SDM: Sharing-enabled Disaggregated Memory System with Cache Coherent Compute Express Link

Hyokeun Lee , Kwanseok Choi[†], Hyuk-Jae Lee[†§], Jaewoong Sim[†]
North Carolina State University
[†]*Department of Electrical and Computer Engineering, Seoul National University*
[§]*Inter-university Semiconductor Research Center, Seoul National University*
hlee48@ncsu.edu, [†]*{kwanseok.choi, hyukjae, jaewoong}@snu.ac.kr*

***Abstract*—Disaggregated memory has been gaining significant traction as a promising solution for scaling memory capacity and better utilizing memory resources in data centers. However, a disaggregated memory system that can simultaneously achieve high performance and user transparency is still not available. Although some modern interconnect technologies now feature hardware coherence protocols that can potentially enable data sharing among multiple computing nodes in a *user-transparent* manner, naively applying these technologies to disaggregated memory systems results in non-negligible performance overheads.**

In this work, we propose SDM, a sharing-enabled, cache-coherent disaggregated memory system that effectively utilizes modern interconnect technology. The key design principle of SDM is to implement a novel, *dedicated* control flow that efficiently enables data sharing among multiple computing nodes without the need to modify user applications, by leveraging the message types defined in the modern memory expansion standard, Compute Express Link (CXL). We also introduce resource management and speculative memory access mechanisms that do not interfere with normal memory transaction channels, thereby further improving the performance of disaggregated memory systems. We evaluate our design based on an in-house simulation framework with detailed analytical models that mimic a cache-coherent *multi-node* disaggregated memory system. The results show that SDM outperforms the optimized baseline system, which is similar to the one employing CXL 3.0, by 5.77× and 2.65× for two distinctive benchmark suites.

***Index Terms*—Disaggregated memory, CXL, cache coherency**

I. Introduction

Modern high-performance computing applications such as graph processing and in-memory databases require a substantial amount of memory capacity that a single compute node is unable to accommodate. To overcome the so-called "memory capacity wall," the industry has thus far tackled the problem at multiple different levels. At the device level, memory vendors such as Samsung, Micron, and SK Hynix continue to increase the memory capacity, even by vertically stacking memory dies (e.g., HBM [44] and HMC [42]).[1] At the module level, Intel has introduced high-density non-volatile memory (NVM) products in DIMM form factors to expand the memory capacity [28], [51], [52]. At the system level, several distributed memory systems have been proposed by aggregating memory capacity from different server nodes within a shared network [1], [4], [33].

However, these approaches do not fundamentally address the memory scalability issue. Although vertically stacked memory and NVM-based products can increase memory capacity to some extent, they still fall short of meeting the memory requirement of modern applications. Increasing the number of memory modules along different channels also becomes challenging, as the pin counts on general-purpose processors are on the verge of saturation. In addition, building distributed memory systems results in considerable costs for essential components (e.g., processors or storage devices) to prepare tens of server nodes.

Fortunately, a disaggregated memory system, which aggregates a large number of memory modules into a separate *memory node (or remote memory)*, has emerged as a promising and flexible solution to alleviate the memory scalability issue [36], [37]. Unlike a traditional computing node that runs user applications, the memory node does not require the inclusion of additional processors, storage devices, or even operating systems, leading to lower development costs for a server system. Computing nodes and memory nodes are connected to a rack-level switch, on which various interconnect protocols (e.g., RDMA [34], CXL [15], or CCIX [13]) are devised to efficiently transfer commands and data.

Problems. While disaggregated memory is an attractive solution that enables memory pool expansion in a cost effective manner, existing disaggregated memory systems encounter two key issues: remote access overheads and user transparency. Firstly, accessing a remote memory node incurs high latency compared to accessing the local memory, as it involves several data buffer copies across software layers when conventional network-based transactions (e.g., TCP) are employed [23]. Although RDMA-based disaggregated memory systems have been proposed to directly access remote memory regions while minimizing data copies and the involved software layers [5], [10], [17], [21], [34], [46], [50], [53], the use of RDMA still necessitates additional copy operations for queue pair-based transactions at the software level [20]. A second-level

[1]Although HBM is considered as a solid technology for improving memory bandwidth today, it was also anticipated to increase the memory capacity by continuously stacking memory dies using through-silicon vias (TSVs). Unfortunately, manufacturers are encountering several process technology-related challenges as they attempt to stack more dies: resistance-capacitance (RC) delay of TSVs, instant power consumption, and reliability issues [18].

979-8-3503-4254-3/23/$31.00 ©2023 IEEE
DOI 10.1109/PACT58117.2023.00016

address translation is also a crucial factor that contributes to a high remote access overhead. In a disaggregated memory system, since each computing node runs its own operating system, the node-level physical address from each computing node needs to be translated to the system-/global-level physical address through second-level address translation. Such node-to-system address mapping needs to be stored in the memory node, leading to a larger number of remote accesses. Secondly, the lack of user transparency is another issue with disaggregated memory systems. Enabling transparent access to memory nodes for user applications is crucial to the widespread adoption of disaggregated memory systems. Still, RDMA-based disaggregated memory systems require users to transfer data from/to remote memory regions by modifying their applications with APIs that abstract RDMA verbs (e.g., *libibverbs* [30]).

Recently, several prior works have been presented to address the aforementioned problems by caching the remote data or metadata (e.g., address translation information) on a computing node [10], [31], [32], or by customizing the swap fault handler in the operating system for user-transparent remote access [10], [34], [46]. However, these works primarily assume a single computing node in a disaggregated memory system, and they have not considered *data coherence among multiple computing nodes*. Some prior works provide data sharing among different nodes using software synchronization [2], [22], [40]. However, it is worth noting that software synchronization would incur a significantly higher performance overhead compared to hardware-assisted solutions, as shown in the studies of traditional multicore and distributed systems [12], [19]. Consequently, we envision that a cache-coherent interconnect for expanded memory pools, Compute Express Link (CXL) [16], is likely to be a key solution to simultaneously address these problems for disaggregated memory systems.

Challenges. CXL is a cache-coherent, transaction-level protocol that abstracts the I/O path using its own *memory syntax*, irrespective of the underlying device media [29]. With its memory syntax, CXL *transparently* expands the main memory pool with minimal modifications to the operating system. However, applying CXL technologies to a disaggregated memory system poses some new challenges. First, CXL 2.0 supports coherence only between a single computing node (i.e., host) and multiple CXL devices. Although the recently-announced CXL 3.0 supports data coherency among multiple computing nodes, a computing node needs to *exclusively* cache the data at a time by invalidating the same data in other computing nodes, compromising the performance of applications that exhibit high data locality. Second, even if data sharing is enabled with CXL, harmoniously combining it with other caching schemes is challenging, as the state-of-the-art disaggregated memory systems employ an address translation caching scheme that decouples access permission checking from address translation [32]. In particular, checking access permission inevitably necessitates remote access; moreover, accessing data (whether cached or not) must be preceded by permission checking, thereby leading to suboptimal system performance due to the serialization.

Contributions. In this paper, we propose a sharing-enabled disaggregated memory system, namely *SDM*, which effectively allows multiple computing nodes to access the shared data without the need of costly invalidations. SDM employs a novel sharing-enabled control flow (SHA-CF) that abstracts all nodes except for the requester host, wherein the memory node *emulates* snoop transactions to other hosts by leveraging the `CXL.cache` and `CXL.mem` message types defined in the CXL specification without violating its own protocol. Meanwhile, the resource management primitives of memory nodes are also essential to our system, such as allocation, free, and address translation. These primitives should not interfere with normal read and write transaction channels of CXL (i.e., `CXL.mem` and `CXL.cache`). As such, we propose to leverage the `CXL.io` channel for device control to manage the memory node resource. We also propose *speculative access* to harmonically combine the address translation caching and the data caching schemes. The key idea is to overlap data access and permission checking speculatively to increase the request processing throughput, where the connection of the session will be disconnected once the violation is detected. Our evaluation shows that SDM achieves 5.77× and 2.65× higher speedups for computation-intensive and memory-intensive workloads compared to the baseline, which is similar to CXL 3.0 and is assisted by a state-of-the-art caching scheme. In summary, this paper makes the following contributions.

- We propose a disaggregated memory system that enables effective data sharing between multiple computing nodes, which builds on a modern interconnect standard, CXL. The control flow in SDM emulates snoop requests among multiple hosts while complying with well-defined CXL specification. To our knowledge, this is the first work that leverages CXL to achieve cacheline sharing in a multi-node disaggregated memory system.
- We present a simple but effective mechanism for remote memory resource management. Using the reserved message field in the `CXL.io` protocol, we define new message types to allocate, free, and walk page tables without complicating the `CXL.mem` and `CXL.cache` transactions that are dedicated to normal read and write transactions.
- We show the inefficiency of naively applying the data caching and the address translation caching schemes to the CXL-based disaggregated memory system. Consequently, we propose a speculative access method to benefit more from caching schemes.

II. Background

In this section, we first describe the disaggregated memory system assumed in this work. We then provide the background of a modern interconnect technology, Compute Express Link (CXL), which potentially enables user-transparent data sharing for multi-host disaggregated memory systems.

A. Assumptions on Disaggregated Memory

The use case of disaggregated memory systems. Disaggregated memory systems can be used either as a swap space to a local main memory [5], [21], [36], [37] or as an expanded main memory pool [2], [10], [17], [22], [32], [34], [40], [46]. In this work, we assume that the disaggregated memory system is constructed for *main memory pool expansion*, similar to the recent proposals due to its attractiveness for performance and system reliability. Note that although some prior works use the memory node as a swap space to improve the performance of demand paging (or page swapping), it may lead to lower system reliability. In a modern OS, an *optimistic* page allocation mechanism is employed to prevent severe performance degradation due to page swapping that results in context switch and TLB shootdowns. A newly accessed virtual address is mapped to a free page, instead of sparing a physical page for that virtual address using page swapping. Otherwise, the out-of-memory killer may trigger a system failure [41], [43], thereby degrading the system reliability. Consequently, disaggregated memory systems are more likely to become an expanded main memory pool.

Architecture of the disaggregated memory system. Figure 1 shows the overview of a baseline disaggregated memory system, which is divided into three parts: memory node, backplane, and computing node. The memory node consists of several memory modules with simple hardware logic that converts the transaction requests to module-compatible commands, or vice versa. The exemption of processors and storage from a memory node allows us to reduce the cost of the memory node and thus enable a larger memory capacity. The backplane redirects the requests or responses from one node to another. The computing node, which consists of basic hardware components, runs user applications.

A disaggregated memory system generally requires a *second-level* page table walk to translate the node-level address to the system-level address. As shown in Figure 1, the memory management unit (MMU) first translates the virtual address to the *local physical address*, which becomes the memory request address. Once the last-level cache (LLC) miss occurs, the system agent multiplexes memory requests from the LLC to either local memory or the backplane, according to the request address. In the backplane, the second-level page table walk translates the local physical address to the *global physical address*. Then, the memory node data is accessed after checking access permission flags.

A stronger baseline. The additional page table walk incurs significant performance overheads in disaggregated memory systems due to the increased number of remote accesses for the walk. In this work, we assume that the address translation caching scheme such as [32] is applied to our system to reduce the number of remote accesses as a *stronger and more reasonable* baseline. The translation caching scheme decouples translation information (i.e., mapping from the virtual address to the global physical address) and access permission flags. The translation information is cached in the local memory of the computing node, while the permission flag is cached in the backplane because all computing nodes need to check the access permission before accessing the memory node.

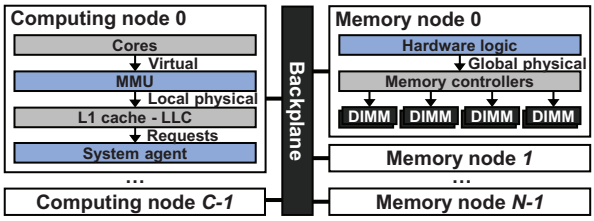


Fig. 1. Overview of a disaggregated memory system.

B. Compute Express Link (CXL)

CXL is a transaction-level multi-protocol specification that is proposed by a consortium organized with various leading industrial companies. Processor/memory vendors have recently announced the CXL-enabled products [6], [27], [45], [47], and some service providers are considering applying the technology to disaggregated memory [35], [39].

The basic concept of CXL is to provide a standard, cache-coherent interface for various hardware devices (e.g., memory or accelerators) based on three independent logical channels at the transaction level: `CXL.mem`, `CXL.cache`, and `CXL.io`. `CXL.mem` transfers simple memory read and write requests; `CXL.cache` transfers caching-related messages (e.g., coherence states) based on the conventional *MESI protocol*; `CXL.io` transfers I/O semantics through PCIe-like packets to control and initialize devices. Note that the coherence states of computing nodes are stored in the dedicated coherence directory, which is implementation-specific. The directory area is accessed by the CXL home agent and coherence bridge (COHB) component, both in the system agent (Section IV). Furthermore, CXL does not consider transient coherence states, as a component is only allowed to have one snoop outstanding to a given cacheline address at a time in CXL. Therefore, the component must wait until it receives the snoop response before issuing the next snoop to that address.

At the device level, three CXL device types are defined based on the aforementioned logical channels: type 1, type 2, and type 3. Type 1 devices are attractive when they are expected to become fully coherent external caching devices or accelerators. Type 1 devices only transmit cache coherence-related messages on `CXL.cache` channel without incorporating `CXL.mem`; hence, device-attached memory is only private to the device. In practice, some domain-specific accelerators (e.g., NPU) that proactively consume host data can be manufactured as type 1 devices.

Type 2 is a proper device type if the device-attached memory is desired to be not only fully coherent but visible to the host. Such a host-visible memory is referred to as host-managed device memory. Data in host-managed device memory is accessed via `CXL.mem`, followed by interactive messages of coherence states and snooping on the

TABLE I
CXL MESSAGE TYPES USED IN THIS PAPER.

Channel	Message	Src.	Description
CXL.mem	MemRd	Host	Memory read request
	MemWr	Host	Memory write request
	MemInv	Host	Invalidation request
	MemRdFwd	Host	Notify the device that that the data can be pull from its memory
	Cmp-⋆	Mem	Completion message
CXL.cache	RdOwn	Mem	Get cacheline ownership
	RdAny	Mem	Read a cacheline
	CLFlush	Mem	Invalidate a cacheline
	GO-⋆	Host	Respond one of the coherence states as an indicator of state transition

CXL.cache channel. Different from type 1 devices, type 2 devices operate as direct data providers with respect to the host. For example, high-cost copy operations between GPU and host (e.g., cudaMalloc) can be avoided with type 2 devices.

Type 3 devices are used to expand the memory pool. Type 3 devices solely include host-managed device memory, which is accessed via the CXL.mem channel. Compared to other device types, type 3 uniquely has a feature similar to the logical volume, namely a multi-logical device; it partitions a memory device into 16 isolated logical regions, spanning secure and isolated regions for different processes. Recently, memory manufacturers have announced memory expansion products based on type 3 [45], [47]. In this paper, memory nodes are assumed as *type 2 devices* to develop a high-performance disaggregated memory system based on the cache-coherent memory pool.

CXL messages used in this work. Table I describes the message types used in this paper. Note that CXL defines more messages than the ones in the table; we only explain the message types essential to the baseline and the proposed control flow. The third column indicates the source of each message type. Coherence states mentioned in the table are M (modified), E (exclusive), S (shared), and I (invalid), which are the same as the MESI protocol. Both RdOwn and RdAny are read request messages from the memory. For Cmp-⋆ and GO-⋆, "⋆" can be overloaded with coherence states (i.e., MESI) that specify the coherence state of the cacheline. As an example, after a device dispatches RdOwn or RdAny to the host, the device will receive a response accompanying coherence states, which would be the coherence state in that device. According to the CXL specification, RdOwn is not allowed to receive GO-S (shared), whereas RdAny is allowed to receive GO-I, GO-S, GO-E, and GO-M as responses.

III. MOTIVATION

In this section, we first set the design goals of our proposed disaggregated memory system, SDM. We then explain the motivational challenges and inefficiencies of naively building on the CXL protocol for disaggregated memory systems.

A. Design Goals

Goal 1: Leveraging as a main memory pool. Some prior works [5], [21], [36] use disaggregated memory nodes as swap spaces to improve the performance of page swapping. Page swap occurs when the currently allocated physical memory is preempted by another logical address. That is, swap spaces are used for storing *allocated page data* in the backup storage. In modern operating systems, physical memory is allocated if free page blocks (i.e., not preempted memory) are available in the memory pool. If free blocks are not available in the system, otherwise, the out-of-memory (OOM) killer may trigger a system failure [41], [43], potentially leading to lower system reliability. Thus, it is necessary to leverage disaggregated memory as the main memory pool rather than the swap space.

Goal 2: Low remote access latency. Accessing memory nodes requires several remote accesses, which potentially incurs large performance overhead. In a disaggregated memory system, the remote accesses originate from two activities: metadata access and data access. The metadata access includes the *address translation* and the *permission check (i.e., referring to ACM, or access control metadata)*. In particular, address translation leads to several remote memory accesses due to the multi-level page table walk in modern architectures [24]. Checking the page permission is always necessary before accessing normal data [32]; hence, lowering the average access latency is also required for a high-performance disaggregated memory system.

Goal 3: Preserving user transparency. Several bleeding-edge technologies, such as NVM, neural processing units, and disaggregated memory, have been announced to improve system performance and scalability. However, these technologies are challenging to be widely adopted in the market due to the lack of user transparency. In most cases, programmers need to manually modify their source code to take advantage of new technologies (e.g., Intel PMDK [25] or tensor processing units [48]). Such human effort is error-prone; hence, user transparency is also a highly desirable feature in a high-performance disaggregated memory system.

Our key observation is that applying the CXL protocol to a disaggregated memory system can achieve the aforementioned design goals. However, naively leveraging CXL brings new challenging issues that need to be addressed with extra effort. In the following subsection, we explain the problem definitions of our work.

B. Problem Statements

Limitation of the naive approach/CXL 3.0. As a naive approach, invalidation-based control flow (*INV-CF*) can be considered as the baseline cache coherence management for a multi-host disaggregated memory system. The main idea of INV-CF is caching the data *exclusively* in a single computing node by *invalidating* the same data in the others. The INV-CF needs to define two representative primitives: *fetch-clean* and *fetch-dirty*. For simplicity, we assume two host nodes (H-1 and H-2) and one memory node (MN) to explain these primitives, as shown in Figure 2. The *fetch-clean* primitive occurs when a host requests the data that is cached by at most one host. In Figure 2 (a), for instance, H-2 first reads data by sending the MemRd message to the memory node. Once the

memory node receives the message, it sends RdOwn to other hosts (i.e., H-1 in this example) to obtain the ownership of the requested data. If the requested data has been cached in H-1, H-1 invalidates the cacheline and sends MemRdFwd to the memory node. Finally, H-2 obtains the data along with Cmp-E, which updates the coherent state as E-state (exclusive).

The *fetch-dirty* primitive is also needed when a host requests the data that is cached and modified by another host. In Figure 2 (b), H-2 tries to fetch the modified data that is cached in H-1. Using RdOwn, the memory node also tries to obtain the ownership of the requested data. Different from the previous scenario, H-1 issues MemWr to forward the up-to-date data to the memory node. Once the memory node completes writing the data from H-1, it signals Cmp, which is the predefined response of MemWr, to H-1. Finally, H-1 issues MemRdFwd to the memory node after invalidating the cacheline, similar to the case of the *fetch-clean* primitive. As shown in the shaded parts in the figure, *fetch-clean* incurs two additional transactions, whereas *fetch-dirty* needs four transactions for data invalidation.

Applying INV-CF to a four-node disaggregated memory system (see Section V), we measure *shared cacheline and access ratios* for computing-intensive (PARSEC) and memory-intensive workloads (Intel GAP), as shown in Figure 3. In general, most workloads yield high ratios, whereas *raytrace* is low due to its stream access pattern. The shared cacheline ratio is the ratio of the number of accessed cachelines (from more than one host) to the number of all accessed cachelines; the shared access ratio denotes the ratio of the number of accesses on shared cachelines to the total number of accesses. The shared cacheline ratio is the potential metric that the system performance can be improved by *allowing* data sharing in multiple computing nodes. Furthermore, the higher shared access ratio may yield higher performance even for the workloads with low shared cacheline ratios, because a low shared cacheline ratio may result from a large denominator.

Note that INV-CF is similar to the coherence management of the most recent CXL 3.0. In CXL 3.0, back-invalidation channels (i.e., BISnp from device to host and BIRsp from host to device) are newly introduced to manage the coherence of multiple computing nodes. Compared to the INV-CF operations in Figure 2 (b), for instance, a computing node H-2

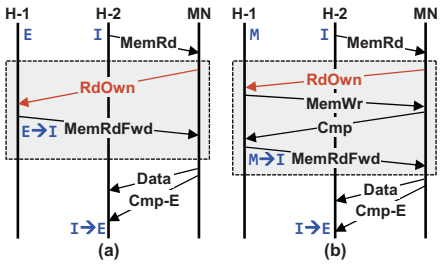


Fig. 2. Two representative primitives of INV-CF: (a) fetch-clean, (b) fetch-dirty. The red arrows indicate CXL.cache messages; the black arrows indicate CXL.mem messages; blue indicates transitions of coherence states.

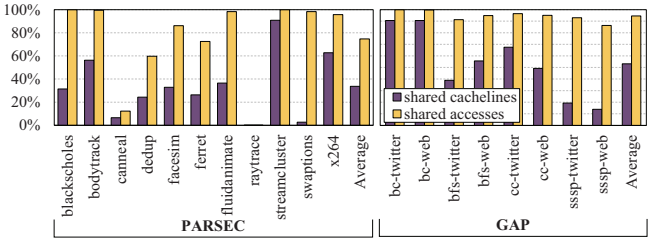


Fig. 3. Shared cacheline ratio and shared access ratio.

requests a cacheline that is being cached by another computing node H-1. After receiving MemRd from H-2, the memory node issues BISnpInv to invalidate the data cached in H-1[2]. Subsequently, H-1 forwards the data along with the MemWr message to the memory node if the cached data is dirty; on the other hand, the BIRspI message is directly responded to the memory node if the data is clean. Finally, the memory node would send Cmp-E and data to H-2, as in the previous examples.

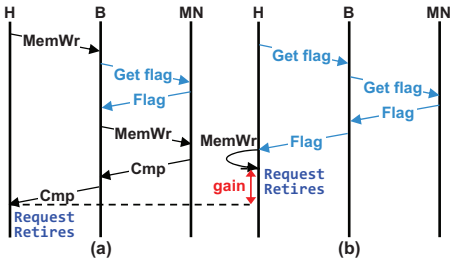


Fig. 4. Control flows of: (a) only address translation caching, (b) combined scheme (address translation caching + data caching). The blue arrows indicate transactions of permission flags.

The inefficiency of combining two caching schemes. In a disaggregated memory system, two independent caching schemes are expected to enhance system performance. The first scheme is our strong baseline, address translation caching scheme, to reduce the number of remote accesses incurred by two-level address translation (see Section II-A). The second one is a simple data caching mechanism that caches memory node data in a computing node. However, naively combining these two schemes results in an inefficiency issue. Figure 4 compares latency diagrams of two cases. Note that we assume that the local physical address is already translated to the global physical address for both cases. Also, one host node (H), backplane (B), and one memory node (MN) are assumed. Figure 4 (a) shows the control flow of the sole translation caching scheme. The memory request (e.g., MemWr) goes to MN because H does not cache the data. The backplane requests a permission flag from MN to check the access permission. Finally, the memory request is forwarded to MN if B confirms that there is no permission issue.

[2]The invalidation can be either broadcast or point-to-point depending on the implementation-specific data structure of the directory or the snoop filter in DCOH, as indicated in CXL 3.0.

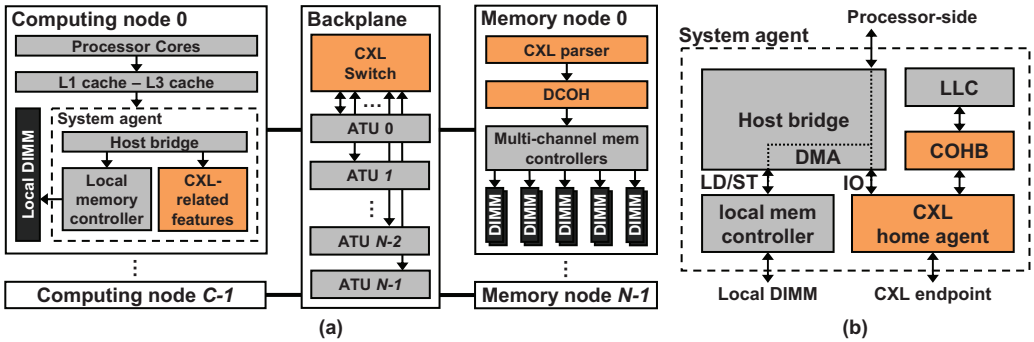


Fig. 5. System overview of SDM: (a) SDM architecture, (b) internal architecture of system agent. Orange blocks indicate new features.

Figure 4 (b) shows the timing diagram of the combined scheme. In this case, H may already have the requested data in its cache, but the data access permission still needs to be checked for the memory request. Thus, H directly requests a permission flag from MN to check the access permission. As B receives the flag from MN, B can forward the received flag to H. As such, the performance gain becomes the time difference between the two retirement points. Therefore, enlarging this difference can further improve the performance (Section IV-C).

IV. SDM: Sharing-enabled Disaggregated Memory

A. Architectural Overview of SDM

Focusing on a cache-coherent disaggregated memory system that is developed with CXL, we propose a sharing-enabled disaggregated memory, namely *SDM*, in which shared data can *co-exist* among multiple computing nodes. To enable such a feature, we first need to develop a disaggregated memory architecture that supports different channels of CXL protocols. Figure 5 illustrates an architectural overview of SDM. At a high level, the SDM consists of computing nodes, a backplane, and memory nodes.

In one of the computing nodes, the system agent[3] is augmented by introducing *CXL-related features* to the I/O path of the host bridge, which include a *coherence bridge (COHB)* and a *home agent*. The home agent acts like an interface between a computing node and a backplane. Once the home agent receives CXL messages (e.g., `RdOwn`), it redirects the messages to COHB, which is defined in the CXL specification. COHB reads and updates the coherence directory that is stored in the local memory. The coherence directory holds the coherence state of each cache line belonging to that computing node. Furthermore, contiguous allocation is possible for the coherence directory; hence, it can be indexed using algebraic computation. The latency of calculating the directory index would be negligible compared to the latency of others (e.g., interconnect). However, looking up in the cache directory is not considered in our evaluation, and hence the real implementation would have longer paths for coherence management. Rather than interrupting the processor, a direct memory access (DMA) engine is utilized for CXL features to directly communicate with local memory. Finally, the home agent generates CXL messages to respond to incoming request messages according to the states referenced by COHB.

In the backplane, the CXL switch distributes CXL requests from different computing nodes to the memory node. Before arbitrating requests, the CXL switch communicates with address translation units (ATU *i*), which perform the second-level page table walk to translate the local physical address to the global physical address.

In the memory node, two additional features are also included. First, the CXL parser obtains the source and request type information from an incoming CXL request. Second, the device coherency agent (DCOH), which is defined in the CXL specification, holds coherence states of memory nodes and generates `CXL.cache` messages based on the information extracted from the CXL parser. Additionally, the memory node might require a directory, namely the bias table managed by DCOH logic; the bias table holds coherence states if the memory node has a private data cache for caching data from other nodes. However, the introduction of private caches is beyond the scope of this paper.

In SDM, several design considerations must be addressed in detail. First, a control flow that facilitates data sharing in a multi-host disaggregated memory system is required instead of adopting a high-overhead control flow (i.e., INV-CF). Moreover, the control flow should not contradict the CXL specification (Section IV-B). Second, a set of remote memory management mechanisms for page allocation, deallocation, and page table walk is essential. The management mechanisms should not interfere with the normal memory transaction channels (i.e., `CXL.mem` and `CXL.cache`) to ensure higher system performance. We observe that these mechanisms can be implemented by smartly leveraging a special CXL channel instead of introducing additional sideband protocols (Section IV-C). Lastly, a speculative approach is proposed to harmoniously combine the address translation caching scheme and data caching scheme (Section IV-D).

[3]In some materials, *system agent* is referred to as root complex. In this work, we use the term system agent henceforth, as announced in [14].

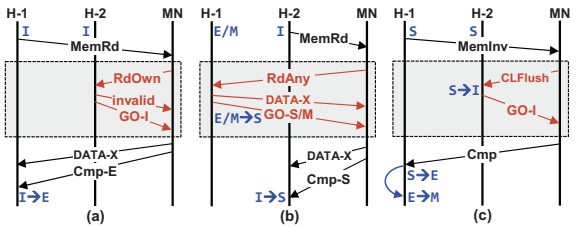


Fig. 6. Primitives of SHA-CF: (a) fetch-exclusive, (b) fetch-share, (c) invalidate-before-modify. Red arrows: `CXL.cache` messages. Black arrows: `CXL.mem` messages; Curved arrow: the internal write operation in the host.

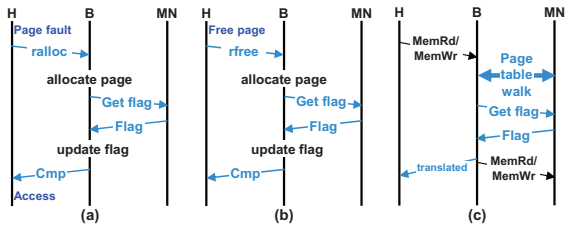


Fig. 7. Control flows of remote memory management primitives: (a) ralloc, (b) rfree, (c) rwalk. The blue arrows indicate `CXL.io` messages.

B. SHA-CF: Sharing-enabled Control Flow

In this section, we explain the proposed coherence control flow, SHA-CF, that enables data sharing among multiple computing nodes in SDM. The key idea behind SHA-CF is to *abstract all nodes*, except for the requester node, as a memory device and *emulate snooping* on multiple hosts by leveraging `CXL.cache` messages. Figure 6 shows three scenarios of crucial primitives of SHA-CF: *fetch-exclusive*, *fetch-share*, and *invalidate-before-modify*. Note that all scenarios in this figure assume two hosts (H-1 and H-2) and one memory node (MN); however, SHA-CF can be generalized to support multiple computing and memory nodes. Also, note that the key principle of our SHA-CF *strictly* obeys transaction protocols (i.e., request and response messages are paired) defined in the CXL specification without any redefinition.

First, the *fetch-exclusive* primitive (Figure 6 (a)) defines the control flow when a host (H-1) fetches data that is not shared by other hosts. To do so, H-1 sends the `MemRd` message to read data, `DATA-X`, from the memory node. Then, the memory node broadcasts `RdOwn` to check the existence of `DATA-X` in other hosts. Once the memory node receives `GO-I` with an invalid line from H-2, the memory node sends `Cmp-E` and `DATA-X` to let H-1 update the coherence state as E-state (exclusive).

The *fetch-share* primitive is an essential control flow for SDM to share data with multiple hosts. In Figure 6 (b), we assume that H-1 initially holds `DATA-X`. Firstly, H-2 issues a read request for `DATA-X` to the memory node. The memory node then broadcasts `RdAny` similar to the *fetch-exclusive* primitive. Different from the case of *fetch-exclusive*, H-1 responds `DATA-X` along with `GO-S` if the data is not dirty. If `DATA-X` is dirty, `GO-M` is returned instead to forward the up-to-date data to H-2. Note that memory node needs to write back `DATA-X` if the memory node receives `GO-M` and `DATA-X`, because `GO-M` indicates that `DATA-X` was modified (M-state) in H-1.

The *invalidate-before-modify* primitive is provided to address the inconsistency issue, as the cached data may be modified. In Figure 6 (c), both hosts are sharing `DATA-X`, which would be modified by H-1 in this case. Before updating `DATA-X`, H-1 issues `MemInv` to the memory node to notify back-invalidation. The memory node thereby broadcasts `CLFlush` to invalidate `DATA-X`. Thus, H-2 responds with `GO-I` to the memory node after invalidating its cache line. Then, H-1 modifies `DATA-X` after H-1 receives `Cmp` as the response to `MemInv`. Note that the *invalidate-before-modify* primitive can be implemented using back-invalidation messages in CXL 3.0 with fewer transactions.

C. Remote Memory Management

In this subsection, we explain a mechanism to manage page allocations in SDM. To prevent interference with normal memory transactions, SDM exploits a *vendor-defined* message field.[4] The proposed mechanism incorporates three remote page management primitives: *ralloc*, *rfree*, and *rwalk*. Figure 7 illustrates the procedures of three primitives, where one host (H), backplane (B), and one memory node (MN) are shown in the examples.

The *ralloc* primitive newly allocates a physical page of the memory node. In Figure 7 (a), the computing node sends an *ralloc* request to the backplane, as the host MMU triggers the initial page fault (e.g., modified `do_page_fault()` in the kernel). Then, the address translation unit (ATU) in the backplane allocates a page and updates the corresponding permission flag, thereafter sending the completion signal, `Cmp`, to the computing node.

The *rfree* releases the allocated page of a memory node. In Figure 7 (b), the computing node triggers the de-allocation process of a remote page (e.g., modified `deallocuvm()` in the kernel) and sends the *rfree* message. In this case, the address translation unit de-allocates the requested page and unmasks the corresponding permission flag.

The last primitive, *rwalk*, translates the local physical address to the global physical address. In Figure 7 (c), a normal memory request (e.g., `MemRd`) goes to the backplane. The second-level page table walk then occurs by repeatedly accessing the page table in the memory node. After acquiring permission flags, the original request is redirected to the memory node. Note that the translated information can also be brought to the host, because the translation caching is assumed for a stronger baseline (see Section II-A). Thus, the proposed mechanism needs the modification of the host MMU to manage three primitives and their responses. Furthermore, as mentioned in [32], the coherency of cached translation is managed using invalidation-based handling. For example, a mapping from a local physical address to a global physical

[4]It is defined as the byte-15 of `CXL.io` transaction-level packet.

address is changed due to job migration. Subsequently, ATU would invalidate the cached translation by dispatching requests that carry the local physical address. In our system, since address translation is managed through the `CXL.io` channel, the invalidation can also be performed through the same channel by defining a dedicated message with the byte-15 of `CXL.io`.

D. Speculative Access

The speculative access in SDM is used to harmonically combine the address translation caching scheme and data caching scheme. The main idea of speculative access is *access-before-permit*; that is, the memory request is speculatively processed before checking the permission flag. Thus, the computing node does not need to wait for the permission flags from the memory node (see Figure 4 (b)).

After the permission flag arrives at the computing node, the CXL home agent hardware performs the permission verification process. For example, for a write request, the original write request retires if the request is permitted on the corresponding data; otherwise, the updated data must be rolled back to the original state with the "replay data". To support this replay behavior, the speculative access requires a replay buffer in the computing node. Once the roll-back is required, the old data entries in the replay buffer are used to restore the updated data back to the old state if the speculation fails for the write request (i.e., write is not permitted). For the secure operation of the system, if the data is illegitimately accessed by the malicious session, access control violation handling will isolate the corresponding session to prevent further requests from that user session.

E. Modification of System Agent

To support CXL transactions in SDM, the system agent must be modified accordingly. Figure 5 (b) drills down the system agent in a computing node. A system agent cache is located in the system agent to cache the remote data; the data in the system agent cache can also be cached by L1-L3 caches of processor cores. Next to the system agent cache, the host bridge multiplexes the memory request to either the local DIMM controller or the CXL home agent; the home agent acts as an interface between a computing node and a backplane. Once the home agent receives the coherence-related message, it redirects the message to the coherency bridge (COHB), which reads and updates the coherence directory that is stored in the local memory. Note that the coherence directory is an essential feature to store the coherence state of each line in the system agent cache. Without interrupting the processor, a direct memory access (DMA) engine is introduced in the CXL home agent to directly communicate with the local memory.

V. Experimental Methodologies

Simulating a multi-node disaggregated memory system on a cycle-level simulator is challenging due to the prohibitively long simulation time. Hence, following the methodologies in prior works [10], [36], we develop a lightweight, in-house evaluation platform using Intel PIN tool [38]. In our simulator, multiple computing nodes and one memory node are modeled; statistics for each node are evaluated while running workloads on the *real machine*. For speculative access, we optimistically assume that speculation is always true, as access control cannot be extracted by the user-level Intel PIN tool; furthermore, most applications are read-intensive.

TABLE II
System configurations.

Computing node [9]	
L1 cache	8-way set associative, 32KB, 64B line, 1ns
L2 cache	4-way set associative, 256KB, 64B line, 4ns
L3 cache	16-way set associative, 2MB, 64B line, 40ns
Memory latency	80ns
Memory node	
Latency	80ns
Network interconnection [32]	
Latency	500ns

To manage coherence in our simulator, we use a global table at a cacheline granularity [10]. Each entry consists of an address and the coherence states of all computing nodes; hence, proper transaction messages are generated when a coherence state changes. For example, messages of *fetch-exclusive* are issued if the table entry of the accessed line initially has four **I** states. Then, one of the coherence states corresponding to the accessing node becomes the **E** state.

Meanwhile, the statistics (e.g., the number of transactions and the executed cycles for each memory operation) are accumulated in the evaluation runtime. We also can obtain the average memory access time (AMAT) using the aforementioned statistics and the following formula:

$$\begin{aligned} AMAT = {} & t_{permit} + t_{L1} + MR_{L1} \\ & \times (t_{L2} + MR_{L2} \times (t_{L3} + MR_{L3} \times t_{RM})) \end{aligned} \tag{1}$$

$$\begin{aligned} t_{RM,avg} &= (\sum_{p}^{P} N_p \times TX_p) \times t_{network}/RMs \\ P_{INV-CF} &= \{fetch_dirty, fetch_clean\} \\ P_{SHA-CF} &= \{fetch_exclusive, fetch_share, \\ &\qquad invalid_before_mod\} \end{aligned} \tag{2}$$

In Equation (1), t_{permit} denotes the latency of the permission check; t_{Lx} denotes the cache hit latency at Lx-level (see Table II); MR_{Lx} denotes the cache miss rate of Lx cache, which can be obtained from the runtime statistics. Equation (2) shows the $t_{RM,avg}$ (i.e., remote access latency) of the system. P denotes a set of valid primitives for a specific system. We use P_{INV-CF} and P_{SHA-CF} as P to evaluate INV-CF and SHA-CF, respectively. In the equation, N_p denotes the number of a primitive at runtime; TX_p denotes the number of messages required for a primitive; $t_{network}$ denotes the network interconnection latency; RMs is the number of remote memory requests. All equation parameters are obtained from the runtime statistics.

Table II summarizes configurations. Note that data fully reside in the memory node because the memory footprint of disaggregated memory workloads is typically larger than a local memory capacity. We use the same memory hierarchy latency and network latency values used in [9] and [32], respectively. Also, we execute 32 threads for each workload for every simulation, where threads are uniformly assigned to each compute node.

Workloads. We evaluate our design (SDM) with two benchmark suites that have distinctive characteristics: PARSEC (computation-intensive) [8] and Intel GAP (memory-intensive) [7]. PARSEC is a computation-intensive suite including large-scale commercial workloads. We evaluate 11 workloads in PARSEC benchmark suites: *blackscholes*, *bodytrack*, *canneal*, *dedup*, *facesim*, *ferret*, *fluidanimate*, *raytrace*, *streamcluster*, *swaptions*, and *x264*. In contrast, Intel GAP consists of graph analytic workloads, which are likely more realistic in disaggregated memory systems. We use two real-world graphs, *twitter* and *web*, to which four kernels are applied, *bfs* (breadth-first search), *bc* (betweenness centrality), *cc* (connected components), and *sssp* (single-source shortest paths). Therefore, 8 graph workloads are evaluated by combining these various graphs and kernels.

VI. Results

We quantitatively evaluate three different configurations: INV-CF, SDM (SHA-CF), and SDM-full (SHA-CF + speculative access). Note that address translation caching scheme [32] is applied to INV-CF, which is regarded as the *strong baseline* (see Section II-A).

A. Speedup Comparison

In this subsection, we evaluate IPC and average memory latency of SDM. IPC implies the overall throughput of the system; the average latency indicates the average duration between the request generation and retirement. IPC is not a good metric for evaluating multithreaded applications in a *real machine*, because real microarchitectural components may reduce the number of executed instructions spent on codes, such as spin-lock loops in these applications [3]. However, our in-house simulation is based on the PIN tool, and only the user-level load/store instructions are injected into our disaggregated memory system model. That is, our simulation is similar to trace-driven simulation; hence, IPC can directly show performance improvement in our case.

Figure 8 (a) presents IPC values normalized to INV-CF. In general, SDM yields higher IPC for both benchmark suites. For PARSEC, SDM shows 23% higher performance (maximally 2.08×); however, some workloads (e.g., *canneal*, *raytrace*) show minor improvement, because these workloads have relatively low miss rates and low shared ratios and (see Section III-B). For example, *raytrace* yields nearly 100% of LLC miss rate according to our evaluation. On the other hand, the performance improvement becomes more significant when it comes to Intel GAP, as it yields 86% higher IPC

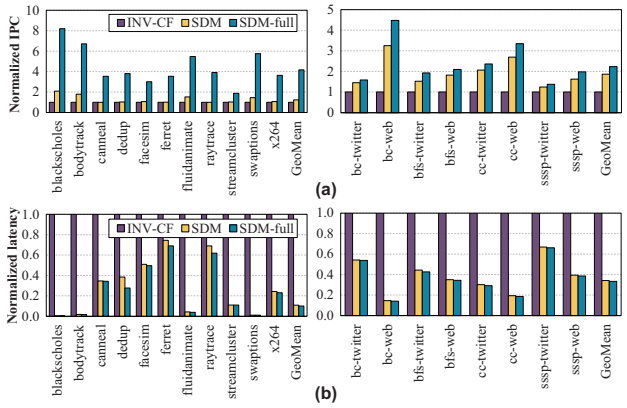


Fig. 8. Speedup of different SDM configurations: (a) normalized IPC, (b) normalized latency.

(maximally 4.48×). This is because, Intel GAP is memory-intensive, resulting in higher benefits from the data sharing among multiple hosts. Furthermore, SDM-full outperforms the baseline by 4.16× and 2.23× for PARSEC and Intel GAP, respectively. These values are also higher than SDM, because the speculative access mechanism significantly reduces the permission check overhead of remote write requests. For example, the remote memory access ratio of SDM is 28%, leading to significant performance improvement.

Figure 8 (b) shows average memory latency of SDM and SDM-full over the baseline. In general, SDM and SDM-full reduce 90% and 91% of memory latency, respectively, for PARSEC. For Intel GAP, the latency is reduced by 66% and 67%. The latency improvement of SDM-full is relatively small compared to SDM, because the speculative access buffers requests in a replay buffer for higher processing throughput, whereas the latency is the time difference between the generation point to the retirement point of requests.

B. Sensitivity to the Number of Compute Nodes

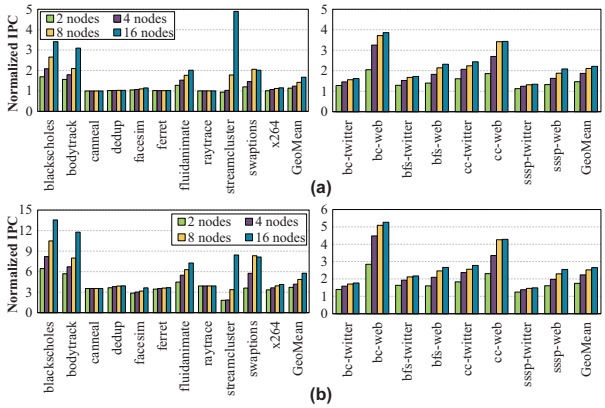


Fig. 9. IPC regarding different numbers of hosts: (a) SDM, (b) SDM-full.

A scalable industrial data center consists of numerous computing nodes, which highly affect the overall system

performance. Therefore, this subsection discusses the system performance by varying the number of computing nodes and network latency. Figure 9 presents normalized IPC values of SDM and INV-CF by varying the number of computing nodes. Particularly, Figure 9 (a) shows the performance of SDM, as the improvement becomes more notable with the increased number of computing nodes for both benchmark suites. For PARSEC, the IPC improvement reaches up to 67% when the number of computing nodes becomes 16. For Intel GAP, the IPC is improved by up to 2.21×. Such increasing tendencies indicate that the overhead of SHA-CF coherence management is negligible, whereas the benefit from data sharing capability is significant. In Figure 9 (b), the speedup becomes further notable for SDM-full. The speedups are 5.77× and 2.65× for PARSEC and Intel GAP, respectively, when the number of computing nodes is extended to 16. In general, the performance of graph workloads does not scale well compared to computation-intensive workloads. As the number of nodes grows, the shared data in the memory node will be accessed by different nodes with higher probability. Consequently, the coherence management between different compute nodes becomes more frequent. Furthermore, the memory intensity of Intel GAP is higher than that of PARSEC, yielding lower improvement.

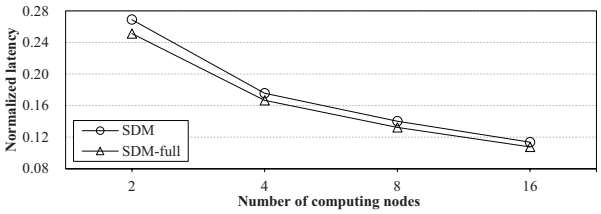


Fig. 10. Sensitivity analysis of latency regarding different numbers of computing nodes.

Figure 10 shows the average memory latency of SDM and SDM-full normalized to INV-CF, regarding different numbers of computing nodes. Each point value is normalized to INV-CF with the same number of computing nodes. Similar to IPC improvement, the latency reduction rate also becomes more significant, as the number of computing nodes increases. For example, the latency reduction rate achieves 91% for a disaggregated memory system having 16 computing nodes. In conclusion, SDM is scalable regarding the increasing number of computing nodes.

C. Sensitivity to the Network Latency

The network latency would become a major bottleneck due to its high latency; the reason for the high latency can be slow interconnection speed or the arbitration contention by numerous nodes. Specifically, the latency of network interconnection ranges from a low of about 100 ns to a high of about 1000 ns, according to previous announcements of network interconnection [26], [49]. In this subsection, the average memory access latency is evaluated by varying the network interconnection latency.

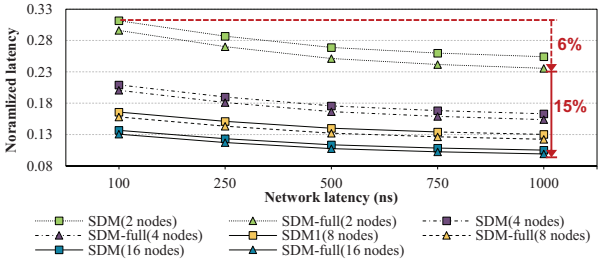


Fig. 11. Sensitivity analysis of latency regarding network latency and numbers of computing nodes.

Figure 11 shows the average memory latency with varying network latency and the number of computing nodes, where all results are normalized to INV-CF. In general, both SDM and SDM-full yield larger latency reduction rates, as both the network latency value and the number of computing nodes become larger. For example, for SDM with two computing nodes, the latency reduction rate enlarges from 69% to 75% (i.e., 6%) when the network latency varies from 100 ns to 1000 ns, and SDM-full yields 1% lower latency compared to SDM. Furthermore, for the 1000 ns of network latency, the latency reduction rate is increased by 15% when the number of computing nodes doubles from two. Consequently, SDM facilitates a high-performance disaggregated memory system with high scalability, even for high-latency network.

VII. Related Work

Table III shows the qualitative comparison of related works of disaggregated memory systems. This section discusses the works related to SDM in detail.

Disaggregated memory as swap space. Disaggregated memory has been utilized as a faster swap space compared to storage devices. Scarce physical memory incurs swap operations between main memory and storage devices, leading to significant performance degradation. Therefore, the latency of the swap space allocated in storage devices determines the overall performance of a system. Previous works leverage a memory node as swap space to notably reduce the swap latency [5], [21], [36], because accessing memory devices in the memory node requires much shorter latency than accessing storage devices. In [36], [37], hypervisors are augmented to treat remote memory nodes in disaggregated memory systems as fast swap space, while preventing modification on both OSes and applications. In [5], a swap mechanism (or swap system) is directly developed on a local operating system rather than on a hypervisor to further reduce interface overheads. To simplify the implementation, the authors of [21] propose a swap mechanism by simply adding a virtual block device interface on a local operating system. However, neither faster nor larger swap space addresses the memory shortage issue, because the modern operating system throws the OOM killer and incurs system failure if a system runs out of physical memory. On the other hand, SDM expands the main memory

TABLE III
QUALITATIVE COMPARISON OF DIFFERENT DISAGGREGATED MEMORY SYSTEMS.

Architecture	Transparency	Granularity	Role	Interconnect protocol	Coherency of compute nodes
PS [36]	✓	page	swap space	PCIe	not supported
FaRM [17]	✗	page	memory pool	one-sided RDMA	not supported
Grappa [40]	✗	page	memory pool	Infiniband	software synchronization
Infiniswap [21]	✓	page	swap space	one-sided RDMA	not supported
LegoOS [46]	✓	page	memory pool	two-sided RDMA	not supported
Remote Regions [2]	✗	page	memory pool	RoCE	software synchronization
DCM [31]	✓	page	memory pool	Infiniband	not supported
XMemPod [11]	✓	page	memory pool	Infiniband	not supported
FastSwap [36]	✓	page	swap space	one-sided RDMA	not supported
pDPM [50]	✗	page	memory pool	one- & two-sided RDMA	not supported
RACE [53]	✗	page	memory pool	one-sided RDMA	atomic update (RDMA ATOMIC)
Kona [10]	✓	cacheline	memory pool	one-sided RDMA	not supported
DeACT [32]	✓	page	memory pool	unknown	not supported
MIND [34]	✓	page	memory pool	one-sided RDMA	customized hardware manager
Clio [22]	✗	page	memory pool	customized interconnect	software synchronization
SDM (proposed)	✓	cacheline	memory pool	CXL	hardware (a feature from CXL)

pool based on a disaggregated memory system, potentially leading to higher system reliability.

Abstraction of disaggregated memory. Several works have developed frameworks that abstracts the remote memory management mechanisms [2], [17], [22], [40], [50], [53]. For example, in [2], a new file system is proposed to access remote memory as files in volatile storage. Similar to Intel PMDK [25] that provides persistent objects, [17] and [40] implement application-specific object classes to simplify the remote memory management for in-memory database and graph applications, respectively. In [53] and [50], application-specific interfaces are developed to reduce the access latency of hashing and key-value store data structures, respectively. In [22], a *malloc-like* interface, which minimizes the application modification, is developed based on its own hardware platform. Nevertheless, these approaches require application modifications, making programs error-prone. In contrast, our SDM facilitates user-transparent remote memory accesses and memory management mechanisms.

User-transparent main memory pool. Some prior works have been proposed to develop high-capacity main memory pools based on disaggregated memory systems that are transparent to user-level applications [10], [11], [31], [32], [34], [46]. For example, [46] is the first memory pool disaggregation work that splits operating system kernels at device granularity. In [11], a hierarchical memory orchestration system is introduced to transparently allocate physical memory according to the access latency of different devices (i.e., local memory, remote memory, and storage devices). In [34], the remote memory management mechanisms are built in the programmable switch, minimizing extra costs on computing nodes and memory nodes. However, these works access the remote memory by always triggering page fault, leading to frequent context switches between the application and the operating system. To overcome this limitation, a platform that accesses the remote memory in a fine-grained manner is proposed [10]; it hooks memory management function calls (e.g., *malloc* and *free*) using dedicated hardware primitives. Still, this approach does not consider the data coherence among multiple computing nodes.

In conclusion, most works access remote memory at page granularity with RDMA, amplifying memory transaction overheads. Furthermore, several works resolve the data coherency among computing nodes using software-based synchronization methods, which let a process or a thread exclusively access the critical section. Such approaches generally incur significant performance overheads. In contrast, our SDM accesses the remote memory at cacheline granularity owing to the intrinsic characteristics of CXL while preserving the coherency.

VIII. CONCLUSION

This paper proposes a user-transparent cache coherent disaggregated memory system, SDM, by fully leveraging the transaction-level cache coherent protocol, CXL; it is the first academic work of a data sharing-enabled, CXL-based multi-host disaggregated memory system. Our SDM employs a novel cache coherent control flow, SHA-CF, to facilitate data sharing between multiple hosts. The SHA-CF emulates the snooping among multiple hosts without violating the CXL specification. To manage remote memory resources, we also propose several essential primitives; our management mechanisms leverage the reserved message types in the `CXL.io` packet. Thus, the normal memory transaction channel is not disturbed. Lastly, speculative access is proposed to efficiently combine two orthogonal caching schemes by simply adding a tiny replay buffer to the computing node. Our evaluation results show that the proposed architecture significantly improves the performance compared to the strong baseline. We envision that our proposed CXL-based architecture would pave a new way for a scalable, high-performance disaggregated memory system in the future.

ACKNOWLEDGMENT

We appreciate the anonymous reviewers for their valuable comments to improve the quality of our paper. This work was supported by a research grant from SK Hynix and by the Ministry of Science and ICT under the ITRC support program (IITP-2023-RS-2022-00156295) and the artificial intelligence semiconductor support program to nurture the best talents (IITP-2023-RS-2023-00256081) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation). The Institute of Engineering Research at Seoul National University provided research facilities for this work. Jaewoong Sim is the corresponding author.

REFERENCES

[1] A. Agarwal, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, K. Kurihara, B. H. Lim, G. Maa, D. Nussbaum, M. Parkin, and D. Yeung, "The mit alewife machine: A large-scale distributed-memory multiprocessor," Tech. Rep., 1991.

[2] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, S. Novaković, A. Ramanathan, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei, "Remote regions: a simple abstraction for remote memory," in *2018 USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[3] A. Alameldeen and D. Wood, "Ipc considered harmful for multiprocessor workloads," *IEEE Micro*, vol. 26, no. 4, pp. 8–17, 2006.

[4] K. Alnaes, E. Kristiansen, D. Gustavson, and D. James, "Scalable coherent interface," in *Proceedings of International Conference on Computer Systems and Software Engineering (CompEuro)*, 1990.

[5] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker, "Can far memory improve job throughput?" in *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*, 2020.

[6] AMD. (2022) 4th gen amd epyc processor architecture. [Online]. Available: https://www.amd.com/en/campaigns/epyc-9004-architecture

[7] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," 2015. [Online]. Available: https://arxiv.org/abs/1508.03619

[8] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[9] B. Brett. (2016) Memory performance in a nutshell. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/memory-performance-in-a-nutshell.html

[10] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[11] W. Cao and L. Liu, "Hierarchical orchestration of disaggregated memory," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 844–855, 2020.

[12] J. B. Carter, C.-C. Kuo, and R. Kuramkote, "A comparison of software and hardware synchronization mechanisms for distributed shared memory multiprocessors," Tech. Rep., 1996.

[13] CCIX, "Cache coherent interconnect for accelerators," Sep 2019.

[14] I. Cutress. (2015) The intel broadwell xeon e3 v4 review: 95w, 64w and 35w with edram. [Online]. Available: https://www.anandtech.com/show/9532/the-intel-broadwell-xeon-e3-v4-review-95w-65w-35w-1285-1285l-1265

[15] CXL, "Compute express link specification revision 2.0," Oct 2020.

[16] CXL, "Compute express link (cxl) specification revision 3.0," 2022. [Online]. Available: https://www.computeexpresslink.org/download-the-specification

[17] A. Dragojevi, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[18] A. Farmahini-Farahani, S. Gurumurthi, G. Loh, and M. Ignatowski, "Challenges of high-capacity dram stacks and potential directions," in *Proceedings of the Workshop on Memory Centric High Performance Computing*, 2018.

[19] F. Glaser, G. Tagliavini, D. Rossi, G. Haugou, Q. Huang, and L. Benini, "Energy-efficient hardware-accelerated synchronization for shared-l1-memory multiprocessor clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 633–648, 2020.

[20] D. Gouk, S. Lee, M. Kwon, and M. Jung, "Direct access, High-Performance memory disaggregation with DirectCXL," in *2022 USENIX Annual Technical Conference (USENIX ATC)*, 2022.

[21] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[22] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: A hardware-software co-designed disaggregated memory system," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[23] B. Huang, L. Jin, Z. Lu, M. Yan, J. Wu, P. C. Hung, and Q. Tang, "Rdma-driven mongodb: An approach of rdma enhanced nosql paradigm for large-scale data processing," *Information Sciences*, vol. 502, pp. 376–393, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025519305869

[24] Intel, "Intel® 64 and ia-32 architectures software developer's manual volumn 3a: System programming guide, part 1." [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[25] Intel. Persistent memory development kit. [Online]. Available: https://pmem.io/pmdk/

[26] Intel, "Intel omni-path architecture (packet integrity protection and local link integrity counter white paper)," 2017. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/IntelOPA_PktIntegProt_WP_J79955_v1_0.pdf

[27] Intel, "Intel architecture day 2021," 2021. [Online]. Available: https://www.intel.com/content/www/us/en/newsroom/resources/press-kit-architecture-day-2021.html

[28] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, "Basic performance measurements of the intel optane DC persistent memory module," 2019. [Online]. Available: http://arxiv.org/abs/1903.05714

[29] M. Jung, "Hello bytes, bye blocks: Pcie storage meets compute express link for memory expansion (cxl-ssd)," in *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, 2022.

[30] X. Z. Khooi, C. H. Song, and M. C. Chan, "Towards a framework for one-sided rdma multicast," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2022.

[31] K. Koh, K. Kim, S. Jeon, and J. Huh, "Disaggregated cloud memory with elastic block management," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 39–52, 2018.

[32] V. R. Kommareddy, C. Hughes, S. D. Hammond, and A. Awad, "Deact: Architecture-aware virtual memory support for fabric attached memory systems," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[33] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy, "The stanford flash multiprocessor," in *Proceedings of the 21st Annual International Symposium on Computer Architecture (ISCA)*, 1994.

[34] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong, and A. Bhattacharjee, "Mind: In-network memory management for disaggregated data centers," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP)*, 2021.

[35] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: Cxl-based memory pooling systems for cloud platforms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[36] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, 2009.

[37] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2012.

[38] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2005.

[39] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "Tpp: Transparent page placement for cxl-enabled tiered-memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[40] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin, "Latency-Tolerant software distributed shared memory," in *2015 USENIX Annual Technical Conference (USENIX ATC)*, 2015.

[41] L. K. Organization. Out of memory management. [Online]. Available: https://www.kernel.org/doc/gorman/html/understand/understand016.html

[42] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *2011 IEEE Hot Chips 23 Symposium (HCS)*, 2011.

[43] G. Rodrigues. (2009) Taming the oom killer. [Online]. Available: https://lwn.net/Articles/317814/

[44] Samsung. (2021) Samsung brings in-memory processing power to wider range of applications. [Online]. Available: https://news.samsung.com/global/samsung-brings-in-memory-processing-power-to-wider-range-of-applications

[45] Samsung. (2022) Samsung electronics introduces industry's first 512gb cxl memory module. [Online]. Available: https://news.samsung.com/us/samsung-electronics-introduces-industrys-first-512gb-cxl-memory-module/

[46] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "LegoOS: A disseminated, distributed OS for hardware resource disaggregation," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018.

[47] SKHynix. (2022) Sk hynix develops ddr5 dram cxl memory to expand the cxl memory ecosystem. [Online]. Available: https://news.skhynix.com/

[48] Tensorflow. Use tpus. [Online]. Available: https://www.tensorflow.org/guide/tpu?hl=en

[49] T. Trader. (2021) With new owner and new roadmap, an independent omni-path is staging a comeback. [Online]. Available: https://www.hpcwire.com/2021/07/23/with-new-owner-and-new-roadmap-an-independent-omni-path-is-staging-a-comeback/

[50] S.-Y. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated Key-Value stores," in *2020 USENIX Annual Technical Conference (USENIX ATC)*, 2020.

[51] K. Wu, A. Arpaci-Dusseau, R. Arpaci-Dusseau, R. Sen, and K. Park, "Exploiting intel optane ssd for microsoft sql server," in *Proceedings of the 15th International Workshop on Data Management on New Hardware*, 2019.

[52] J. Yang, B. Li, and D. J. Lilja, "Exploring performance characteristics of the optane 3d xpoint storage technology," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 5, no. 1, pp. 1–28, 2020.

[53] P. Zuo, J. Sun, L. Yang, S. Zhang, and Y. Hua, "One-sided RDMA-Conscious extendible hashing for disaggregated memory," in *2021 USENIX Annual Technical Conference (USENIX ATC)*, 2021.