Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Towards scalable and configurable simulation for disaggregated architecture

Daegyeong Kim [a], Wonwoo Choi [a], Chang-il Lim [a], Eunjin Kim [a], Geonwoo Kim [a], Yongho Song [a], Junsu Lee [a], Youngkwang Han [a], Hojoon Lee [b,*], Brent Byunghoon Kang [a,*]

[a] Graduate School of Information Security, Korea Advanced Institute of Science and Technology, South Korea
[b] Department of Computer Science and Engineering, Sungkyunkwan University, South Korea

ARTICLE INFO

Keywords:
Disaggregated architecture
System simulator
Cloud computing

ABSTRACT

The increasingly more data-intensive workloads imposed on cloud computing are calling for fundamental changes in the computer architecture level. The current machine-oriented architectures, where hardware resources, such as main processors, accelerators, and memory, are rigidly tied to a machine, do not allow flexible resource management. Disaggregated architectures propose breaking the tight bonding of the compute and storage components inside a machine. Each component becomes a node on a high-bandwidth fabric interconnect network and is managed in resource pools of its type. The disaggregated architecture cloud can be flexibly scaled in a fine-grained manner to react to the demanded workload. However, the architectures are often inaccessible to researchers as they employ specialized hardware or require a large-scale investment to set up a testing environment. This paper presents the design and implementation of a simulator for disaggregated architectures called DisaggSim. We generalize existing disaggregated architectures and design an extensible and detailed simulation model. Also, we implement the simulator on top of gem5, a broadly-used system simulator in the research community. Through our evaluation, we demonstrate that DisaggSim can be used to evaluate various performance characteristics of a given disaggregated architecture specification. We publicly release the simulator to facilitate rapid prototyping and evaluation of research on disaggregated architectures.

1. Introduction

Recently, both academia and industry have devoted considerable efforts towards disaggregation of the cloud. Disaggregation of cloud computing resources detaches the computing resources, such as processors, memory, and accelerators, from traditional machine-oriented machines, such that each resource can be managed independently according to various workloads.

In *machine-oriented architectures*, hardware resources are organized in the granularity of *machines*. Such resources (processors, memory, and accelerators) are bound to a motherboard. This tight coupling between hardware resources typically undermines the flexibility of resource management in cloud environments. For instance, cloud service providers cannot augment a single resource category independently because the number of resources that can be equipable by a single machine is limited (e.g., the number of memory slots). The demand for a solution to the problem is on the rise, given today's data-intensive workloads in the cloud that require disproportionately higher memory and accelerator capacities.

* Corresponding authors.
E-mail addresses: hojoon.lee@skku.edu (H. Lee), brentkang@kaist.ac.kr (B.B. Kang).

https://doi.org/10.1016/j.simpat.2023.102743
Received 12 September 2022; Received in revised form 4 February 2023; Accepted 27 February 2023

A paradigm shift to fundamental computer architecture has recently arisen to address the inflexible resource management problem. The main concept of disaggregated architecture is to, as literally, *disaggregate* tightly combined resources and *aggregate* them into flexible resource pools. In the architecture, processors and accelerators are represented as compute nodes, and memory resources are described as memory pool nodes. Each node includes a single resource type, and the fabric interconnect extensively connects the nodes. A system administrator can configure the system to use each type of resource as desired and combine them as a subsystem, which is served as a complete virtual set of a computer.

Several proprietary implementations for disaggregated architectures have been proposed in the industry, such as Gen-Z specification [1], Intel's Rack Scale Design (RSD) [2], Hewlett Packard Enterprise (HPE)'s 'the machine [3],' and etc [4,5]. Recently, IntelliProp has developed the industry's first fully disaggregated and composable memory fabric with Compute Express Link (CXL) Standard [6].

In academia, prior work includes hardware-based approaches [7–17], software-based approaches [18–34]. These disaggregated resources are all directly attached to the network fabric for inter-resource communication, whereas CPU and memory are connected to the system bus in a machine-oriented architecture.

It is worth noting that there is no publicly available simulator for disaggregated systems. The availability of such a simulator could help researchers explore new disaggregated cloud designs. Many disaggregated systems are proprietary and inaccessible to researchers [3–5]. Such architectures employ specialized hardware [35,36], which researchers cannot access, and simulation environments [12,19,29,37] that may or may not exist internally are not open to the public or are purpose-specific. Moreover, they often require a sizeable corporate-scale investment to prototype. Therefore, a simulator that accurately models such systems would facilitate the advancement of this field.

In this paper, we present DisaggSim, a scalable and configurable simulation tool to facilitate prototyping and evaluation of new research ideas for a disaggregated architecture. We generalize the characteristics of common disaggregated architectures and identify the role of the key components that constitute disaggregated cloud architecture. We also identify and define 24 performance and architectural parameters based on our comprehensive study of the related literature and documentation of commercial prototypes. We implement DisaggSim based on the generalized disaggregated architecture using the gem5 simulator [38], the most extensively used architectural simulator in academia and industry. We also evaluate DisaggSim using *unmodified* benchmark applications (SPECint2006 [39]) and memory bandwidth benchmark applications (STREAM [40]) with various configuration parameters and values. The results of the experiments demonstrate how design factors affect the performance of a disaggregated system. Furthermore, we provide helpful insights to solve the current disaggregated system problems and research direction for future disaggregated architectures. Finally, we publicly open the source code of DisaggSim to allow researchers to prototype and evaluate new disaggregated architecture designs.[1]

This paper is structured as follows: We first provide the background of the paper in Section 2. Next, we present the design goals of DisaggSim in Section 3. Sections 4 and 5 describes the detailed design and implementation of DisaggSim. In addition, Section 6 demonstrates the evaluation of DisaggSim using benchmarks. Section 7 discusses future directions and limitations of this work. Finally, we conclude the paper in Section 8.

2. Background and related work

2.1. Disaggregated architecture

Disaggregated architectures have recently gained traction in industry [1–5] and academia [7–34] as future generation architectures. Disaggregated architectures decouple individual hardware components that have been traditionally linked to machines. The components, such as CPUs and memory modules, disaggregated from a machine are connected to a shared, system-wide *fabric interconnect*, supporting high-bandwidth communication between attached components. Unlike input/output (I/O) channels of machine-oriented architecture directly managed by CPUs and motherboard chipsets, a fabric interconnect is an independent component and exposes abstract interfaces for connection. Each hardware component can be independently attached to or detached from the interconnect, allowing the system configuration to be changed more flexibly, even if resource requirements are unpredictable.

The performance of the fabric interconnect is critical for disaggregated systems due to the nature of resource disaggregation that (almost) all traffic between resources is transferred through the fabric interconnect, and the volume of traffic increases. Therefore, the critical point in resource disaggregation is to minimize the performance degradation of the entire system due to an interconnect bottleneck [41].

Thus, the existing disaggregated system prototypes use or are assumed to use various specialized or state-of-the-art interconnect technologies to meet their required performance. The used interconnect technologies can be categorized by their materials: electrical [5,12,29,42], optical [3,11], both [9], or either [1,2]. Although designing a faster fabric interconnect is necessary [19], the performance of fabric interconnect is limited to the current level of technologies and physical materials. Furthermore, composing a disaggregated system with products of state-of-the-art network devices needs large-scale investment. For example, a single switch device of InfiniBand [36] costs at least tens of thousands of dollars. These limitations are main reasons that disaggregated architectures are still in the research stage and inspire us to develop DisaggSim measuring how its (network-related) parameters and values influence the entire performance of a simulated disaggregated system.

[1] Our source code is publicly available at https://github.com/KAIST-CysecLab/DisaggSim.

2.2. Proprietary architectures

In the early stages, individual vendors presented disaggregated systems for enterprise purposes, such as HPE 'The Machine' [3], Intel RSD [2], Facebook Disaggregated Rack [4], and Huawei NÜWA [5]. Intel RSD publicly provides APIs compatible with the Redfish framework to manage each resource. However, Intel RSD is still in the developing phase and does not currently support memory disaggregation. AMD's dense server business, formerly SeaMicro, released a disaggregation-enabled server [43] as a product. However, they opened only the abstract and several features of their systems.

Several industry-leading companies have built consortia to develop disaggregated systems and have published their prototypes or specifications. OpenCAPI [44], being led by IBM, tries to solve problems of existing I/O operation in emerging accelerated computing and advanced memory/storage devices. They also published ThymesisFlow [7], a prototype and the first HW/SW co-designed model using an OpenCAPI port. In addition, dReDBox project group [45], funded by the European Union and led by IBM Research Ireland, published dReDBox [9,10]. Gen-Z consortium [1], led by HPE, disclosed several features and specifications, while they have not released a prototype. HPE's Gen-Z protocol and IBM's OpenCAPI protocol have been recently merged into CXL consortium [35]. IntelliProp, a leading company for composable memory in HPC, developed a fully disaggregated memory fabric using CXL Standard [6].

2.3. Academic work

Previous academic work refers to disaggregation of compute and memory resources as far memory [22,24,31], remote memory [46], fabric attached memory [18,32,47]. However, the concept of disaggregation has been extended to disaggregated architectures [7–17,19–21,23,25–28,33,34,48–50]. The main concepts of their systems are similar; they disaggregate hardware resources and interconnect them using high-performance interconnect technologies. However, the detailed designs and implementations differ depending on their goals and challenges. For example, Zombieland [8] offers a power-efficient memory model for a disaggregated system. This system uses an advanced configuration and power interface (ACPI) that manages and handles the power for each component. The authors suggested that if a server remains in an ACPI sleep state (i.e., zombie state), its memory alone could be active and remotely accessible to save the energy consumed by a disaggregated system. Gao et al. [19] use a special swap device to emulate *remote* memory and the main memory as *local* memory for deriving networking performance. Lim et al. [20] implement a disaggregated memory system prototype using a hypervisor. They analyzed the effects of page eviction algorithms in terms of performance and cost. Koh et al. [23] proposed a new RDMA-supported disaggregated memory system for large memory and a page management policy. They also used a hypervisor to support memory scalability on a virtual machine without the complexity and reduced the performance overhead.

3. Motivation and objectives

3.1. Prototyping of fabric-aware processor architectures

DisaggSim supports prototyping and performance evaluation of fabric-aware processor architectures. Disaggregated architecture that evolves around high-speed fabric interconnects often calls for processor architecture changes to bridge the in-system bus architecture with the fabric. For instance, the in-system bus packets must be translated to a fabric interconnect packet format. The translation must also translate the bus addresses to the fabric interconnect addresses to be routed to the correct resource on the fabric. Such processor architecture adaptations for the disaggregated clouds present many interesting challenges regarding performance and security as discussed in previous works [20,32,50].

3.2. Design and implementation of fabric interconnect architectures

We generalize and implement fabric interconnect and fabric-attached components. The specific design and implementation of the resource management schemes on the interconnect are another topic of interest for many researchers [18,20,50,51]. Also, the fabric-aware processor architecture, interconnect protocol and interconnect components often work accordingly. Therefore, detailed component-by-component modeling of the interconnect is imperative in disaggregated architecture simulation and is also a daunting challenge.

The interconnect composition and the role of each component are in line with the tentative standards [1,9] and existing academic works [11,32,50]. We define the components and implement them in the simulator in a modular way (e.g., as extensible gem5 C++ classes) such that they can be modified to reflect a given fabric interconnect specification.

3.3. Evaluation of disaggregated architecture scalability and performance characteristics

Our strategy in DisaggSim regarding simulation accuracy is to define and expose detailed configurable parameters for fabric interconnect and fabric-attached components. To the best of our knowledge, there is neither publicly available hardware implementation nor detailed hardware-level specifications for disaggregated architecture [24,33]. Also, their performances have only been shown through a rather limited scope [19]. Therefore, the goal of DisaggSim is to provide detailed and flexible simulation with customizable parameters, rather than to model a specific system.

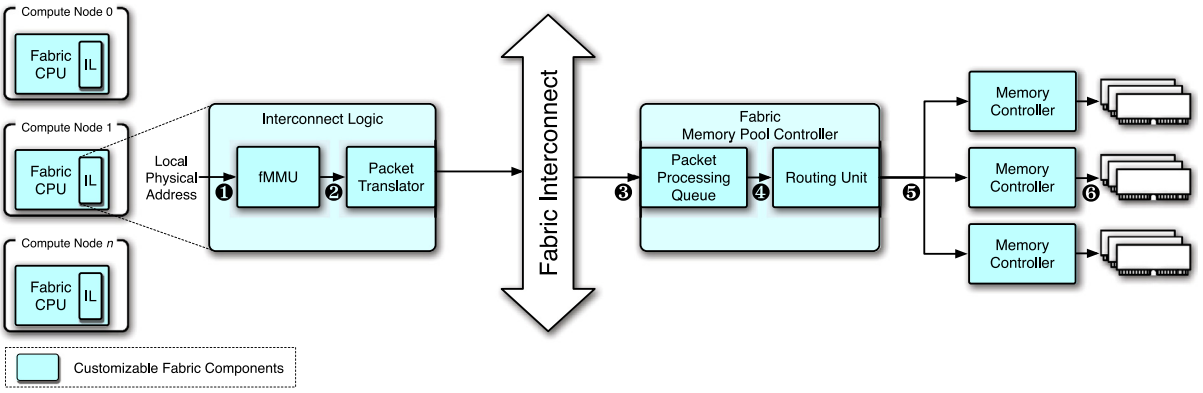


Fig. 1. Customizable fabric-attached components and the fabric interconnect in DisaggSim.

4. Disaggsim design

There is not yet a clear agreement on the best methods to design and implement this new architecture because the disaggregated architecture is a relatively new and rapidly developing field. Designs of the disaggregated architecture differ from conventional machine-oriented architecture approaches and vary depending on their particular use cases because they have different advantages, difficulties, and goals. We generalize the designs of existing disaggregated architectures by choosing their common elements through an extensive and rigorous study in this section. Our generalized design enables researchers to rapidly simulate their new systems by adjusting the configuration of design components.

4.1. Overall design of our generalized disaggregated architecture

Fig. 1 shows the overall structure of the generalized disaggregated system simulated in DisaggSim. The components labeled as *Customizable Fabric-attached Components* are new system components based on our generalization of the existing systems.

Terminology. We refer to the disaggregated components that perform computation as *compute nodes*. Compute nodes are capable of computing and directly attached to the fabric interconnect instead of a system bus or network-on-chip. These nodes can be added and removed on demand according to the required computation. A compute node can be a main processor or an accelerator such as a GPU. However, we focus on the processor in this work because an accelerator compute node can be considered a slight variant of a processor compute node (see Section 7.4).

The storage devices, such as memory, are no longer tied to a single machine but rather managed in a centralized way [1,7,9,10,12]. We call the fabric-wide memory management structure and collective set of memory the *fabric memory pool*.

We use the term *domain* to describe a group that virtually aggregates hardware resources to comprise a complete computing assembly that functions like a machine. Each domain consists of the following nodes: a compute node and a portion of the fabric memory space within one or more memory controllers.

Key Components in Disaggregation. We define the following key components that comprise a disaggregated architecture: *Interconnect Logics*, the *Fabric Interconnect*, and the *Fabric Memory Pool Controller (FabricMPC)*.

A disaggregated architecture must interface and manage the communication between the compute nodes and memory pool. It must enable the interoperation of possibly heterogeneous compute nodes and memory devices that are no longer directly interfaced with the processors' local memory controllers. Moreover, the fabric interconnect often speaks in its bus specification. Hence, additional adapters or logics (e.g., Gen-Z logic [1], STU [32]) must be added to the compute nodes to translate the compute node memory requests to the fabric memory requests. We call such logics *Interconnect Logics*. In the process, the *fabric MMU* (fMMU) translates the local physical addresses to fabric-wide addresses.

A disaggregated architecture must include a high-speed fabric interconnect whose bandwidth and performance characteristics vary depending on the material used and design. DisaggSim includes a fabric interconnect with a custom but highly configurable fabric protocol which all nodes in the architecture use.

In contrast to a machine-oriented architecture, where the CPU stores or loads data from adjacent memory modules, in disaggregated architecture, memory space is located at the remote site and directly attached to the fabric interconnect. Furthermore, unlike a compute node with computational functionality, memory controllers and modules have limited processing power and are designed to perform only memory I/O operations. DisaggSim features a memory component called FabricMPC to address the limitation of computing power and longer-distance memory access. Previous studies have similar components such as buffer pool memory [49], Gen-Z media controller [1], memory pool [17,52,53], programmable switch [48], and remote memory controller [29]. The primary role of FabricMPC is inter-component communication between the fabric interconnect and memory controllers. It controls the memory pool management hierarchy, including memory controllers and modules.

Rapid Prototyping of Disaggregated System Instances. With the generalized key components, DisaggSim allows rapid prototyping of disaggregated system instances by modifying configuration files and, if necessary, the source codes. Specifically, we provide configuration files that build the organization of primary components with various parameter values and the source code written modularly.

Overall Flow of Remote Memory Accessing. We explain the overall flow of remote memory access starting from a bus packet generated in a compute node to a memory module in the fabric memory pool (❶–❻ in Fig. 1). ❶ In a compute node, the CPU accesses a memory location and generates a bus packet that contains a local physical address. fMMU takes the local physical address as input and translates it to a global fabric address. ❷ Given the translated information, the packet translator extends the CPU packet to a fabric packet. ❸ This fabric packet then traverses the fabric interconnect and arrives at the Fabric memory pool. ❹ Packet processing queue in FabricMPC processes the fabric packet and forwards it to the routing unit. ❺ Routing unit routes the fabric packet to the corresponding memory controller. ❻ Finally, the memory controller determines the corresponding memory module using the fabric memory address.

```
struct Packet_Request {
    uint32 src_nid;
    uint32 dst_nid;
    uint32 memory_controller_index;
    uint64 fabric_addr;
    uint8  opcode;
    uint32 seq_num;
    uint32 crc;
};
```

Listing 1: The structure of a fabric packet

4.2. Interconnect modeling

Most machine-based architecture uses system buses as network topologies to communicate with a limited number of network entities. All the entities communicate through the shared interconnect in disaggregated architecture; therefore, its network topology must be efficient and scaled to support higher network traffic and low latencies. We model the interconnect as *off*-chip network that can afford such inter-component communication.

Interconnect Protocols for Fabric-Attached Components. The fabric interconnect requires node identifiers (NIDs) to determine which component should receive fabric packets. Therefore, FabricMPC assigns node identifiers to fabric-attached nodes. We also extend the existing CPU packet structure initially for machine-oriented architecture by adding source NID (SrcNID) and destination NID (DstNID) fields to traverse the fabric interconnect, as described in Listing 1.

Since FabricMPC maintains one or more memory controllers that are not directly attached to the fabric interconnect, we assign indexes managed locally in the memory pool to memory controllers. Moreover, we add a memory controller index field to the fabric packet structure for the routing unit in FabricMPC to send packets to corresponding memory controllers. If a packet is not bound for memory, this field is `-1`.

4.3. Compute node modeling

Most disaggregated systems adopt customized system-on-chip (SoC) as a design choice [1,2,9–11]. The SoC customization is essential because existing processors cannot directly connect to the interconnect, whereas disaggregated architecture requires hardware resources to be directly attached to the interconnect. Alternatively, an external device (e.g., an FPGA device) [7,54] can be used to bridge the processor and the fabric. We adopted the former model in which the commodity processor architectures are extended to include such logic. With this approach, the processors can be readily connected to the fabric to communicate the interconnect protocol.

4.4. Address and packet translation

Local Physical to Global Fabric Memory Address Translation. Many disaggregated systems such as Gen-Z [1], ThymesisFlow [7], and DeACT [32] employ *fabric address space* that maps the resources on the fabric. This additional memory space requires a management unit that resembles the Memory Management Unit (MMU) employed in modern commodity processors. Prior studies feature a single central MMU such as Rack MMU [55] and ZMMU [1], or multiple decentralized MMUs such as System Translation Unit (STU) [32] and Remote MMU (RMMU) [7]. We adopted the latter design of the MMU organization for fabric memory management since this design can reduce the bottleneck of centralism that occurs accessing the remote memory pool. We call such memory translation component in the interconnect logic fMMU (❶ in Fig. 1).

We only implement a rudimentary address translation in our current implementation because the address translation can vary depending on resource organizations and architectural design. However, hardware-based address translation schemes can be tested on our simulator by modifying the generalized disaggregated architecture.

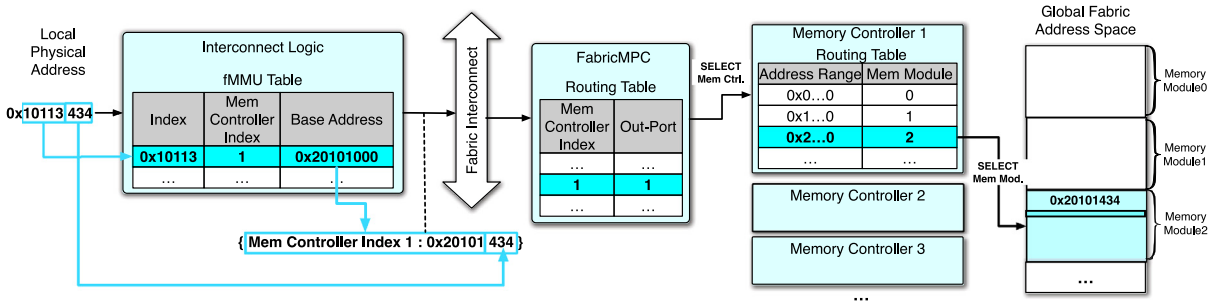


Fig. 2. Local physical to global fabric address translation and packet routing with the global fabric address.

Specifically, the fMMU component takes CPU packets as input and traverses the fMMU table with the base of a local physical address that is a result of virtual-to-physical address translation performed in the CPU. The fMMU then identifies the table entry that consists of the corresponding local-to-fabric mapping information (the format of the pair {memory controller index:base address}). For example, as depicted in Fig. 2, a local physical address `0x10113434` is translated to a fabric address `Mem Controller Index 1:0x20101434`. The translation is done by identifying the corresponding fMMU table entry with 0×10113 as a table index and adding the mapped base address 0×20101000 with the offset 0×434. Both unmodified cores in compute nodes and memory controllers in the memory pool locally manage their physical memory space. However, the compute node can access the exact location of the integrated remote memory pool using the global fabric memory address derived from the fMMU address translation.

4.5. Fabric memory pool management

The fabric memory pool consists of several *unmodified, non-fabric-aware* media and FabricMPC to control them. FabricMPC can accommodate DRAM and NVM (Non-Volatile Memory) modules. FabricMPC enables the memory pool establishment and management, as described on the right side of Fig. 1. The media are accessed through memory controllers connected to FabricMPC. Notably, we employ fully disaggregated memory that does not use local memory (see Section 7.6).

FabricMPC consists of a packet processing queue, routing unit, and memory controllers. The packet processing queue in FabricMPC receives fabric packets from the fabric interconnect, processes the packets, and forwards them to the routing unit (❹ in Fig. 1).

Routing Unit. Multiple compute nodes in domains can simultaneously send fabric packets containing node ID and memory controller index fields to the fabric memory pool via the fabric interconnect. When fabric packets are routed from compute nodes to FabricMPC using dstNID, FabricMPC routes the packets to the corresponding memory controllers using memory controller index fields in the packet headers and decapsulates the packets for non-fabric-aware memory controllers. The routing unit in FabricMPC maintains mappings of memory controller indexes and controller-side port numbers (❺ in Fig. 1). The memory controllers then receive the decapsulated original CPU packets that do not include node ID information.

Memory Controller. A memory controller is in charge of accessing media-specific storage structures, such as the DRAM rows and columns according to physical addresses included in the request packets (❸ in Fig. 1). The controller has read and write queues so that requests can be simultaneously serviced depending on the capacity of the queues. As long as the queues are not full, compute nodes do not have to resend packets, even if the controller is busy processing other requests.

Hardware Resource Management. Unlike machine-oriented architecture, hardware components in a disaggregated system require resource allocation information to determine where to send packets because resources are disaggregated and located at remote sites. Therefore, FabricMPC maintains a global resource allocation table containing mapping pairs of node IDs for compute nodes and memory pool nodes. In the initialization stage of DisaggSim, FabricMPC sends packets to all nodes to be initialized and assigns node IDs to all nodes, including compute nodes and the memory pool node. As a result, fabric-attached components can maintain the resource allocation information and route fabric-aware packets according to node IDs in the fabric packet headers.

Our generalized design targets a shared-nothing architecture (SN) as ThymesisFlow [7] and heterogeneous workload as Fastswap [24]. In this model, literally, each compute node does not share the same memory space in the single fabric memory pool. SN has a few advantages: First, SN terminates resource contention among compute nodes. Second, SN terminates single points of failure that can be introduced in memory-sharing disaggregated architecture [34,55]. Third, the system administrator can simply scale the number of hardware resources by adding nodes because the resources are independent of each other. Finally, this architecture allows a more straightforward design that avoids concurrency problems (e.g., race condition, deadlock, and bottleneck) and focuses on other design factors. Nonetheless, DisaggSim can be extended to model a shared-memory architecture.

Dynamic Memory Range Remapping. A memory controller has its own physical address space, which can be merged with address space provided by other memory controllers to offer a view of continuous memory to a compute node. For example, suppose that two memory controllers are equipped with 8 GB DRAM modules, respectively. Each memory controller has its own memory space from 0 to 8 GB - 1. However, in the compute-node view, the memory range is considered one continuous range from 0 to 16 GB - 1. When the compute node sends requests for memory access, FabricMPC uses a memory controller index that the

interconnect logic embeds into request packets to correctly route the memory requests between the two memory controllers and transparently supports the continuous range of memory space in a domain. Because the fMMU of the interconnect logic translates each request address into the address of the destination memory controller, the memory controller can process the request without extra translation.

5. Simulator implementation

Our simulator extends the gem5 simulator (version 20.1.0.2) since it is the most commonly used architectural simulator [56,57]. Also, many previous works have extended gem5 to enable simulation of new system components and system types [58–62], and such works have made tremendous contributions to the researchers in the field. The simulator can execute program instructions with a given system model at the cycle level. The simulated system is configured as a set of components and links that connect the components. The configuration is described in Python and parsed at runtime, while the components' internal behavior is implemented in C++.

The generalized components and their roles in the architecture are implemented into gem5 simulator objects (i.e., extensible C++ objects) that can be incorporated into the construction and evaluation of disaggregated architectures. Such objects become the foundation of the disaggregated architecture simulation. Specifically, we exploit the object-oriented design pattern such that the architecture-specific custom component classes inherit the general component classes. This design strategy allows the simulator to be adapted for prototyping and evaluating various disaggregated architectures with different design details. In this section, we describe further implementation details worth noting.

5.1. Multiple-system simulation

The gem5 simulator assumes a single machine-based system where memories and peripherals are attached to a CPU. Namely, if multiple CPUs are inserted into a system, the simulator automatically allocates hardware resources to the entire set of CPUs for a single system in the current gem5 implementation. Hence, individual hardware components must be segregated into separate subsystems to implement the concept of domain. Specifically, we configured each component to be a part of different systems, initially representing a single machine in the machine-oriented architecture.

Furthermore, gem5 grants a globally unique requester ID per each requester object (e.g., CPU core), where such an ID is used to locate the source of a gem5 packet. However, in the current gem5 implementation, requester IDs are managed within a system, which cannot manage globally unique IDs in DisaggSim due to the multiple-system modification described above. Thus, we configured the requester IDs to be unique in the entire system.

5.2. Fabric interconnect

We extended gem5's built-in network simulator, Garnet [63], to demonstrate the feasibility of a fabric interconnect. This network simulator is widely used with the exploitation of gem5 compatibility. Garnet provides cycle-level modeling with pipelined routers, virtual-channel flow control, and micro-architectural details.

We modified Garnet to support an off-chip network since the simulator is currently designed to simulate a network-on-chip environment. We then made changes to the interconnect simulator to support incorporating the multi-system into an one-memory-pool disaggregated system. Below are the main disaggregated-system-specific changes that are necessary for simulating disaggregated systems.

Interface. Interfaces create a connection between each component and the fabric interconnect. They consist of message buffers, ports, and features that support sending/receiving fabric packets and reliable transmission. We modified compute nodes and FabricMPC to connect fabric interconnect instead of the current gem5 environment where CPU cores are directly connected to memory and caches. Support for cache coherency protocol is an essential aspect of any interconnect design. However, at this point, cache coherency in fabric interconnects does not have enough tangible details in previous studies. Hence, we do not implement a cache coherency protocol in our current simulation implementation. We expect that our simulator can facilitate the prototyping of cache coherency for fabric interconnect (see Section 7.2).

Topology. The topology affects the behavior of the interconnect and, hence, influences the performance of the entire system. Garnet supports several topologies so that users can evaluate their network simulation with various configurations. In DisaggSim, we use a crossbar topology so that all components can be directly connected to the fabric interconnect with one hop, and simultaneously communicate with less contention. Nevertheless, DisaggSim is not restricted to any topology and can employ another topology by changing a simple parameter.

5.3. Compute node

A compute node is mainly comprised of a computing core, requiring an interconnect logic to connect itself to the fabric interconnect. Since the concept of interconnect logic does not exist in machine-oriented architecture, we implement the interconnect

Table 1
DisaggSim's configurable parameters and baseline values.

Component	Parameter	Baseline value
Fabric interconnect	Link width bits	288 bits/phit [66]
	Link latency	6.8 ns/packet [66]
	Transmission latency	10 ns/packet [67]
	Switch processing latency	40 ns/packet [67]
Compute node	Number of compute nodes	1
	Number of cores per compute node	1
	L1, L2, L3 cache size	32 KB, 256 KB, 2 MB
	Type of CPU	`DeriveO3CPU`
	Instruction set architecture	x86
	Address translation latency	60 ns/packet [50]
Fabric memory pool controller	Number of packet processing queues	1
	Routing latency	40 ns/packet
	Frontend latency	1 ns/packet[a]
	Forward latency	0.5 ns/packet[a]
	Response latency	1 ns/packet[a]
	Header latency	0.5 ns/packet[a]
Memory controller	Number of memory controllers per domain	2
	Write buffer threshold	50–85% [68]
	Frontend latency	10 ns/packet [68]
	Backend latency	10 ns/packet [68]
Memory module	Number of memory modules per memory controller	2
	Size of memory modules	256 MB

[a]Denotes values derived from gem5's default values.

logic that faces the interconnect into a compute node in gem5. The interconnect logic is connected to a compute node via a dummy memory interface that merely forwards memory packets to the other end to ensure all memory traffic from a compute node passes through interconnect logic.

Compute nodes in our simulator are mainly designed based on x86, which is a dominant architecture in data centers. However, given the recent rise in ARM architecture-based servers such as AWS Graviton2 [64] and RISC-V support for Gen-Z disaggregated system [65], we also support such architectures as compute nodes using a simple change of the ISA parameter, which is inherited by the nature of gem5.

5.4. Fabric memory pool controller

FabricMPC internally manages a port for each memory controller. At initialization time, a port is configured according to the gem5 configuration file that specifies domain information. When requests are sent by compute nodes, FabricMPC retrieves the memory controller index field from the requests' packet header and forwards the requests to the corresponding port.

Since a memory controller of FabricMPC can accommodate multiple DRAM and NVM modules, packet routing in memory controllers is necessary. Memory controllers compare each medium's assigned address range and the packets' physical addresses. Note that DstNIDs and memory controller indexes are not used in this step; non-fabric-aware DRAM and NVM media can be used without modification to support disaggregated systems.

We also implement read and write request queues serving as simple caches in memory controllers. Therefore, the memory controllers immediately handle consecutive memory accesses to identical physical addresses without accessing the memory location repeatedly.

5.5. Configurable parameters

DisaggSim supports various configuration parameters, which could define the behaviors of the generalized components of disaggregated architectures as described in Table 1. These parameters include different types of latency in a disaggregated architecture and the number of compute nodes and memory components. Researchers can conduct a range of experiments only by simply configuring them. Note that gem5 parameters can be configured by adding command-line options or editing configuration files.

6. Evaluation

In this section, we show the evaluation methods and results of DisaggSim. Our evaluation shows how our simulator can be utilized to model a disaggregated system and how a system would respond to different configurations and parameters, allowing one to estimate and deduce the source of performance bottlenecks and draw performance requirements for each module.

6.1. Experiment setup

We set baseline simulation settings as shown in Table 1, whose details are to be explained in the following subsection. Each compute node is set to be an `x86` architecture modeled with the `DeriveO3CPU` CPU model with L3 caches. All simulations were conducted on a machine with two AMD EPYC 7601 processors at 2.2 GHz and 512 GB of RAM and running a Ubuntu 18.04.5 system with kernel version v4.15.0. We used the system call emulation mode because we did not need to simulate an operating system for running application-level benchmark suites. This mode can avoid kernel-level noise and take advantage of faster performance in the slow, cycle-level gem5-based simulator.

Benchmark Tools. We employ two different benchmark tools: SPEC CPU2006 suite [39], which represents non-trivial, real-world workload, and STREAM benchmark suite [40] to examine the scalability of the number of compute nodes. In particular, we used the integer subset of SPEC CPU2006 (SPECint2006) and modified the STREAM benchmark tool to use integer data types to exclude floating-point computing.

We excluded four applications from SPECint2006 benchmark suite, one of which enters the endless loop (`mcf`) and three of which crashes with a segmentation fault (`gcc`, `omnetpp`, and `xalancbmk`). The `mcf` benchmark requires 1,700 MB of memory requirement due to extensive uses of longs and pointers [69]. Also, the working set size of `mcf` is 21.1 times higher than the average value of other benchmarks [70]. However, our baseline parameters configure each compute node to equip 1 GB of memory (2 memory controllers × 2 memory modules × 256 MB). Therefore, we believe the large memory footprint of `mcf` may cause 'never end' in a reasonable time.

Regarding the crashing benchmarks (`gcc`, `omnetpp`, and `xalancbmk`), the *unmodified* gem5 did not support the three crashed benchmark applications with the configuration of `DeriveO3CPU` in our feasibility test.[2] The `omnetpp` benchmark crashes because of a segmentation fault, whereas `gcc` and `xalancbmk` due to invalid pointer free. Although different reasons caused the errors, they were commonly thrown by an instruction decoder module in the x86 processor. Fixing Bugs from the vanilla gem5 is beyond the scope of this paper because we mainly focus on the delta (the codes that we have additionally implemented on the vanilla gem5) from gem5.

For the STREAM benchmark, we adjusted the value of the `STREAM_ARRAY_SIZE` parameter to be 1 M at compile time as the benchmark writers suggest that the array should be at least four times as large as the cache size. The array size of 1 M is sufficient for the requirement as the adjusted array is 8 MB (1 M entries × 8-byte integer) in total, and the L3 cache size in each compute node was configured to be 2 MB per core in DisaggSim.

The STREAM benchmark tool includes four different memory operations: `Copy`, `Scale`, `Add`, and `Triad`. `Copy` and `Scale` are 2-array operations, and `Add` and `Triad` are 3-array operations. Certainly, 3-array operations create more memory accesses than 2-array memory operations. All except `Copy` additionally perform simple arithmetic operations. The STREAM benchmark application performs all four memory operations as a batch task and chooses the best results from 10 trials. To measure the execution time of each memory operation, we separated the original STREAM benchmark applications into four individual applications with each memory operation.

6.2. Simulation parameters and default values

We simulated a generalized disaggregated system to demonstrate that the simulator can be employed to perform functional correctness tests and performance estimation. For the evaluation, we set reasonable default parameters values through a rigorous study of existing works [38,50,66–68] as shown in Table 1, which can be adjusted in a fine-grained manner to reflect the characteristics of the simulated system. We explain the parameter values and the reasoning for choosing the values as the default parameters for the generalized disaggregated system.

Packet Translation and Address Translation Latency at the Compute Node (L_1). Interconnect logic at a compute node performs packet translation and address translation. The packet translation is required to bridge different interconnect protocols, and the address translation is necessary to locate appropriate locations in the target remote memory. Such processing is configured to take 60 ns, according to [50].

Transmission Latency at the Interconnect (L_2). As the packet size is 80 bytes (16 bytes/flit×5 flits/data packet), one packet takes about 12 ns (640 bits/53.125 Gbps) to load to the link connected to the fabric interconnect (*transmission latency*). The transmission latency presented by Gao et al. [19] is 0.82 μs for 4 KB (16 ns for 80 bytes), similar to the experimental environment considering the different bandwidths.

Link Latency at the Interconnect (L_3). After loading is complete, the packet is transferred through the links to the fabric interconnect. Transferring a packet takes 6.8 ns of *link latency*. All transfers in the interconnect take the latency for each of them, including a router-to-router transfer. We assumed the bandwidth of an interconnect link to be 53.125 Gbps and the phit[3] of a link to be 288 bits, according to Gen-Z PHY [66]. As a result, the latency of a single lane link is 5.42 ns (288 bits/53.125 Gbps). We adopted one-fourth of link latency, at 1.36 ns/flit (6.8 ns/packet) for each link, as the links in our design have four virtual channels [66].

Switching Latency at the Interconnect (L_4). When the packet arrives in the fabric interconnect, it takes 40 ns [67] of *switching latency* for the packet to be routed to a router or node. A router reads the packet's header and passes it over to another router through

[2] We have confirmed that the official gem5 repository no longer maintains the status of SPEC CPU2006 [71].

[3] Physical digit (phit) is the amount of data transferred in a single cycle.

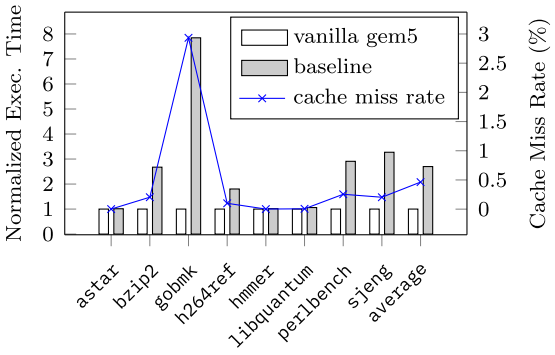


Fig. 3. Performance comparison between a single machine (vanilla gem5) and a single compute node in DisaggSim. Cache miss rate is identical for both systems. Performance overhead and cache miss rates are directly proportional, illustrating the impact of memory fetch latency in disaggregated systems.

a link until it reaches the end of the interconnect (out port). At the out port, the packet is loaded to a link and copied from the corresponding interface to the interconnect logic in FabricMPC. Our experiments used the default configuration of gem5 with the unlimited sending and receiving queues in the fabric interconnect. The value of the queuing delay is included in L_4.

Latencies at the Memory Pool (L_5, L_6, and L_7). The FabricMPC accepts the incoming packet from the interconnect logic and forwards it to an appropriate memory controller according to the memory controller index information of the packet. This routing in the memory pool subsystem takes 40 ns/packet to complete (*FabricMPC routing latency* (L_5)), which is the same delay as the switching latency of the interconnect router. A memory controller, identical to an unmodified DDR4 controller, executes low-level I/O operations. As memory controllers contain read/write queues for requests (see Section 4.5), some requests can be serviced directly by queues in the controllers without accessing physical memory modules. In this case, only the *frontend latency* (L_6) is applied; otherwise, *backend latency* (L_7) is also added. For evaluation, we defined three parameters (write buffer threshold, frontend latency, and backend latency) of memory controllers that affect system performance as default values according to [68].

Miscellaneous Latencies for Simulator Implementation (L_8–L_{11}). Additional small latencies are applied to the FabricMPC due to the packet routing implementation of gem5. These latencies include *FabricMPC frontend latency* (1 ns/packet, L_8), *FabricMPC forward latency* (0.5 ns/packet, L_9), *FabricMPC response latency* (1 ns/packet, L_{10}), and *FabricMPC header latency* (0.5 ns/packet, L_{11}). These are all default values of gem5 `IOXBar` class, which is a parent class of the FabricMPC object.

6.3. Performance comparison vs. Single machine system (Experiment 1)

Fig. 3 illustrates the results of SPECint2006 benchmark for comparison between the unmodified gem5-simulated machine (vanilla gem5) that represents the machine-oriented architecture and our simulated disaggregated system with parameters in Table 1 (baseline value). Each experimental result of benchmark applications is normalized to the vanilla gem5.

With the results of Experiment 1, we found that benchmark applications with a large number of total cache misses incurred significant overall performance overhead in the current design of generalized disaggregated architecture with realistic parameter values compared to the machine-oriented architecture. The benchmark application `gobmk` exhibits the highest variation in execution time (7.85x) and accesses the remote memory most frequently (2.93% of the complete cache miss rate). However, we observed that the `hmmer` and `libquantum` had minimal runtime variations and that their localities are high (0.0011% and 0.0058% of complete cache miss rate, respectively), which conform to the previous study of SPEC2006 analysis [72].

The experiment implies that memory access latency is the most dominant performance overhead in the disaggregated system. Our results show that the performance overhead is almost directly proportional to the cache miss rates. This indicates that the latency of fetching data from the fabric memory pool to the compute node processor caches is indeed the culprit of the observed performance overhead. Memory access latency comprises many different factors, such as the address translation latency in the interconnect logic, switch processing latency in the interconnect, and more. We expect future research efforts to propose and evaluate new methods for reducing memory access latency. For instance, the address translation cost can be significantly reduced through the use of fabric-wide address translation caching (e.g., FAM translation cache [32]), which is absent in our current modeling (see Section 7.3).

6.4. Number of compute nodes and performance in each node (Experiment 2)

Fig. 4 shows the performance within a *single compute node* when the different total number of compute nodes are present on the fabric interconnect. We investigated the performance impact of additional compute nodes on individual performance on each compute node. With all simulation parameters intact, we only increased the total number of compute nodes in steps up to 8, and evaluated compute node performance using SPECint2006 benchmark running in each node.

We observed gradual performance degradation as the number of compute nodes increased. With eight compute nodes, the overhead is noticeable, marking 8.59% in `gobmk`. The performance overhead will predictably further increase as the compute node number increases. We observe the same pattern of overhead as in Fig. 3; the same set of workloads (`bzip2`, `gobmk`, `perlbench`,

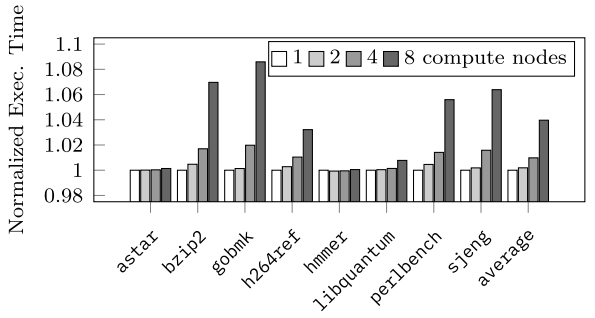


Fig. 4. Normalized execution time within a single compute node in the multiple-compute-node environment where the configuration input is the different number of compute nodes and the baseline configurations.

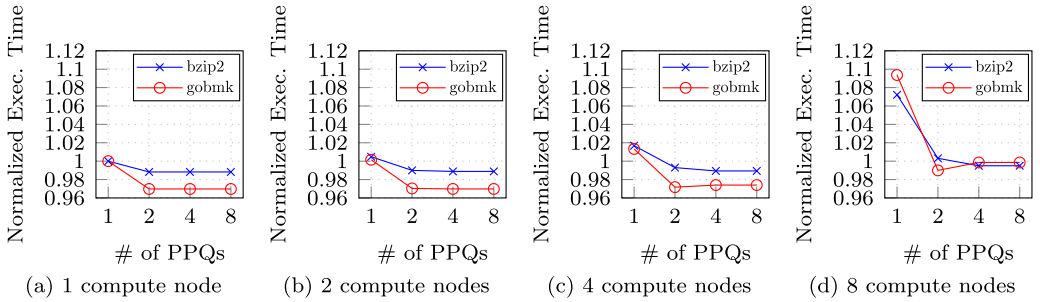


Fig. 5. The results of scalability and bottlenecks in the SPECint2006 benchmark experiments, where PPQ stands for packet processing queue. Note that we normalized the execution time with respect to the baseline configurations (where both the # of compute nodes and packet processing queues are 1) in Table 1 for comparison among experimental results.

and etc.) showed higher overhead. This indicates that the memory serving capacity of our system with baseline simulation parameters becomes a bottleneck with the increased number of compute nodes. To confirm this observation and demonstrate how one can estimate and scale the packet processing capacity of a disaggregated system to the target compute node number, we present the results of Experiment 3 in the coming subsection.

6.5. *Scaling packet processing capacity with many compute nodes (Experiment 3)*

We found the culprit of the performance degradation in many-compute-node scenarios to be the limited packet processing capacity in FabricMPC. We observed that the single packet processing queue is constantly filled, leaving a large number of packets on hold. This packet processing queue is in charge of processing fabric packets from the interface and sequentially transferring them to the packet multiplexer in FabricMPC.

Fig. 5 (SPECint2006) and Fig. 6 (STREAM) show the performance in each compute node with the number of *packet processing queues* in the FabricMPC. For many compute node scenarios, we additionally conducted the STREAM benchmark, accentuating memory access performance [40], whereas the SPEC benchmark showed general performance. To scale the packet processing capacity with the increased number of incoming packets, we experiment with adapting parallelism in the FabricMPC, a design pattern shown in some previous works [3,9–11]. We increased the number of packet processing queues in which the requests from the compute node accumulate in steps up to 64. The hardware packet processing logic in the FabricMPC is also duplicated so that the packet processing is performed in parallel.

SPEC Benchmark With Many Compute Nodes. With Fig. 5, we focus on the SPEC benchmark workloads that showed the most performance overhead in our previous experiments: `bzip2` and `gobmk`. We varied the number of compute nodes ({1, 2, 4, 8}) as well as the number of processing queues ({1, 2, 4, 8}). The results show that the increased number of processing queues released performance overhead in all cases. In the cases of multiple processing queues (i.e., all the compute node cases with the 2, 4, 8 queues), the average performance improvement was 3.2% and 5%, respectively in `bzip2` and `gobmk` benchmarks. Furthermore, we observed that duplicated processing queues were more effective in the cases with the larger number of compute nodes because the pressure caused by the memory accessing bottleneck became higher. Specifically, eight compute nodes with two processing queues significantly released the bottleneck effects by about 10% of system performance improvement.

STREAM Benchmark With Many Compute Nodes. STREAM benchmark [40] clearly shows the scalability of DisaggSim and bottlenecks incurred by the critical paths due to its simple memory operations measuring memory bandwidth. As shown in Fig. 6, we gradually scaled the number of compute nodes up to 32 with the baseline of the configurations to demonstrate those of the

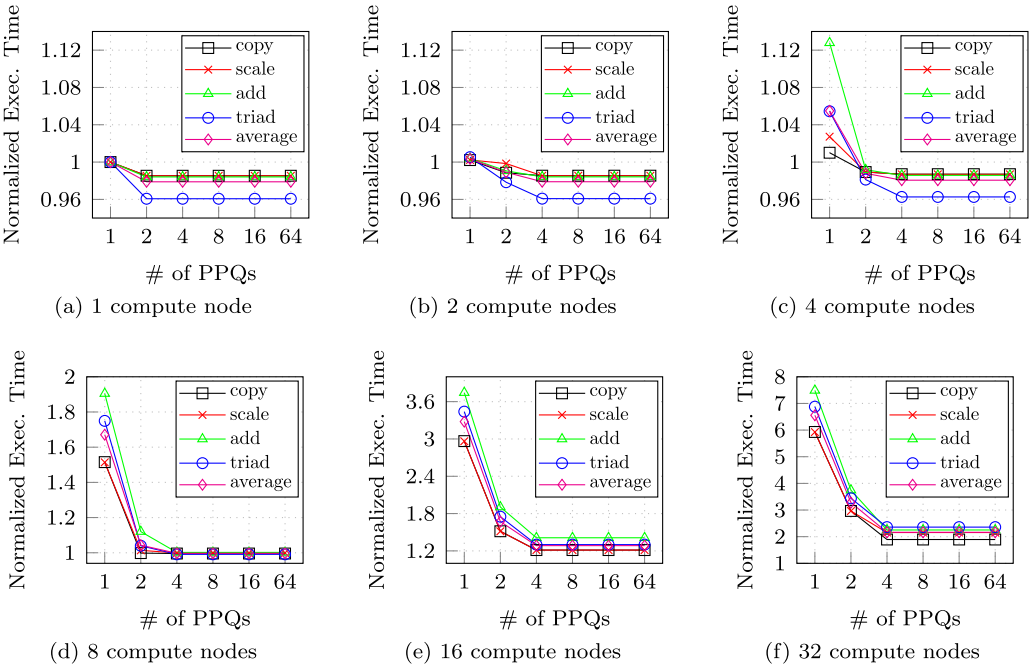


Fig. 6. The experimental results of scaled compute nodes and multiple processing queues (PPQs) in the STREAM benchmark. Also, it is notable that we normalized the execution time with respect to the baseline configurations (where both the # of compute nodes and packet processing queues are 1) in Table 1 for the comparison.

current data centers that represent 32 domains in a server. Each experiment in Fig. 6 was conducted with parameters with a pair of the different numbers of compute nodes and packet processing queues in FabricMPC.

Although our change to the number of packet processing queues is a simple example of avoiding the bottleneck effect, the experimental results demonstrate that this parallelism effectively released the bottlenecks as depicted in Fig. 5. In our experimental results, the execution environment of the increased number of compute nodes incurred additional overhead in a single compute node due to bottlenecks (see each subfigure in Fig. 6 where the number of packet processing queues is 1). In particular, for the 2-array benchmark experiments (`copy` and `scale`), the overhead of 8, 16, and 32 compute nodes was approximately x1.5, x3, and x6, respectively. Further, as expected, the 3-array operations (`add` and `triad`) incurred relatively higher overhead than the 2-array ones due to the increased number of remote memory accesses. The overhead trend of increasing compute nodes indicates that the doubled number of compute nodes causes more than doubled overhead. In other words, the more compute nodes run the benchmark tool, the more bottlenecks they create.

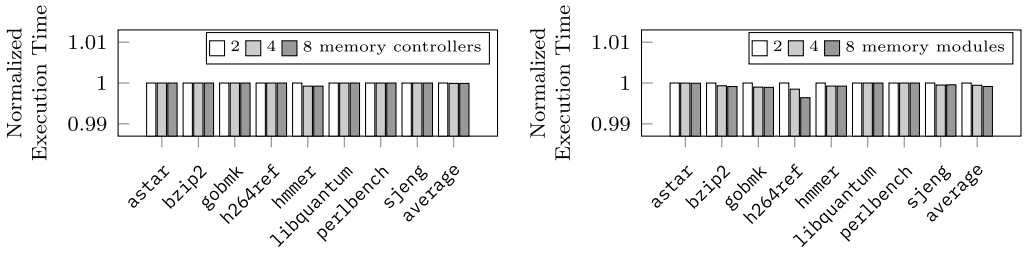
The effectiveness of the parallelism was higher in the environment that causes more remote memory accesses, such as the configuration of more compute nodes or the benchmark applications with 3-array memory operations. On the contrary, we observed that the effectiveness of the parallelism of packet processing queues is insignificant if the number of packet processing queues is sufficient to process incoming fabric packets. The results of experiments indicate the importance of employing multiple memory pools or multiple interfaces in a single memory pool for distributing memory accesses. Furthermore, system designers must be aware that any critical path or bottleneck hinders the scalability of each component in terms of system performance.

The experiments show that DisaggSim can be used to estimate the pressure on the fabric memory pool and consequently allow drawing the performance requirements for the FabricMPC given the approximate number of compute nodes. That is, our simulator can generate the packet traffic on the fabric according to the number and computing power of compute nodes. FabricMPC or an equivalent fabric packet processing component can be prototyped and evaluated using our simulator. As one strategy to scale FabricMPC with the increased influx of packets, we demonstrated the parallel packet processing queues mimicking similar methods attempted in previous works. We expect other methods for efficient fabric packet handling to be evaluated with DisaggSim.

6.6. Memory capacity and memory controllers components (Experiment 4)

We conduct experiments to understand the influence of memory capacity and the number of memory channels using memory-related parameters as illustrated in Fig. 7.

DisaggSim has two configurable options to adjust the number of memory components for each domain: the number of memory controllers and the number of memory modules per memory controller. We gradually increased the number of memory controllers (Fig. 7(a)) and memory modules (Fig. 7(b)). Notably, every domain was configured to use the same type (DDR4) and quantity of memory controllers in the evaluation.



(a) Normalized execution time with the different numbers of *memory controllers* and the baseline configurations

(b) Normalized execution time with the different numbers of *memory modules* and the baseline configurations

Fig. 7. Experimental results of capacity and parallelism on memory controllers and memory modules: the results show that the scaled number of memory controllers and memory modules do not effectively affect the system performance.

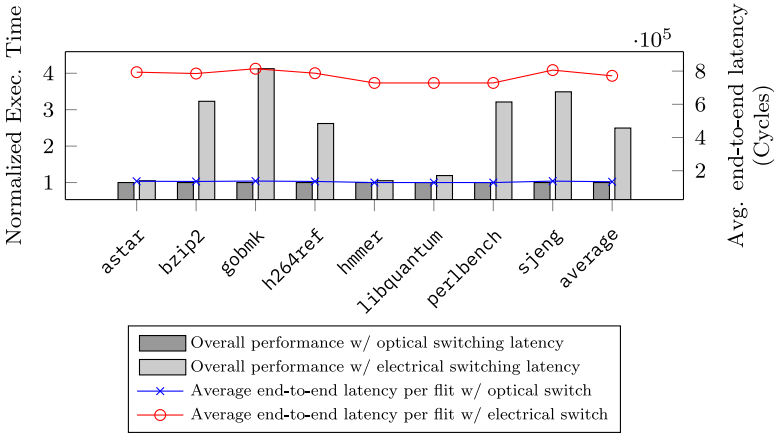


Fig. 8. Performance and latency comparison of different materials of the fabric interconnect.

In the results, leveraging multiple memory controllers did not reduce the average memory access time, although they provided a domain with multiple data channels. Fig. 7(a) depicts the effect of increasing the number of memory controllers per domain. System performance was not affected or was negligibly (less than 0.1%) improved even if each compute node used more memory controllers and memory capacity.

Similarly, the performance improvement was negligible (less than 1%) when each compute node was assigned more remote memory capacity by increasing the number of memory modules with the fixed number of memory controllers (Fig. 7(b)). More memory accessing channels and extra memory capacity did not affect the system performance because any configurations in our experiments met all benchmark memory requirements.

We conclude that memory scalability has a minimal change in performance. In other words, the system does not suffer from performance degradation once the memory capacity meets the requirements. Furthermore, the performance improvement may be insignificant even if unnecessary extra memory is allocated.

6.7. Interconnect latency (Experiment 5)

The performance of fabric interconnect has been a significant consideration in previous disaggregated system prototypes, because the interconnect fabric is a critical path for every compute node to access remote memory, as we discussed in Section 2.1. Especially, the material of an interconnect switch is a severe factor that influences the end-to-end latency and the entire performance of a disaggregated system.

We conducted experiments to understand the system performance with the different switching latencies of the fabric interconnect depending on the material (an electrical/optical interconnect) as described in Fig. 8: an optical switch with 40 ns [67] of latency and an electrical switch with 240 ns [19] of latency. Using the electrical switch increased about x5.78 the end-to-end latency based on the average end-to-end latency value in the optical versus electrical switch. We also observed x2.5 system performance degradation with the electrical switch compared to the optical switch.

The experimental results clearly show how the disaggregated system as a whole responds to the latency of the fabric interconnect using DisaggSim. This indicates that researchers can predict and select the fabric material according to the scalability and

performance requirements of the modeled system. Along with choosing the optimal fabric interconnect, researchers can estimate the required processing capabilities of the packet processing component (FabricMPC); as shown in Fig. 6, packet processing components must scale with the latency of the fabric material to avoid becoming the bottleneck of the system.

7. Future directions

We discuss the possible future directions for DisaggSim in this section. However, these are not just our future work as we make DisaggSim publicly available in the hopes of seeing contributions from other researchers in the field.

7.1. Access control

Fabric-wide access control schemes and implementations would be an interesting addition to DisaggSim. In the current commodity processor architectures, the processors either allow or disallow access to system resources (e.g., addresses). In contrast, a central authority must exist in disaggregated architectures to mediate the compute node access to the resources. A recent work [32] has discussed access control in disaggregated systems. We expect more work to be followed, and DisaggSim to be utilized to evaluate new access control schemes for disaggregated systems.

7.2. Cache coherence protocol

Cache coherence can be a performance bottleneck as the number of compute nodes increases in most disaggregated architectures. Therefore, previous studies partially or entirely limit the cache coherence in their systems. Specifically, LegoOS [18] does not use coherent caches to efficiently use their interconnects. Firebox [11] and the simulator introduced by [19] have restricted the cache coherence domain to one CPU or SoC. The Gen-Z specification [1] leaves the vendor-specific coherency interconnect undecided and argues that implementation of Gen-Z can avoid crossing the coherency interconnect to minimize the performance penalties. ThymesisFlow [7] targets data center workloads where applications share nothing or make heavy use of in-memory processing. Memory Blades [12] modifies a processor to use special hardware, the coherence filter. This filter redirects the cache-fill requests to the remote memory and strains unnecessary coherence protocol requests to reduce the cache-coherence traffic. We do not include a cache-coherence protocol in our current simulator for the same reason; there is no tangible fabric cache-coherence implementation to model in our simulator. However, our simulator can be leveraged to prototype and evaluate new protocols.

7.3. Reducing the memory access latency

Memory access latency is the most dominant performance overhead in the disaggregated architecture, as we demonstrated in Experiment 1 (Section 6.3). To reduce the memory access latency, Fabric address translation caching featured in DeACT [32] can be employed. A traditional processor's MMU caches page table entries from the page table and stores them in Translation Lookaside Buffer (TLB) to reduce the processing time to translate physical addresses to virtual addresses. Likewise, caching the local physical address to global fabric address translation can optimize the memory access latency in the processing time of fabric memory translation, depending on implementation. In the experimental results, DeACT achieved 95.88% of FAM address translation hit rate.

Furthermore, [19] suggested several solutions to address the memory access latency for each layer to meet minimum latency requirements (3 to 5 μs). The suggestion includes pFabric and pHost for efficient transport protocol, RDMA to bypass the packet processing in the kernel, and CPU-NIC (Network Interface Card) integration to reduce data copies to/from NIC.

7.4. Support for accelerator

As cloud providers recently offer machine learning [73–75] and gaming services [76,77], the demand for accelerators is growing rapidly. However, adding more accelerators suffers from the same limitation (e.g., PCIe slots) as memory devices in machine-oriented architecture. For example, Google Cloud AI service allows users to use a maximum of 16 GPUs. A disaggregated architecture can be a solution that supports a larger number of accelerators to process data-intensive workloads.

We plan to exploit the gem5 feature of Accelerated Processing Unit (APU) [78], which integrates CPU and GPU and shares unified memory space. It implements heterogeneous system architecture and simulates Graphics Core Next 3 (GCN3) ISA with adjustable configurations, designs, and cache hierarchies. The APU can replace the CPU in compute node in our current implementation so that the compute node can execute both x86 and GCN3 binaries.

The other option for implementing accelerators in the gem5 simulator is using discrete GPUs. The recent version of gem5 (v22.0) was introduced with discrete GPU support [79,80]. This option needs additional work to aggregate discrete GPUs into a GPU pool and route fabric packets to the corresponding GPU, as we did on FabricMPC.

7.5. Support for ISA heterogeneity

We plan to investigate the feasibility of multiple ISA-supporting simulating environments where, for example, an x86 processor-equipped compute node and an ARM processor-equipped compute node connected to the fabric interconnect simultaneously run. This future work will take complex considerations regarding different ISAs, cache and memory hierarchies, CPU topologies, and core frequencies into account. The recent version of Garnet [81] supports accurate interconnect modeling in heterogeneous systems. This new feature can assist our future work on the heterogeneity of compute nodes.

7.6. Local memory design considerations

We generalized fully disaggregated systems that many existing disaggregated systems that do not use per-compute-node local memory [9,10,26,37,82]. However, memory operations for fabric-attached remote memory incur more latency and degradation of overall system performance, because of the additional address translation and longer round-trip distance between processors and fabric-attached memory. A possible design choice to address such problems is to employ local memory within a compute node, which is called partially disaggregated memory [3,12,29,83]. Gao et al. [19] demonstrated that performance degradation is not critical (less than 10%) if a relatively small amount of local memory is provided in a disaggregated system with existing technologies for typical server workloads.

However, partially disaggregated memory requires software stacks (hypervisors, OSes, or applications depending on implementation) and compilers to be aware of the presence of local memory. The software can decide on either local or fabric-attached memory to load/store data depending on the data size and the frequency of data usage [55]. Despite the performance improvement, legacy applications that are hard to be modified to use such system calls may not exploit partial memory disaggregation.

Such use of the local memory leads to an inflexible system configuration, which the machine-oriented architecture suffers. Full resource disaggregation is essential for ideal future-generation data centers to avoid such a problem. We expect that improved interconnect technologies will make the ultimate resource disaggregation feasible and practical.

7.7. Simulator validation

A real machine and detailed performance specifications for disaggregated architecture are publicly unavailable to the best of our knowledge [24,33]. To address the lack of hardware support, many researchers simulate or emulate their ideas in the field (e.g., SST simulator [32], soNUMA emulation platform [29], ZSIM simulator [30], and other simulating environments [19,20,33]). Therefore, our work is a best-effort approach toward achieving accurate simulation through configurable and detailed parameters. We plan to validate our simulator and recalibrate the parameters once the specifications of real hardware become available. We expect the performance measurement to be nearly cycle-accurate [84] as long as the parameters that describe the architecture closely model the real system as previous gem5-based simulations presented [60,62,78,85].

8. Conclusion

In this paper, we presented the design, implementation, and evaluation of a simulator for fully disaggregated systems. We generalize the common components and characteristics of the fully disaggregated systems through a thorough study. Based on this generalization, we implemented the main components and specified fine-grained simulation parameters that can be adjusted to simulate a given disaggregated system accurately. This simulator supports the rapid prototyping of the main component and performing simulations to test functional correctness and acquire accurate performance estimations. The evaluation demonstrated a simulation of a general disaggregated system and proved that this simulator could be leveraged to model a disaggregated system accurately.

Acknowledgements

This work was supported by the following programs: Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00666, Research on Security of 5G-data-intensive Computing Platforms Based on Disaggregated Architecture), the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1A2C2101134, No. NRF-2022R1C1C1010494) and (No. 2022-0-01199, Graduate School of Convergence Security (SungKyunKwanUniversity)).

Data availability

Data will be made available on request.

References

[1] Gen-Z Consortium, et al., Gen-Z core specification 1.1, 2018, https://genzconsortium.org/. (Accessed 3 January 2023).
[2] Intel, White paper: Intel Rack Scale Design Architecture, Technical Report.
[3] Hewlett Packard Enterprise, The machine: A new kind of computer, 2020, https://www.hpl.hp.com/research/systems-research/themachine. (Accessed 3 January 2023).
[4] Facebook Engineering, Disaggregate: Networking recap, 2020, https://engineering.fb.com/2017/01/30/data-center-engineering/disaggregate-networking-recap/. (Accessed 3 January 2023).
[5] Huawei, Huawei reveals a highly efficient big data appliance prototype, NUWA, at Spark Summit 2015, 2015, https://www.huawei.com/en/news/2015/06/hw_441402. (Accessed 3 January 2023).
[6] Supercomputing (SC'22), Compute express link™: The breakthrough CPU-to-device interconnect, 2022, https://www.computeexpresslink.org/sc-22. (Accessed 3 January 2023).
[7] C. Pinto, D. Syrivelis, M. Gazzetti, P. Koutsovasilis, A. Reale, K. Katrinis, H. P. Hofstee, ThymesisFlow: A software-defined, HW/SW co-designed interconnect stack for rack-scale memory disaggregation, in: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, 2020, pp. 868–880.

[8] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, Daniel Hagimont, Welcome to Zombieland: Practical and energy-efficient memory disaggregation in a datacenter, in: Proceedings of the Thirteenth EuroSys Conference, EuroSys '18, Association for Computing Machinery, New York, NY, USA, 2018.

[9] M. Bielski, I. Syrigos, K. Katrinis, D. Syrivelis, A. Reale, D. Theodoropoulos, N. Alachiotis, D. Pnevmatikatos, E. H. Pap, G. Zervas, V. Mishra, A. Saljoghei, A. Rigo, J. F. Zazo, S. Lopez-Buedo, M. Torrents, F. Zyulkyarov, M. Enrico, O. G. de Dios, DReDBox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter, in: 2018 Design, Automation Test in Europe Conference Exhibition, DATE, 2018, pp. 1093–1098.

[10] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, T. Berends, Rack-scale disaggregated cloud data centers: The dReDBox project vision, in: 2016 Design, Automation Test in Europe Conference Exhibition, DATE, 2016, pp. 690–695.

[11] Krste Asanović, FireBox: A hardware building block for 2020 warehouse-scale computers, in: 12th USENIX Conference on File and Storage Technologies, FAST 14, USENIX Association, Santa Clara, CA, 2014.

[12] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, Thomas F. Wenisch, Disaggregated memory for expansion and sharing in blade servers, in: ISCA '09, Association for Computing Machinery, New York, NY, USA, 2009.

[13] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, Yiying 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 '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 417–433.

[14] Taekyung Heo, Seunghyo Kang, Sanghyeon Lee, Soojin Hwang, Jaehyuk Huh, Hardware-assisted trusted memory disaggregation for secure far memory, 2021, arXiv preprint arXiv:2108.11507.

[15] Liu Ke, Xuan Zhang, Benjamin Lee, G Edward Suh, Hsien-Hsin S Lee, DisaggRec: Architecting disaggregated systems for large-scale personalized recommendation, 2022, arXiv preprint arXiv:2212.00939.

[16] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, Myoungsoo Jung, Direct access,{high – performance} memory disaggregation with {directcxl}, in: 2022 USENIX Annual Technical Conference, USENIX ATC 22, 2022, pp. 287–294.

[17] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, Hakim Weatherspoon, Shoal: A network architecture for disaggregated racks, in: 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 19, 2019, pp. 255–270.

[18] Yizhou Shan, Yutong Huang, Yilun Chen, Yiying Zhang, Legoos: A disseminated, distributed OS for hardware resource disaggregation, in: 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 18, USENIX Association, Carlsbad, CA, 2018, pp. 69–87.

[19] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, Scott Shenker, Network requirements for resource disaggregation, in: 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 16, USENIX Association, Savannah, GA, 2016, pp. 249–264.

[20] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, T. F. Wenisch, System-level implications of disaggregated memory, in: IEEE International Symposium on High-Performance Comp Architecture, 2012, pp. 1–12.

[21] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, Kang G. Shin, Efficient memory disaggregation with infiniswap, in: 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 17, USENIX Association, Boston, MA, 2017, pp. 649–667.

[22] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, et al., Software-defined far memory in warehouse-scale computers, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 317–330.

[23] K. Koh, K. Kim, S. Jeon, J. Huh, Disaggregated cloud memory with elastic block management, IEEE Trans. Comput. 68 (1) (2019) 39–52.

[24] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, Scott Shenker, Can far memory improve job throughput? in: Proceedings of the Fifteenth European Conference on Computer Systems, EuroSys '20, Association for Computing Machinery, New York, NY, USA, 2020.

[25] Hasan Al Maruf, Mosharaf Chowdhury, Effectively prefetching remote memory with leap, in: 2020 USENIX Annual Technical Conference, USENIX ATC 20, 2020, pp. 843–857.

[26] Blake Caldwell, Sepideh Goodarzy, Sangtae Ha, Richard Han, Eric Keller, Eric Rozner, Youngbin Im, Fluidmem: Full, flexible, and fast memory disaggregation for the cloud, in: 2020 IEEE 40th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2020, pp. 665–677.

[27] Wenqi Cao, Ling Liu, Hierarchical orchestration of disaggregated memory, IEEE Trans. Comput. 69 (6) (2020) 844–855.

[28] Youngmoon Lee, Hasan Al Maruf, Mosharaf Chowdhury, Asaf Cidon, Kang G Shin, Mitigating the performance-efficiency tradeoff in resilient memory disaggregation, 2019, arXiv preprint arXiv:1910.09727.

[29] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, Boris Grot, Scale-out NUMA, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 3–18.

[30] Jorge Gonzalez, Mauricio G Palma, Maarten Hattink, Ruth Rubio-Noriega, Lois Orosa, Onur Mutlu, Keren Bergman, Rodolfo Azevedo, Optically connected memory for disaggregated data centers, J. Parallel Distrib. Comput. 163 (2022) 300–312.

[31] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K Aguilera, Adam Belay, {AIFM}:{High – Performance},{Application – Integrated} Far memory, in: 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 20, 2020, pp. 315–332.

[32] Vamsee Reddy Kommareddy, Clayton Hughes, Simon David Hammond, Amro Awad, DeACT: Architecture-aware virtual memory support for fabric attached memory systems, in: 2021 IEEE International Symposium on High-Performance Computer Architecture, HPCA, IEEE, 2021, pp. 453–466.

[33] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, Guoqing Harry Xu, Semeru: A memory-disaggregated managed runtime, in: 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 20, USENIX Association, 2020, pp. 261–280.

[34] Irina Calciu, M Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, Aasheesh Kolli, Rethinking software runtimes for disaggregated memory, in: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021, pp. 79–92.

[35] Sharma Debendra, Agarwal Ishwar, Compute express link 3.0, 2022, https://www.computeexpresslink.org/_files/ugd/0c1418_a8713008916044ae9604405d10a7773b.pdf. (Accessed 3 January 2023).

[36] NVIDIA Inc., InfiniBand networking solutions, 2020, https://www.nvidia.com/en-us/networking/products/infiniband/. (Accessed 3 January 2023).

[37] Pramod Subba Rao, George Porter, Is memory disaggregation feasible? A case study with spark SQL, in: 2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS, IEEE, 2016, pp. 75–80.

[38] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, David A. Wood, The Gem5 simulator, SIGARCH Comput. Archit. News 39 (2) (2011) 1–7.

[39] John L. Henning, SPEC CPU2006 benchmark descriptions, ACM SIGARCH Comput. Archit. News 34 (4) (2006) 1–17.

[40] John D. McCalpin, et al., Memory bandwidth and machine balance in current high performance computers, IEEE Comput. Soc. Tech. Committee Comput. Archit. (TCCA) Newslett. 2 (19–25) (1995).

[41] Mikhail Haurylau, Sharon M. Weiss, Philippe M. Fauchet, Dynamically tunable 1D and 2D photonic bandgap structures for optical interconnect applications, in: Tuning the Optical Response of Photonic Bandgap Structures, vol. 5511, International Society for Optics and Photonics, 2004, pp. 38–49.

[42] Paul Teich, RESEARCH PAPER: AMD's disaggregated servers, 2016, https://moorinsightsstrategy.com/white-paper-amds-disaggregated-servers/. (Accessed 3 January 2023).
[43] A.M.D. SeaMicro, SeaMicro SM15000 fabric compute systems, 2021, https://www.digchip.com/datasheets/parts/datasheet/1539/SM15000-pdf.php. (Accessed 3 January 2023).
[44] openCAPI consortium, openCAPI, 2020, https://opencapi.org/. (Accessed 3 January 2023).
[45] dRedBox, DRedBox's official web page, 2021, http://www.dredbox.eu. (Accessed 3 January 2023).
[46] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, Orion Hodson, {FaRM}: Fast remote memory, in: 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 14, 2014, pp. 401–414.
[47] Vamsee Reddy Kommareddy, Jagadish Kotra, Clayton Hughes, Simon David Hammond, Amro Awad, PreFAM: Understanding the impact of prefetching in fabric-attached memory architectures, in: The International Symposium on Memory Systems, in: MEMSYS 2020, Association for Computing Machinery, New York, NY, USA, 2021, pp. 323–334.
[48] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, Abhishek Bhattacharjee, Mind: In-network memory management for disaggregated data centers, in: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, 2021, pp. 488–504.
[49] Dario Korolija, Dimitrios Koutsoukos, Kimberly Keeton, Konstantin Taranov, Dejan Milojičić, Gustavo Alonso, Farview: Disaggregated memory with operator off-loading for database engines, 2021, arXiv preprint arXiv:2106.07102.
[50] Kevin T. Lim, Jichuan Chang, Trevor N. Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, Thomas F. Wenisch, Disaggregated memory for expansion and sharing in blade servers, in: Stephen W. Keckler, Luiz André Barroso (Eds.), 36th International Symposium on Computer Architecture, ISCA 2009, June 20-24, 2009, Austin, TX, USA, ACM, 2009, pp. 267–278.
[51] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novaković, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, Michael Wei, Remote regions: A simple abstraction for remote memory, in: 2018 USENIX Annual Technical Conference, USENIX ATC 18, USENIX Association, Boston, MA, 2018, pp. 775–787.
[52] Vaibhawa Mishra, Joshua L. Benjamin, Georgios Zervas, MONet: Heterogeneous memory over optical network for large-scale data center resource disaggregation, J. Opt. Commun. Netw. 13 (5) (2021) 126–139.
[53] Pengfei Zuo, Jiazhao Sun, Liu Yang, Shuangwu Zhang, Yu Hua, One-sided {RDMA – Conscious} extendible hashing for disaggregated memory, in: 2021 USENIX Annual Technical Conference, USENIX ATC 21, 2021, pp. 15–29.
[54] Seokbin Hong, Won-Ok Kwon, Myeong-Hoon Oh, Hardware implementation and analysis of Gen-Z protocol for memory-centric architecture, IEEE Access 8 (2020) 127244–127253.
[55] Sebastian Angel, Mihir Nanavati, Siddhartha Sen, Disaggregation and the application, in: 12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 20, USENIX Association, 2020.
[56] Jason Lowe-Power, et al., The gem5 simulator: Version 20.0+, 2020.
[57] gem5: Publications, 2021, https://www.gem5.org/publications/. (Accessed 3 January 2023).
[58] Yoongu Kim, Weikun Yang, Onur Mutlu, Ramulator: A fast and extensible DRAM simulator, IEEE Comput. Archit. Lett. 15 (1) (2016) 45–49.
[59] Alian Mohammad, Umur Darbaz, Gabor Dozsa, Stephan Diestelhorst, Daehoon Kim, Nam Sung Kim, Dist-gem5: Distributed simulation of computer clusters, in: 2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2017, pp. 153–162.
[60] Mohammad Alian, Daehoon Kim, Nam Sung Kim, Pd-gem5: Simulation infrastructure for parallel/distributed computer systems, IEEE Comput. Archit. Lett. 15 (1) (2015) 41–44.
[61] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, Timothy G. Rogers, Accel-Sim: An extensible simulation framework for validated GPU modeling, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, ISCA, 2020, pp. 473–486.
[62] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, Bruce L. Jacob, DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator, IEEE Comput. Archit. Lett. 19 (2) (2020) 110–113.
[63] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, Niraj K Jha, GARNET: A detailed on-chip network model inside a full-system simulator, in: Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on, IEEE, 2009, pp. 33–42.
[64] Amazon Web Services, AWS graviton processor, 2021, https://aws.amazon.com/ec2/graviton/?nc1=h_ls. (Accessed 9 February 2022).
[65] Abner Chang, Mohan Parthasarathy, Michael Ruan, Accelerating innovation using RISC-V and Gen-Z, 2021, https://genzconsortium.org/wp-content/uploads/2019/04/Accelerating-Innovation-Using-RISC-V-and-Gen-Z_V1.pdf.
[66] Karl Bois, Chris Brueggen, Elene Chobanyan, Michael Krause, Pete Maroni, James Regan, Highly-resilient 50G Gen-Z PHY and channel design, 2021, https://genzconsortium.org/wp-content/uploads/2019/07/Highly-Resilient-50G-Gen-Z-PHY-and-Channel-Design.pdf.
[67] Dan Alistarh, Hitesh Ballani, Paolo Costa, Adam C. Funnell, Joshua Benjamin, Philip M. Watts, Benn Thomsen, A high-radix, low-latency optical switch for data centers, in: Steve Uhlig, Olaf Maennel, Brad Karp, Jitendra Padhye (Eds.), Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015, ACM, 2015, pp. 367–368.
[68] Andreas Hansson, Neha Agarwal, Aasheesh Kolli, Thomas Wenisch, Aniruddha N. Udipi, Simulating DRAM controllers for future system architecture exploration, in: 2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2014, pp. 201–210.
[69] Löbel Andreas, SPEC CPU2006 benchmark description, 2011, https://www.spec.org/cpu2006/Docs/429.mcf.html. (Accessed 3 January 2023).
[70] Darryl Gove, CPU2006 working set size, SIGARCH Comput. Archit. News 35 (2007) 90–96.
[71] R. Bobby Bruce, The official gem5 repository, 2022, https://github.com/gem5/gem5-resources/commit/af5e91cbee324ed6a0a3cfc9c34ebfff41778170. (Accessed 3 January 2023).
[72] Kathlene Hurt, Eugene John, Analysis of memory sensitive SPEC CPU2006 integer benchmarks for big data benchmarking, in: Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, PABS '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 11–16.
[73] Amazon, AWS AI services, 2023, https://aws.amazon.com/machine-learning/ai-services/. (Accessed 3 January 2023).
[74] Google, Google cloud AI service, 2023, https://cloud.google.com/products/ai. (Accessed 3 January 2023).
[75] Microsoft, Microsoft azure AI, 2023, https://azure.microsoft.com/en-us/solutions/ai/. (Accessed 3 January 2023).
[76] Sony, PlayStation now, 2023, https://www.playstation.com/en-us/ps-now/. (Accessed 3 January 2023).
[77] NVIDIA, Geforce now, 2023, https://www.nvidia.com/en-us/geforce-now/. (Accessed 3 January 2023).
[78] Gutierrez Tony, Puthoor Sooraj, Ta Tuan, Sinclair Matt, Beckmann Brad, The AMD gem5 APU simulator: Modeling GPUs using the machine ISA, 2018, http://old.gem5.org/wiki/images/1/19/AMD_gem5_APU_simulator_isca_2018_gem5_wiki.pdf. (Accessed 3 January 2023).
[79] Poremba Mattew, Dutu Alexandru, Jain Gaurav, Fotouhi Pouya, Boyer Michael, Bradford M. Beckmann, Towards full-system discrete GPU simulation, 2020, https://www.gem5.org/2020/06/01/towards-full.html. (Accessed 3 January 2023).
[80] Lowe-Power Jason, gem5-22.0 released, 2022, https://www.gem5.org/project/2022/06/18/gem5-22-0.html. (Accessed 3 January 2023).
[81] S. Bharadwaj, J. Yin, B. Beckmann, T. Krishna, Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling, in: 2020 57th ACM/IEEE Design Automation Conference, DAC, 2020, pp. 1–6.
[82] D. Syrivelis, A. Reale, K. Katrinis, I. Syrigos, M. Bielski, D. Theodoropoulos, D. N. Pnevmatikatos, G. Zervas, A software-defined architecture and prototype for disaggregated memory rack scale systems, in: 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS, 2017, pp. 300–307.
[83] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, Yungang Bao, Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces, in: 2019 IEEE/ACM 27th International Symposium on Quality of Service, IWQoS, IEEE, 2019, pp. 1–10.

[84] Anastasiia Butko, Rafael Garibotti, Luciano Ost, Gilles Sassatelli, Accuracy evaluation of GEM5 simulator system, in: 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, ReCoSoC, 2012, pp. 1–7.
[85] Alian Mohammad, Umur Darbaz, Gabor Dozsa, Stephan Diestelhorst, Daehoon Kim, Nam Sung Kim, Dist-gem5: Distributed simulation of computer clusters, in: 2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, IEEE, 2017, pp. 153–162.