performance of Intel Optane Persistent Memory 200 Series in Memory Mode with Lenovo ThinkSystem Servers.

[52] Samsung Newsroom. 2022. *Samsung Electronics Introduces Industry's First 512GB CXL Memory Module*. https://news.samsung.com/global/samsung-electronics-introduces-industrys-first-512gb-cxl-memory-module

[53] SK Hynix Newsroom. 2022. *SK hynix Develops DDR5 DRAM CXLTM Memory to Expand the CXL Memory Ecosystem*. https://news.skhynix.com/sk-hynix-develops-ddr5-dram-cxltm-memory-to-expand-the-cxl-memory-ecosystem/

[54] Rotem Industrial Park. 2019. NegevHPC Project. https://www.negevhpc.com. [Online].

[55] Onkar Patil, Latchesar Ionkov, Jason Lee, Frank Mueller, and Michael Lang. 2019. Performance characterization of a dram-nvm hybrid memory architecture for hpc applications using intel optane dc persistent memory modules. In *Proceedings of the International Symposium on Memory Systems*. 288–303.

[56] Ivy Peng, Ian Karlin, Maya Gokhale, Kathleen Shoga, Matthew Legendre, and Todd Gamblin. 2021. A holistic view of memory utilization on HPC systems: Current and future trends. In *The International Symposium on Memory Systems*. 1–11.

[57] Ivy Peng, Roger Pearce, and Maya Gokhale. 2020. On the memory underutilization: Exploring disaggregated memory on hpc systems. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 183–190.

[58] Milan Radulovic, Darko Zivanovic, Daniel Ruiz, Bronis R de Supinski, Sally A McKee, Petar Radojković, and Eduard Ayguadé. 2015. Another trip to the wall: How much will stacked dram benefit hpc?. In *Proceedings of the 2015 International Symposium on Memory Systems*. 31–36.

[59] Sadhana Rai and Basavaraj Talawar. 2023. Nonvolatile Memory Technologies: Characteristics, Deployment, and Research Challenges. *Frontiers of Quality Electronic Design (QED) AI, IoT and Hardware Security* (2023), 137–173.

[60] Daniel Reed, Dennis Gannon, and Jack Dongarra. 2022. Reinventing high performance computing: challenges and opportunities. *arXiv preprint arXiv:2203.02544* (2022).

[61] Chaoyi Ruan, Yingqiang Zhang, Chao Bi, Xiaosong Ma, Hao Chen, Feifei Li, Xinjun Yang, Cheng Li, Ashraf Aboulnaga, and Yinlong Xu. 2023. Persistent Memory Disaggregation for Cloud-Native Relational Databases. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 498–512.

[62] Andy Rudoff. 2017. Persistent memory: The value to hpc and the challenges. In *Proceedings of the Workshop on memory centric programming for hpc*. 7–10.

[63] Arthur Sainio et al. 2016. NVDIMM: changes are here so what's next. *Memory Computing Summit* (2016).

[64] Steve Scargall. 2020. Programming Persistent Memory: A Comprehensive Guide for Developers. (2020).

[65] Steve Scargall and Steve Scargall. 2020. libpmemobj: A Native Transactional Object Store. *Programming Persistent Memory: A Comprehensive Guide for Developers* (2020), 81–109.

[66] Debendra Das Sharma, Robert Blankenship, and Daniel S Berger. 2023. An Introduction to the Compute Express Link (CXL) Interconnect. *arXiv preprint arXiv:2306.11227* (2023).

[67] Galen M Shipman, Sriram Swaminarayan, Gary Grider, Jim Lujan, and R Joseph Zerr. 2022. Early Performance Results on 4th Gen Intel (R) Xeon (R) Scalable Processors with DDR and Intel (R) Xeon (R) processors, codenamed Sapphire Rapids with HBM. *arXiv preprint arXiv:2211.05712* (2022).

[68] Montage Technology. 2022. *Montage Technology Delivers the World's First CXL™ Memory eXpander Controller*. https://www.montage-tech.com/Press_Releases/20220506

[69] TB Tristian and L Travis. 2019. Analyzing the performance of Intel Optane DC persistent memory in app direct mode in Lenovo ThinkSystem servers.

[70] 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.

[71] Ying Wang, Wen-Qing Jia, De-Jun Jiang, and Jin Xiong. 2023. A Survey of Non-Volatile Main Memory File Systems. *Journal of Computer Science and Technology* 38, 2 (2023), 348–372.

[72] Michèle Weiland, Holger Brunst, Tiago Quintino, Nick Johnson, Olivier Iffrig, Simon Smart, Christian Herold, Antonino Bonanni, Adrian Jackson, and Mark Parsons. 2019. An early evaluation of intel's optane dc persistent memory module and its impact on high-performance scientific applications. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 1–19.