CXL-ClusterSim: Modeling CXL-based Disaggregated Memory Cluster for Pooling and Sharing using gem5 and SST

Kaustav Goswami, Maryam Babaie, Hoa Nguyen, Venkatesh Akella, and Jason Lowe-Power

University of California, Davis

Abstract

Large-scale AI training and inference require hundreds of gigabytes to terabytes of DRAM with high peak to average utilization ratios, resulting in overprovisioning. In cloud computing, DRAM constitutes a significant share of the cost. Yet, as shown by recent articles, DRAM is heavily under utilized. Memory disaggregation is a solution to both these problems. With the advent of the CXL protocol, there is renewed interest in designing and optimizing computing systems with disaggregated memory. However, at present, there are limited simulation tools available for exploring the design space and evaluating the performance tradeoffs in computer systems with disaggregated memory.

In this paper, we propose CXL-ClusterSim, a full-system modeling and simulation framework by combining the gem5 simulator for fidelity, with the Structural Simulation Toolkit (SST) for parallel simulation. We outline the challenges in creating this simulation infrastructure and present a design that is scalable, flexible, and reasonably fast to help computer architects to explore the design space of CXL-based disaggregated memory and identify new opportunities for hardware/software codesign and performance optimization.

1. Introduction

Emerging applications in scientific computing and artificial intelligence applications are memory-intensive. For instance, GPT-4 has 1.76 trillion parameters, and the number of parameters in large transformer models is expected to grow by a factor of 410× every two years [3, 14]. However, memory performance and capacity have increased at a modest pace due to the challenges in memory scaling [23]. To make matters worse, the ratio of the peak to average memory demand in training large ML models is high. That means systems are typically overprovisioned [20], which adds to the cost and power consumption. For example, Microsoft reports up to 25% of the DRAM capacity becomes stranded in their Azure cloud servers [20].

Researchers propose memory disaggregation, *i.e.*, the decoupling of compute and memory resources, as a solution to these challenges [5, 21]. In this architecture, memory is typically *pooled* out of a memory blade, where physical capacity is aggregated into a central reservoir for dynamic allocation to specific compute nodes [11], or *shared*, allowing multiple nodes to concurrently access the same address space for low-latency data exchange [2]. By enabling both precise resource partitioning and multi-node collaboration, this approach addresses modern workload requirements while significantly mitigating memory underutilization.

Though the idea of memory disaggregation has been around for many years [21], the advent of Compute Express Link (CXL) protocol has made the design and implementation of disaggregated memory systems practical. As an open standard built upon the PCI Express (PCIe) infrastructure, CXL facilitates high-speed, coherent communication between CPUs and memory devices. With its low latency and high bandwidth capabilities, CXL allows for the creation of flexible memory pools that can be dynamically allocated to different processors in a network, effectively addressing the challenges of memory stranding and underutilization in data centers and cloud services.

Despite growing research into CXL-based disaggregated memory, current modeling infrastructures often lack the full-system fidelity required to accurately capture the performance nuances of these emerging architectures. Gouk *et al.* used FPGA-based prototypes, which are based on older CXL specifications, hard to access and modify, and not scalable to multiple nodes [15]. Puri *et al.* introduced DRackSim [30], a simulation framework built on gem5 [22] and DRAMSim2 [32] to model memory disaggregation. However, the framework relies on gem5's Syscall Emulation (SE) mode. Therefore, it cannot account for operating system (OS) overheads or complex kernel-level memory management behaviors.

In another direction, researchers have used NUMA nodes to emulate disaggregated memory [20, 24]. However, studies show that using NUMA nodes as a proxy for CXL-based disaggregated memory is not ideal and reveal some differences between real CXL devices and NUMA-emulated CXL [36].

In this work, we propose *CXL-ClusterSim*, a scalable and flexible simulation infrastructure to model and assess the performance of CXL-based disaggregated memory systems through full-system simulation. Simulation is an indispensable tool in the architectural development of future systems due to its accessibility, scalability, and ability to model complex systems cost-effectively. The current research landscape lacks comprehensive frameworks for the hardware/software co-design of CXL-based disaggregated memory, hindering the design-space exploration. Moreover, CXL specifications are written with functionality in mind, and there is a need to evaluate the practical performance of these theories. CXL-ClusterSim aims to fill this gap by

providing a platform to evaluate the performance of CXL-based disaggregated memory systems.

CXL-ClusterSim is built on top of two popular open-source simulators, gem5 [16, 22] and the Structural Simulation Toolkit (SST) [31]. gem5 is used to model full-system compute nodes (*i.e.*, *hosts* in CXL specification, *system node* in this work) with detailed timing-models, providing a comprehensive view of system behavior, including OS impact, TLBs, huge pages, etc. Each system node consists of multiple cores, a cache hierarchy, and a memory system, running a separate OS and application. SST, on the other hand, is primarily used to simultaneously simulate multiple gem5 nodes, making the simulation of a cluster of system nodes feasible in real time. In addition, SST also models the disaggregated memory blade, referred to as the *remote memory node*. In this work, we demonstrate the capabilities of CXL-ClusterSim, *i.e.*, pooling and sharing, by running a set of full-system experiments.

- First, we run the STREAM benchmark [25, 26] to evaluate the memory bandwidth of the system by varying different NUMA policies. Alongside validating our simulation model, we also present results showing whether system-level effects are captured. We study the scalability by varying the number of hosts. In addition, we show memory pooling with heterogeneous ISAs at two different hosts (ARM and RISCV).
- Second, we demonstrate the ability to run real workloads by running a multiprogrammed example with the NAS Parallel Benchmarks (NPB) [8]. We present a case study of memory stranding effects when NUMA *preferred* is used for the local memory, while the majority of the data is pooled from a 128 GiB device.
- Third, we show memory sharing using GAPBS [9]. We modify the allocation of the graphs to use the remote memory. Several hosts share a graph stored on the remote memory node and execute a graph kernel on the system node. Here we present a *single writer multiple reader* sharing model.

The main contribution of the paper is a full-system simulation and modeling framework for CXL-based disaggregated memory. The framework is a combination of current *state-of-the-art* simulators that will enable future hardware-software co-design research for the computer architecture and computer systems communities. CXL-ClusterSim is not tied to PCIe or link-level details in the CXL specification. It is flexible to explore different implementations of disaggregated memory [2, 18, 19], including the CXL specification and changes to the future CXL specifications. This tool will be released as open-source, and we will work with the gem5 and SST communities to have our extensions merged with the mainline code to the community to enable further research in the area of CXL-based disaggregated memory systems. There are still many more features that can be added to the framework, such as performance/parallel efficiency, CXL table structures, *etc.*, and we look forward to contributions from the community to improve the framework.

Term	Meaning	CXL Spec
system node	A fully-fledged compute node with independent cores, caches, and, operating systems.	host
remote memory node	A memory blade.	device memory
local memory	The main-memory attached with the system node.	local memory
remote memory	Memory inside a memory blade.	host-managed device memory

TABLE 1: Terminologies used in this paper. We define and map them to the CXL specifications. These terms are used interchangeably in this paper.

2. Background and Motivation

2.1. Compute Express Link (CXL)

2.1.1. CXL Overview and Protocols. Compute Express Link (CXL) is an open-standard interconnect designed to facilitate high-speed, low-latency communication between processors, accelerators, and memory expansion devices [1, 2, 34, 35]. It leverages the PCIe physical layer. With CXL 3.1 aligning with PCIe 6.0 to provide up to 64 GT/s per lane [2], CXL maintains memory coherency between the host and attached devices [4]. The protocol stack comprises three sub-protocols: (1) `CXL.io` for device discovery and configuration; (2) `CXL.cache` for device-side caching of host memory; (3) `CXL.mem`, which enables the host to access device memory using standard load/store instructions.

2.1.2. Definitions and Architecture. The CXL specification defines *hosts* and *devices*. Hosts are compute nodes with or without local memory. The devices, on the other hand, are remote memory nodes. There is a global address space where devices are mapped. Hosts may choose to map a part of their own address space into the global address space. The global address space is one contiguous address space. The fabric manager, which is the application-specific logic responsible for orchestrating and managing the control plane of a CXL fabric. In addition, it is tasked with binding hosts and devices to the global address space.

`CXL.mem` supports two memory utilization modes:

- *Memory Pooling*: The device memory is partitioned into distinct segments, each exclusively assigned to at most a single host. This mitigates stranded memory by allowing the fabric manager to reassign capacity based on demand.
- *Memory Sharing*: Introduced in the CXL 3.0 specification [2], this allows multiple hosts to concurrently access the same physical memory range while Back-Invalidate snoops at the cache-line granularity.

Table 1 lists all the terms used for this paper and their correspondence to the CXL specification.

2.1.3. Scope of this Work. The framework provided in this paper is designed to evaluate the *performance* of `CXL.mem`-based disaggregated memory systems only. Although `CXL.io` is used to configure the interface between the host and the device, we keep `CXL.io` as future work since the initialization phase has a negligible impact on the overall performance.

2.2. Simulators

2.2.1. gem5 Simulator. The gem5 simulator is an event-driven, advanced, modular platform for computer architecture research, capable of simulating a wide range of systems from small embedded devices to large-scale server architectures [22]. A *SimObject* is the foundational C++ base class in gem5 representing a hardware component (such as a CPU, cache, or memory controller) that can be instantiated, configured via Python, and managed by the simulation's event queue. Today, gem5 offers a variety of CPU models, ranging from simple in-order processors to complex out-of-order processors, enabling detailed studies of different microarchitectural features. These models support various instruction set architectures (ISAs) such as x86, ARM, RISC-V, *etc.*, making gem5 a versatile tool for cross-ISA research and comparison. In addition to CPU models, gem5 supports detailed modeling of I/O devices like Graphics Processing Units (GPUs), essential for research in parallel processing, machine learning, and high-performance computing.

gem5's memory subsystem provides high-fidelity validated models of memory technologies such as DDR, LPDDR, HBM, and non-volatile memory [16]. The cache models in gem5 are highly configurable, allowing the simulation of different cache hierarchies, policies, coherence protocols like MESI, MOESI, and more, facilitating the exploration of various caching strategies.

In this work, we leverage gem5's full-system mode for simulation. Full-system simulation enables the execution of complete software stacks, including OS and application software, on simulated hardware. For disaggregated memory research, we need to expose the OS interfaces for memory pooling, *i.e.* NUMA and DAX respectively. This mode is particularly valuable for hardware-software co-design research, as it captures system-level interactions which allows researchers to evaluate how changes in hardware (or software) design affect software (or hardware) performance and behavior. By running real workloads and OS such as Linux, researchers can identify bottlenecks, optimize hardware-software stacks and interactions, and develop more efficient and robust systems. In this work, we boot an unmodified Ubuntu-based Linux distribution and run applications compiled on that system, creating an environment exactly the same as what will be found in a real system.

2.2.2. Structural Simulation Toolkit (SST). The Structural Simulation Toolkit (SST) is a comprehensive, highly scalable framework designed for the modeling and analysis of high-performance computing (HPC) systems [31]. The architecture is divided into two primary layers: SST-Core, which provides the underlying event-driven simulation engine and synchronization API, and SST-Elements, a library of modular component models used to construct specific simulation targets. This modularity allows researchers to assemble interchangeable components, such as processors, memory hierarchies, and interconnects, into detailed, custom environments tailored to specific research needs.

We use SST's Message Passing Interface (MPI) for communication between simulated nodes, enabling high-performance parallel execution across distributed computing clusters. Beyond standalone modeling, SST supports sophisticated co-simulation, allowing independent simulators to run concurrently and interact within a unified timeline. This capability is essential for capturing complex system-level interactions, such as how network topology impacts data movement or how memory hierarchy changes affect processor throughput.

Furthermore, SST's extensibility allows it to integrate with external simulation tools to enhance its fidelity. For example, it can be bridged with gem5 to facilitate high-fidelity, full-system CPU-memory simulations [17].

2.2.3. Limitations of Prior SST-gem5 Integration for CXL Disaggregation. While gem5 provides high fidelity by allowing researchers to simulate an entire host in intricate detail, the overall simulation time is slow [6, 17, 28]. This is due to the single-threaded design of gem5 [28]. The simulation time of a cluster of gem5 hosts, running on a single thread, further exacerbates the total simulation time when modeling disaggregated memory. On the other hand, SST provides scalability via the MPI infrastructure. However, it lacks the fidelity provided by a gem5 full-system simulation [17].

The limitation of the prior SST-gem5 integration [17, 29] is that it does not provide a shared functional backing store for simulated memory across distributed MPI ranks. As a result, gem5 functional accesses, which are needed for OS boot, fast-forwarding, checkpointing, and some I/O-device interactions, cannot be resolved correctly when the requested memory is modeled on a different SST rank. We overcome this limitation by using a two-phase workflow: a gem5-only functional initialization phase that boots Linux and captures per-host ROI checkpoints, followed by a timing-accurate gem5 and SST co-simulation phase that restores all hosts across MPI ranks.

A second limitation of the prior SST-gem5 integration is that it does not natively model multi-host CXL fabrics, including switch topologies, packet routing, credit-based backpressure, and fabric-level contention. We overcome this limitation by integrating SST Merlin routers into CXL-ClusterSim, enabling configurable CXL-like topologies and cycle-level evaluation of topology-induced workload slowdown.

3. Simulating Disaggregated Memory

Figure 1 shows an overview of our new simulation infrastructure for modeling CXL-like disaggregated memory. We want to model multi-node CXL clusters with several hosts and devices. We use the gem5 simulator to model and simulate the hosts, and SST to model and simulate the CXL-like interconnect and the remote memory node, or the CXL device.

As discussed in Section 2, both gem5 and SST are *event-driven* simulators. We use the parallel event queue in SST as the main simulation event queue. On each cycle, SST calls into all of the gem5 simulator instances to advance their

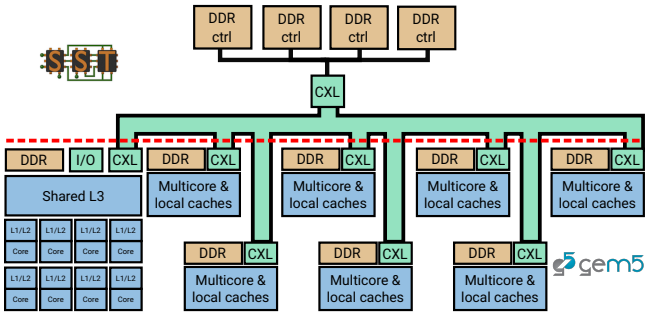


Figure 1: Overview of the simulation infrastructure.

simulation. The parallel event simulation of SST guarantees that all of the gem5 simulations are constantly in sync with one another. Since SST is natively parallel, we should be able to scale the simulation to many nodes and fully use all of the cores on the host system.

The rest of the section presents our approach to simulating `CXL.mem` with system-level fidelity across multiple hosts. Further, we describe the software extensions to gem5 and guide the user on how to get started with the interface.

3.1. Implementation

We organize the implementation discussion around two challenges: *(C1)* initialization and *(C2)* fast-forwarding into the region of interest.

The first challenge, *i.e.*, initialization, is making disaggregated memory OS-visible in a robust, reproducible way. Based on existing research, we opt for exposing the device as a NUMA node when pooling remote memory [20, 24]. For sharing, however, we expose the remote memory as a direct access character device [27].

While SST provides parallelism to gem5 simulations, booting up an OS with `systemd` enabled in gem5 can still take hours [10]. Further, in Section 4.4, we show that contemporary HPC workloads could be up to 85 GiB of application memory, which can take even longer just to allocate the memory. Therefore, we need to incorporate gem5's functional fast-forwarding and checkpointing with SST to jump to the ROI for feasibility.

3.1.1. Challenge 1: Initialization. The first challenge in implementing memory disaggregation is the initialization and mapping of memory ranges across multiple independent system nodes. The CXL 3.1 specification addresses this by exposing the fabric-attached memory to the OS as distinct NUMA nodes [2]. During initialization, system nodes can either allocate memory slices at boot time or via runtime hot-plugging. Regardless of the mechanism, the OS remains agnostic of the underlying hardware topology, interacting with the disaggregated pool through standard NUMA-aware memory management.

While the CXL specification distinguishes between Host Physical Addresses (HPA) and Device Physical Addresses (DPA), our current implementation utilizes a simplified identity mapping with fixed address ranges. These ranges are hardcoded for each system node.

In CXL-ClusterSim, the gem5 simulator executes an unmodified Linux kernel. Consequently, the simulator must provide accurate address details via emulated hardware structures to allow the OS to establish valid software address ranges. On an x86 system, we define separate memory ranges using E820 entries. NUMA topology on x86 is defined via the Static Resource Affinity Table (SRAT). Whereas, on an ARM/RISC-V system, memory ranges and NUMA affinity are defined via the Device Tree Blob (DTB). Figure 2(a) shows how CPU-less NUMA memory node appear in full-system gem5. In both cases, we statically partition memory based on the specific requirements of the target workload.

A critical distinction exists between pooling and sharing in our framework. To facilitate sharing, the remote memory must be exposed as a character device to prevent the kernel from zeroing out the memory and destroying shared data. Following the approach of FAMFS [27], we map the remote memory as a DAX region (`/dev/dax`). This allows user processes to bypass the page cache and access the pool via direct, byte-addressable mappings. Figure 2(b) shows how DAX devices are initialized in full-system gem5. In CXL-ClusterSim, standard NUMA policies are reserved strictly for pooling and are not utilized for shared memory regions.

3.1.2. Challenge 2: Simulating the Region of Interest. The integration of gem5 and SST presents a significant synchronization challenge. As mentioned in Section 2.2.3, gem5 nodes may reside on different MPI ranks, direct functional access, which is common in gem5 for fast-forwarding or emulating non-essential devices, becomes computationally expensive as it requires scheduling global SST events for every memory transaction.

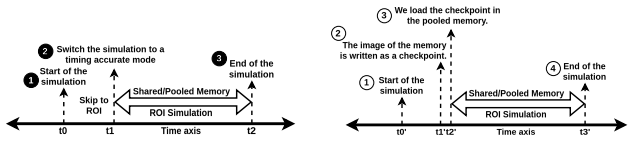
Ideally, a disaggregated simulation should transition seamlessly from functional initialization at time $t0$ (Action ❶) to timing-accurate execution across all nodes (Figure 3(a)). This requires a global synchronization point at $t1$ (shown as Action ❷) to switch from functional to timing models while maintaining a consistent view of the shared memory resource. Action ❸ represents business as

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 1941 MB
node 0 free: 1787 MB
node 1 cpus:
node 1 size: 2007 MB
node 1 free: 1997 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

(a) CPU-less NUMA node via `numactl --hardware` inside gem5. The backend memory is simulated in SST.

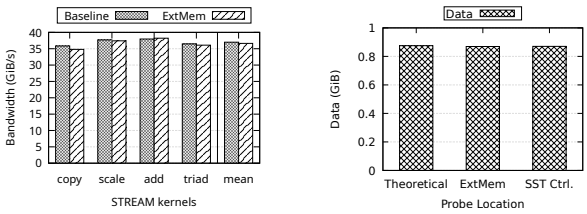
(b) Initialization of a DAX character device in the guest OS inside full-system gem5. The backend memory is simulated in SST.

Figure 2: System-level view of the remote memory ranges.



(a) Ideal simulation timeline. (b) Our simulation timeline.

Figure 3: The simulation timeline for CXL-ClusterSim. We fast-forward the simulation in gem5 to the ROI, take a checkpoint and restore the simulation in gem5 and SST using a timing-based model.



(a) Baseline and observed bandwidth reported by the STREAM kernels at the external memory per kernel.

(b) Comparing the total baseline bandwidth seen in Section 4.1, at the external memory, and at the remote memory controller.

Figure 4: Statistics used to validate the correctness of CXL-ClusterSim. We show the baseline bandwidth of the system and compare it against the bandwidth of a single system.

usual as we proceed to collect and analyze the statistics of the simulation.

The lack of a shared functional backing store in a distributed SST environment makes this "ideal" transition difficult to implement directly. To overcome this, CXL-ClusterSim employs a two-phase simulation strategy using checkpointing (Figure 3(b)).

We decouple the simulation into an initialization phase and a timing-accurate ROI (Region of Interest) phase:

1) **Functional Initialization (gem5-only):** At time $t0'$, we start each system node in gem5 using either the `AtomicSimpleCPU` or `KVM` CPU for high-speed fast-forwarding. During this phase, the memory is modeled functionally. Once the simulation reaches the ROI, we capture a memory checkpoint (Action ②). This process is repeated for each of the N nodes to establish their unique memory ranges.
2) **Timing-Accurate Co-Simulation (gem5 + SST):** We restore the N system node checkpoints within the integrated gem5-SST environment. At this stage ($t2'$), we switch to timing-accurate O3 CPU and memory models. Action ② and Action ③ together form the equivalent of Action ❷. This restoration acts as the global synchronization point, allowing all nodes to proceed in parallel until the simulation concludes at $t3'$ (Action ④).

This checkpoint-based approach allows CXL-ClusterSim to leverage the fast-forwarding capabilities of gem5 while maintaining the parallel scalability and timing accuracy provided by SST.

4. Evaluation

This section describes the evaluation platform used for the experiments. As mentioned before, CXL-ClusterSim uses gem5 [22] and SST [31]. The system nodes are modeled in gem5. Each of these is a comprehensive system, equipped with its own CPUs, caches, and dedicated local memory. They have their own independent operating system (OS). The parameters of the system node that we modeled are given in Table 2.

4.1. Calibration

We validate and calibrate the basic model of the remote memory node using synthetic traces generated at the system node. The objective here is to calibrate our remote memory to a real-world memory device. Our objective is to model a 2400 MHz 4-channel memory with a peak theoretical bandwidth of 76.8 GB/s. The system that we used for the same has a synthetic traffic generator connected directly to a cross-bar switch, connected to the remote memory. We experiment by sending linearly generated synthetic read traffic using gem5's traffic generators. The memory is modeled in SST. This allowed us to find the maximum sustained and practical bandwidth without causing bank conflicts. The average total bandwidth reported was 59.6 GB/s. This is 77.5% of the peak theoretical bandwidth of the DRAM device that we are trying to model. We consider this value the baseline bandwidth of the remote memory.

4.2. Case Study I: Basic Full-System Experiments

We use the popular benchmark STREAM [25, 26] to validate the full-system capabilities of our platform. STREAM is a synthetic benchmark program that measures the total memory bandwidth. There are four kernels in STREAM: copy, scale, add, and triad. The first two kernels work with two arrays, and the latter two kernels use three arrays. We set the size to 64 MiB for each of these arrays.

TABLE 2: Simulation Parameters

Parameter	Specification
CPU type	Out-of-Order
ISA	ARM (unless specified)
CPU core count	8
Frequency	4 GHz
L1 (I/D) cache per core	32 KiB / 32 KiB
L2 cache per core	512 KiB
L3 cache	8 MiB
Memory technology	DDR4 DRAM
Memory frequency	2400 MHz
Number of local channels	1
Number of remote channels	4
Local memory size	8 GiB (unless specified)
Remote memory size	variable/node
Operating system	Ubuntu 22.04.4
Kernel version	5.4.49

For the full-system experiments, we use the Linux utility `numactl` to pin the program's memory only to the local memory (local), interleaving the pages between the two memory devices (interleave), or pinning all data to the remote memory (remote). The local memory has a single DDR4 DRAM channel with a peak theoretical bandwidth of 19.2 GB/s. Each system node assigned a 1 GiB range of memory for this experiment.

We also added gem5's magic instruction-based annotations for the region of interest (ROI) at the beginning and end of each of the four kernels. The changes we made to the program allow us to take and restore checkpoints in gem5 and measure the bandwidth of each kernel individually.

4.2.1. Validating CXL-ClusterSim. The first set of results are shown with a single system node connected to a remote node (Figure 4). In Figure 4(a), we show the baseline and reported bandwidth of the STREAM kernels when `numactl` pins the kernels to the remote memory. The `ExternalMemory` (i.e., the simulated CXL link) reports how much data was moved over this link. We expect to see the baseline and the reported numbers to be *similar* and not exact. This is because there are factors such as caching and prefetching affecting the simulation. The difference reported is less than 1.0%. This suggests that CXL-ClusterSim is set up correctly from gem5's side. Further, Figure 4(b) shows the total bandwidth reported by the SST's memory controller. SST reports the bandwidth off by 0.1%, which is attributed to resetting the statistics using the gem5 annotations.

> **Observation 1**: The baseline and the reported bandwidth using CXL-ClusterSim are within 0.99%. This suggests that our memory disaggregation model is correct.

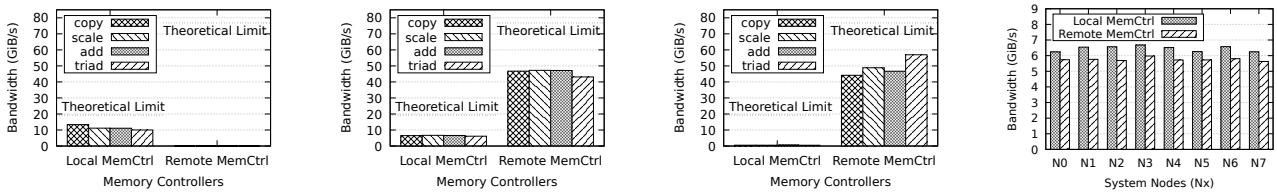
4.2.2. Running STREAM across multiple system nodes. The bandwidth reported at each of the memory controllers in an eight-node cluster is shown in Figure 5. We show the "Local MemCtrl" bandwidth as the average bandwidth of each of the system nodes, and the "Remote MemCtrl" bandwidth as the bandwidth at the remote memory system. We used `numactl` to pin STREAM to use the local memory (Figure 5(a)), interleave between the local and the remote memory (Figure 5(b)), and pin the application memory to the remote memory (Figure 5(c)). We do not see any significant traffic at the remote link when we pin the process to the local memory. Likewise, the traffic is exclusive to the remote memory node when the process is pinned to the same.

In the interleaved memory setup, application bandwidth is heavily constrained by the slower remote memory node. Figure 5(d) shows that the effective memory bandwidth per system node peaks at only 6.45 GiB/s, a significant drop from the 11.4 GiB/s baseline of strictly local memory (Figure 5(a)). Because interleaved NUMA distributes memory allocations across both local and remote nodes, the performance is bottlenecked by the shared remote link. Specifically, the remote node delivers a measured total bandwidth of 46.04 GiB/s across all eight compute nodes (out of the 59.6 GiB/s, as discussed in Section 4.1). This equates to an average remote bandwidth of just 5.75 GiB/s per node, which fundamentally throttles the overall memory throughput of the interleaved execution.

> **Observation 2**: The `numactl` can be effectively used in CXL-ClusterSim. The remote node behaves close to a memory blade, where it is constrained by its own specifications.

4.2.3. CXL Latency. Our next experiment adds latency to the link to the remote memory system (i.e., to model CXL latency). In this experiment, we model four system nodes cluster instead of eight in the previous experiment. We use SST to model this CXL latency. Sharma *et al.* reported that the initial CXL latency numbers reported were within the range of 170 ns to 250 ns [33]. Therefore, we show the bandwidth of the system running STREAM remotely by varying the CXL latency in Figure 6.

In the experiment, we can clearly see that the total bandwidth is the highest when there is no CXL latency. The bandwidth decreases as we increase the latency. The impact on the bandwidth from no latency to 170 ns is 8.95%, and to 250 ns is 29%. Compared to the 170 ns case,



(a) When `numactl` pinned the program to the local memory.

(b) When `numactl` interleaved the program to use both the local and the remote memory.

(c) When `numactl` pinned the program to the remote memory.

(d) Reported average bandwidth per kernel at each of the memory controllers. The program was interleaved between the two memories.

Figure 5: The total bandwidth was measured at each of the memory controllers. There were 8 system nodes in the cluster running STREAM.

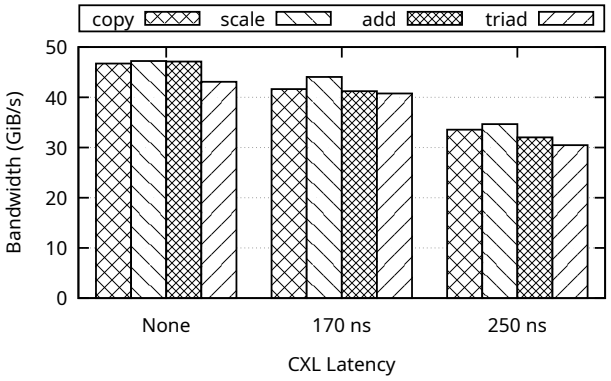


Figure 6: The bandwidth reported by each of the STREAM kernels when the CXL latency was varied. There were 4 system nodes.

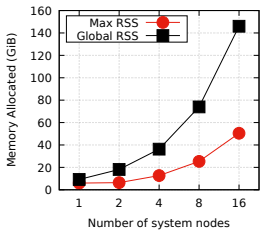
the dip in performance for 250 ns is 22%. Backpressure is implemented on the SST side for the external CXL links.

Observation 3: The disaggregated memory latency has an impact on the bandwidth of the remote node.

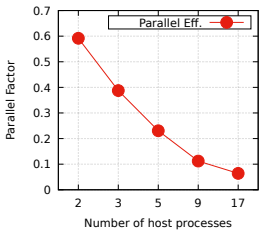
4.2.4. Host System and Parallelization Statistics.

The host system (i.e., the system that is executing the gem5 simulations) also constitutes an important part of the simulations. We used an ARM-based Neoverse-N1 architecture with 160 cores. The system was equipped with 512 GiB of total memory. In this section, we report the simulation statistics for the host system when we increased the number of system nodes from 1 to 16 in powers of 2. Collectively, these statistics are shown in Figure 7.

The first statistic is the amount of host memory required. Figure 7(a) shows the memory requirements on the host system for simulating system nodes from 1 to 16. Since SST



(a) The amount of host memory required to simulate 1 to 16 system nodes. We report the maximum resident set size (Max. RSS) and approximate maximum global resident set size (Global RSS) of the simulation.



(b) Reported parallel efficiency of the cluster when scaling system nodes from 1 to 16 in powers of 2. There is one extra MPI rank for the remote memory node.

Figure 7: Reported host system statistics when the number of system nodes varied.

is an MPI-based software, we report the MPI statistics of SST at the end of each simulation. The memory required by one MPI process is given by the *maximum resident set size*, or simply *RSS*. The total amount of memory required for the entire simulation cluster is given by *Approximate global RSS size*. Our simulations for 8 system nodes allocated 25.3 GiB of memory per system node. The global RSS size was reported to be 73.9 GiB. For 16 system nodes, the global RSS size was reported to up to 145.96 GiB. The trend is observed to be linear for both RSS and global RSS as we scale with the number of system nodes.

The other statistic that is crucial for the CXL-ClusterSim is the wall clock or the simulation runtime. Since SST enables parallel simulation, we would hope that as we increased the number of simulated system nodes the performance of the overall simulation would not degrade. However, we found that as we increased the number of system node, the wall clock time increased significantly.

Figure 7(b) shows the parallel efficiency when scaling system nodes. We use the single system node running STREAM in gem5 memory as our baseline case.

Parallel efficiency (PE) is defined in Equation 1. Parallel efficiency is 1.0 if the time running N system nodes on N threads is the same as the serial execution of a single system node. Parallel efficiency below 1.0 implies synchronization or other overheads in the parallelization of the simulation.

$$\mathrm{PE} = \frac{1}{\mathrm{NUM}_{\mathrm{processes}}} \times \frac{\mathrm{TIME}_{\mathrm{gem5\ only}}}{\mathrm{TIME}_{\mathrm{CXL\text{-}ClusterSim}}} \tag{1}$$

We start by showing the parallel efficiency with two host processes as the first point in Figure 7(b). In this case, we have one host process executing the gem5 simulation (the system node) and one process executing the remote memory system. In this case, we see a 16% improvement in wall clock time compared to running gem5 alone, which implies that there is some parallelism between the compute-based node and the memory node. However, as we continue to scale the system nodes, we see an degradation in the parallel efficiency.

The PE of a two node cluster is 0.38. The number dips to 0.06 for the 16 system nodes case. The reason for this decrement is the bottleneck caused at the SST-core for the remote memory node. We have pinned the STREAM kernels to the remote memory only, which makes the simulator serialize the incoming memory requests across all the system nodes for the remote memory rank.

Later in Figure 14, we show that the memory footprint of these real-world workloads is distributed between local and remote memory. Because the workloads do not rely exclusively on the remote node, the volume of remote memory accesses is limited, which significantly reduces the overall simulation time.

While the parallelism is not efficient, there is a slight speedup (about 1.09× speedup) over the serial version at 17 host processes. This provides an opportunity to improve CXL-ClusterSim's efficiency in the future.

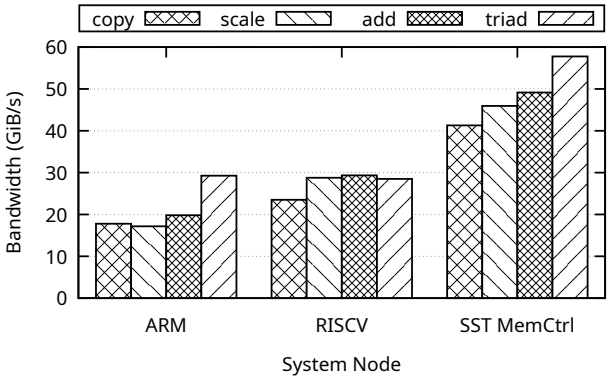


Figure 8: Bandwidth reported by the two system nodes and at the SST's memory controller when two heterogeneous system nodes ran STREAM remotely.

> **Observation 4**: The parallel efficiency is better than linearly scaling the system nodes. There are ample opportunities to improve this further.

4.2.5. Mixing ISAs. The final set of experiments for STREAM shows how heterogeneous ISA system nodes can pool memory from the remote memory node. The memory device is agnostic of the ISA. As long as there are incoming memory requests within its range, the remote node is expected to function. A system node should only be aware of memory ranges. Therefore, we set up an interesting experiment where we use an ARM-based system node and a RISC-V-based system node. Except for the ISA, every other parameter of the systems is identical, defined in Table 2.

We execute STREAM and pin it to the remote memory. Figure 8 shows the reported bandwidth, and we find the results are similar to what we have already seen in Section 4.2.2. This adds a new angle for researchers to explore using CXL-ClusterSim. This is especially interesting as the data for the RISC-V node shows that it was able to exploit the remote node's bandwidth by 31% more than that of the ARM system node. This opens the door to studying and comparing how the differences in the core architecture can drive memory bandwidth utilization.

> **Observation 5**: The remote node is agnostic of the system node's architecture. CXL-ClusterSim, therefore, can be used to study heterogeneous CPUs alongside heterogeneous memories.

4.3. Case Study II: Workload Performance Under Allocation and Placement Policies

This case study directly addresses data allocation and placement strategies in CXL-based disaggregated memory systems. We configure STREAM kernels across a 4-node disaggregated cluster under local DRAM capacity constraints

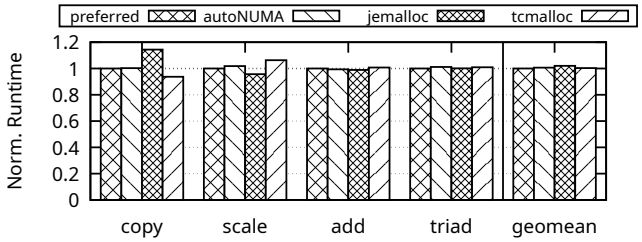


Figure 9: Reported STREAM runtime normalized to numactl –preferred for several allocation and placement policies across four system nodes.

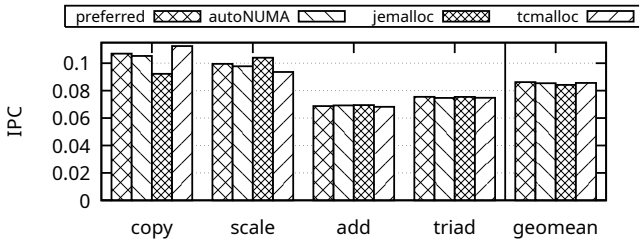


Figure 10: Reported STREAM IPC for several allocation and placement policies across four system nodes.

(64 MiB per node) to intentionally force memory spillover into a 1 GiB remote CXL pool (totaling to 4 GiB). We then study how different allocation and placement policies affect STREAM runtime and IPC under forced remote-memory spillover.

Note that we do not evaluate allocator implementation overhead in isolation. Instead, we evaluate how end-to-end workload performance changes when data is placed using different OS-level and user-level allocation policies under constrained local-memory capacity. This distinction is important because, in a disaggregated memory system, the architectural question is how allocation and placement behavior affects local-memory pressure, remote-memory spillover, CXL-link traffic, and fabric contention.

While we have already shown allocations via NUMA bindings in Section 4.2.2, this case study explicitly demonstrates CXL-ClusterSim's flexibility in supporting full-stack hardware-software co-design. We split this section into two parts: *(a)* allocation and placement policies, and *(b)* network topology exploration.

4.3.1. Workload Performance Under Allocation and Placement Policies. We evaluate how user-space and kernel-space memory-management policies affect workload runtime and IPC under heavy CXL memory spillover. Specifically, we compare four allocation and placement policies: *(a)* static kernel preferred placement using `numactl --preferred`, *(b)* kernel-level dynamic page placement using AutoNUMA, *(c)* `jemalloc` [12], a user-space allocator that uses thread-local arenas to reduce allocation lock contention, and *(d)* `tcmalloc` [13], a user-space allocator that uses per-thread caching lists. Figure 9 and Figure 10

report STREAM runtime and IPC under these policies. These results should be interpreted as workload-level effects of allocation and placement decisions, not as allocator microbenchmark results.

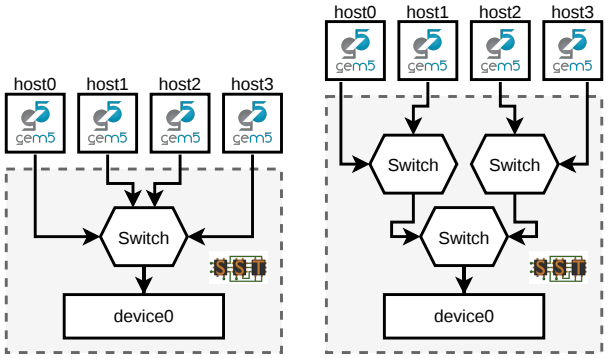
Out-of-the-box user-space allocation policies do not provide blanket speedups over standard OS placement under disaggregated memory spillover. Under the `tcmalloc`-based allocation policy, the STREAM Copy kernel achieves a 0.937× normalized runtime relative to `numactl --preferred`, corresponding to a 6.30% speedup. However, this improvement does not generalize across all kernels: `tcmalloc` slightly degrades performance on multi-operand kernels and yields an overall geometric mean runtime of 1.003×, corresponding to a 0.32% slowdown.

Under the `jemalloc`-based allocation policy, the Scale kernel improves to 0.957× normalized runtime, corresponding to a 4.33% speedup. However, Copy slows down to 1.144×, which we attribute to allocator metadata management and background page-decay behavior. Overall, `jemalloc` produces a 1.020× geometric mean runtime, corresponding to a 2.02% slowdown relative to `numactl --preferred`. Kernel-level dynamic page placement using AutoNUMA closely tracks static preferred placement across the evaluated kernels, with a 1.006× runtime geomean, or 0.65% overhead. Because STREAM exhibits uniform, non-phase-changing memory access patterns, AutoNUMA's page scanning introduces minor overhead without finding substantial access-locality changes that would justify page migration across CXL links.

As kernel complexity increases from two-stream operations such as Copy and Scale to three-stream operations such as Add and Triad, the runtime differences across allocation and placement policies shrink. In these memory-intensive kernels, hardware-level queuing delays, CXL interconnect latency, and remote-memory contention increasingly dominate end-to-end performance, reducing the visible impact of software-level placement differences. This result demonstrates that CXL-ClusterSim can expose when software data-placement policies matter and when fabric-level bottlenecks dominate.

> **Observation 6:** CXL-ClusterSim enables end-to-end evaluation of workload performance under different data allocation and placement policies.

4.3.2. Network Topology Exploration. To showcase CXL-ClusterSim's modularity in modeling complex interconnect architectures, we integrate the SST Merlin network router framework to evaluate packet routing and credit-based backpressure across two distinct network topologies, shown in Figure 11. In the Star topology, all host nodes connect directly to a single central switch, minimizing network hops between hosts and the remote memory device. In the two-tier Tree topology, host nodes connect through a multistage hierarchy of switches, introducing intermediate network hops before requests reach the remote memory device.



(a) Single-switch star topology where all host nodes connect directly to a single central switch that routes traffic to the remote memory device

(b) Two-tier tree topology where host nodes connect through a multi-stage hierarchy of switches, introducing intermediate network hops to reach the remote memory device.

Figure 11: Network topologies evaluated for CXL interconnect modeling.

Figure 12 shows the average STREAM runtime under the evaluated CXL-like fabrics when combined with the same allocation and placement policies from Section 4.3.1. The direct-link configuration provides the lowest-latency path to remote memory. The Star topology adds a central switch but avoids the extra hierarchy of the Tree. In contrast, the Tree topology introduces additional router hops and concentrates traffic near the root switch. Under concurrent multi-host access, this concentration creates flit-level credit backpressure and head-of-line blocking, increasing end-to-end runtime. As a result, the Tree topology produces up to a 4.33× slowdown compared to the direct-link configuration and a 1.70× slowdown relative to the Star topology.

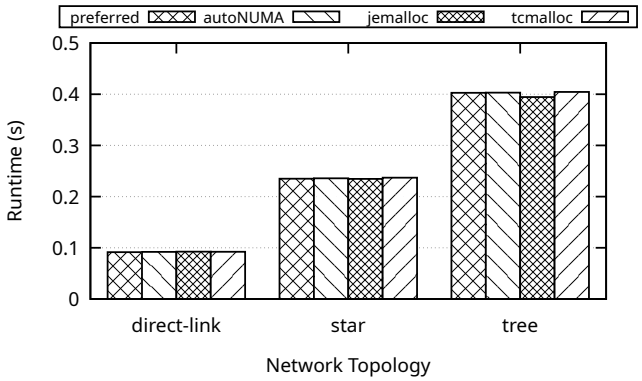


Figure 12: Average STREAM runtime under different CXL-like network topologies and allocation/placement policies.

TABLE 3: NPB Workloads Working Set Sizes in class D

Applications	Working Set Size (GiB)
bt	11
cg	17
ep	1
ft	85
mg	27
sp	12
ua	8

Observation 7: CXL-ClusterSim enables joint exploration of software-level data placement policies and hardware-level CXL fabric topology.

4.4. Case Study III: Memory Pooling

In this study, we evaluate the performance of disaggregated systems while running the NAS Parallel Benchmarks (NPB) [8]. The NPB is a set of scientific programs designed to help evaluate the performance of parallel supercomputers, and they are widely used to assess computational and communication performance in various scientific and engineering domains.

For this study, we use CXL-ClusterSim to configure two different systems to study the memory stranding issue in datacenters, for which memory disaggregation solution has been proposed and implemented. To this end, we consider a cluster that consists of 7 system nodes (host in CXL spec), each with 8 cores, a cache hierarchy, and a connection to the remote memory node. The system configuration parameters are described in Table 2. We chose a subset of NPB workloads, each of them running on a separate system node. Table 3 shows the working set size of the used applications from NPB, rounded up to the nearest GiB. We chose class D as the input size for the NPB workloads, as it is sufficiently large for the systems we configure. We simulate these systems for 500 ms in full-system mode that captures the interactions of the entire system, including the operating system (TLBs, NUMA, huge pages, etc.) and the application behavior during runtime.

We define two different memory setups, as follows:

- **No-NUMA**: In this configuration, each system node is equipped with 128 GiB of local memory. We choose 128 GiB based on the largest workload we test (ft, 85 GiB), rounded up to the nearest power of two. Given the ample local memory, all data required by the benchmarks fit entirely within the local memory. This setup removes the need for remote memory accesses, thereby reducing potential remote memory access latencies. However, it cannot guarantee that the system node will utilize the entire 128GiB of memory, as the working set size of the application may be smaller than the local memory size.
- **NUMA-Local-Preferred**: In this configuration, the system is equipped with a significantly smaller local memory of 8 GiB, supplemented by a shared pool memory of 160 GiB. This setup is designed to reflect a scenario where the local memory is insufficient to hold all the data, necessitating that a portion of the data be stored in remote memory. The system employs a NUMA policy with a preference for local memory access. Thus, it first starts to allocate memory from local memory. If the working set size of the application is larger than the local memory, the remaining portions of the workload inevitably reside in the remote memory, leading to increased latency and potential performance degradation.

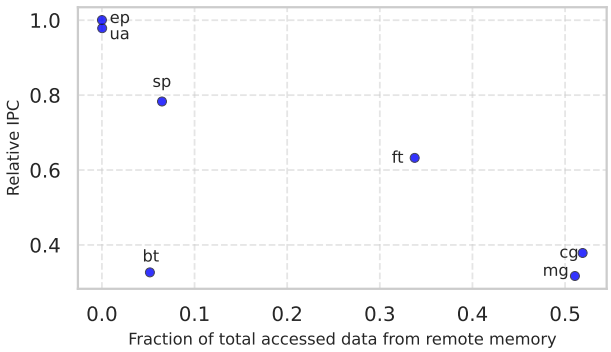


Figure 13: NPB workloads' relative IPC of NUMA-Local-Preferred memory setup compared to the No-NUMA, against the fraction of working set size that was accessed in the remote memory. Y-axis starts at 0.3.

Figure 13 shows the relative instruction per cycle (IPC) of the NPB workloads in the NUMA-Local-Preferred memory setup compared to the No-NUMA memory setup on the Y-axis. It also shows the fraction of the working set size that was accessed in the remote memory on the X-axis. The figure shows that the relative IPC of the workloads decreases as the fraction of the working set size accessed in the remote memory increases, due to the longer latency of accessing the remote memory node and potential interference once the application accesses the remote memory simultaneously. *sp*, *ft*, *cg*, and *mg* show a decrease in their relative IPC as the fraction of accessed remote data increases. The larger the fraction of the working set size accessed in the remote memory, the higher the performance degradation. For instance, 52% of the data of *mg* (working set size of 27 GiB) was accessed in the NUMA-local-Preferred setup was in the remote memory, which resulted in a 0.38 relative IPC, *i.e.*, 62% overall slowdown in its performance compared to No-NUMA setup. Even though the relative IPC decreases, the amount of memory stranding at each system node's local memory is significantly reduced as well. In the No-NUMA setup, each system node has 128 GiB of local memory, which is significantly higher than the working set size of most of the workloads. For example, in the *mg* workload, the working set size is 27 GiB. This means that 79% of the stranded local memory is saved in the NUMA-Local-Preferred setup.

Figure 13 also shows that the relative IPC of the workloads is not significantly affected when the fraction of the working set size accessed in the remote memory is negligible. *ep* and *ua* workloads do not show a significant decrease in relative IPC, as the fraction of accessed remote data for them is close to zero. Note that the NUMA-Local-Preferred memory setup has 8 GiB of local memory. As Table 3 shows, the working set sizes of the *ep* and *ua* workloads are 1 GiB and 8 GiB, respectively, fitting entirely in the local memory.

Out of all the workloads, only *bt* shows a decrease in relative IPC while the fraction of accessed remote data is not significant. The anomalous IPC degradation of the *bt* workload, despite a seemingly low fraction of remote data access, is driven by a combination of `numactl --preferred`, and its memory footprint and highly irregular access patterns. `numactl --preferred` is a soft policy, which, unlike `numactl --mem-bind`, tries to use a preferred numa node, which may not yield positive results everytime. For workloads with irregular access patterns like *bt* and *sp*, we saw varying results each time we ran the experiment. On the other hand, workloads with regular access patterns or smaller remote memory footprint like *mg*, *ep* and *ua* had the fraction of remote memory data similar across multiple runs.

> **Observation 8**: While remote memory access significantly degrades IPC due to increased latency and interference, the NUMA-Local-Preferred setup effectively minimizes memory stranding by rightsizing local allocations to actual workload demands.

4.5. Case Study IV: Memory Sharing

To evaluate memory sharing, we study graph analytics workloads using GAPBS [9]. CXL 3.0 enables memory sharing across hosts; in our framework, CXL-ClusterSim models this on x86 by mapping the remote memory node as a DAX region, akin to the FAMFS approach [27], so user processes can access the remote memory via a direct, byte-addressable mapping without page cache mediation. This mirrors current programming practice with `/dev/dax` for heterogeneous memory systems and allows us to exercise cross-host sharing semantics in a controlled setting.

We use a modified version of GAPBS that supports loading a single graph image into the remote memory node once and then mapping that image into multiple hosts. One designated "*allocator*" host performs the initial allocation and population of the graph in the shared remote memory (the single writer), while the remaining hosts map the graph as read-only (multiple readers). This reflects a common analytics pattern where many readers traverse a static graph snapshot. We concurrently run six GAPBS kernels (*bfs, bc, cc, cc_sv, pr,* and *tc*) each on a separate system node, all reading the same remote graph. We validated the correctness of the changes to the version of GAPBS by observing the output of each of the kernel with the unmodified version.

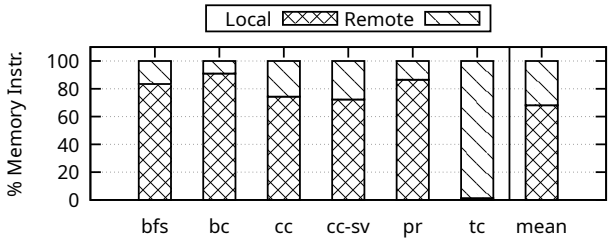


Figure 14: Share of retired memory instructions served by *local* vs. *remote* memory across all the GAPBS kernels under our sharing setup. Bars are normalized to 100% per kernel. *Remote* denotes requests serviced by the SST memory node, and *Local* denotes the node's attached DRAM.

Figure 14 breaks down the fraction of retired memory instructions served by local DRAM versus the shared remote blade across all six GAPBS kernels under our single-writer, multiple-reader configuration. The split is kernel-dependent, confirming that a non-trivial portion of accesses are satisfied by the remote blade when the graph resides in shared memory, while private/stack activity remains predominantly local. On average, 31.8% of the total memory instructions are served from the shared remote memory node.

We compare against a baseline configuration that runs each kernel on a single gem5-only system with private memory (no remote access), reporting instructions per cycle (IPC). In the shared-memory configuration, all kernels run in parallel across hosts, accessing the single, DAX-mapped graph in the remote memory node. We add remote memory path latency in SST (250 ns). Unless otherwise noted, per-kernel IPC is collected over each kernel's region of interest, and we also report a geometric mean across kernels for compactness.

Figure 15 shows the results. As expected, moving the graph to the shared remote memory node reduces IPC relative to the baseline due to *(1)* extra access latency on the CXL path and *(2)* cross-host contention at the remote memory controller when kernels co-execute. Increasing the injected CXL latency will further degrade performance.

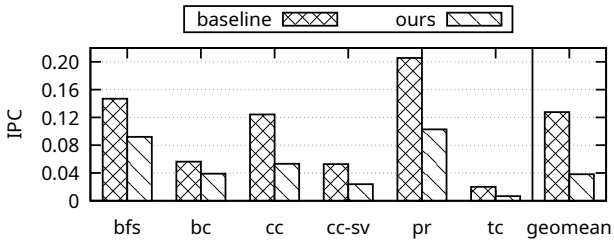


Figure 15: Reported IPC of GAPBS kernels in CXL-ClusterSim, when the same unweighted synthetic graph was shared across all the kernels. We compare the IPC of CXL-ClusterSim's experiments with a single system running one kernel without additional latencies.

Kernels with more irregular, pointer-chasing access patterns (bfs, pr) tend to be more sensitive than those with more streaming-friendly behavior, while compute-heavy kernels exhibit milder slowdowns. The geometric mean reflects these trends across kernels.

This experiment demonstrates that CXL-ClusterSim can model cross-host sharing from user space via DAX-style mappings and capture interference effects in the shared remote memory node. Our current sharing model intentionally uses a single-writer/multiple-reader discipline with read-only mappings on readers; we do not model host-to-host cache-line coherence and keep it as future work. Additionally, the present CXL path omits link-level back-pressure and switch modeling, which likely understates tail-latency growth and interference under heavy co-tenancy; we leave detailed switch/fabric and credit-based flow-control modeling to future work.

Observation 9: CXL-ClusterSim demonstrates that CXL-based memory sharing via DAX mappings enables multi-host analytics at the cost of performance degradation driven by remote access latency and controller contention, particularly in pointer-chasing workloads.

5. Related Work

gem5-based Distributed System Simulation. gem5 is a cycle-level simulator offering a rich amount of micro-architecture details while providing full-system simulation capability. Bringing up multiple system nodes within a single gem5 simulation is a possibility, however only a single CPU thread is used for simulating all the gem5 nodes. As a result, gem5-based distributed system simulation has been the target of many prior works extending the parallelization capability of gem5.

Alian *et al.* propose dist-gem5 [28], a modified version of gem5 supporting distributed system simulation via MPI. The work allows multiple gem5 instances to run simultaneously, and facilitates the communication between the instances using an ethernet inter-node switch model. Compared to our work, dist-gem5 requires maintaining the MPI-related handling, while our gem5-SST implementation offloads this responsibility entirely to SST. COSSIM [37] is another distributed gem5-based framework that is capable of simulating multiple system nodes, and the inter-node communication is primarily ethernet-based. While our work also targets simulating distributed system, the communication of a disaggregated memory system is not ethernet-based.

Puri *et al.* proposed DRackSim [30], where hosts are modeled in gem5, and the remote memory is modeled in DRAMSim2 [32]. The framework relies on gem5's Syscall Emulation (SE) mode, which does not take system-level overheads or kernel-level memory management behaviors.

Recent frameworks like Cohet [39, 40] and Xerxes [7] offer valuable micro-architectural models for single-node CPU-XPU coherence and CXL 3.0+ fabric routing in gem5, respectively. gem5-CXL [38] on the other hand, focuses on optimizing localized memory parallelism. Crucially, CXL-ClusterSim is complementary to these efforts as we focus on multi-host simulation. Both of these works' localized device pipelines and link-layer optimizations can be modularly plugged into our core gem5 component, allowing SST to seamlessly scale their innovations to explore cluster-wide memory pooling and multi-host resource contention.

SST is a discrete event-based simulation toolkit which comes with a MPI-based framework for parallelizing simulations and facilitating communication between simulations [31]. SST naturally complements the lack of parallelization in gem5, as well as offers the separation of the responsibility of handling MPI events to SST. Unsurprisingly, there is a history of integration between gem5 and SST. Hsieh *et al.* [17] show an integration of gem5 and SST in 2012, demonstrating the scaling capability of scaling the number of system nodes using SST. However, we are not aware of the status of the 2012 integration with more recent versions of gem5 and SST. A more recent integration of gem5 and SST in 2021 [29] addresses the maintainability of gem5/SST integration by making the integration part of gem5 tests. The work is demonstrated to be able to bring up multiple gem5 *cores* running in parallel using a recent version of SST. This prior gem5-SST combination was focused on providing the detailed core model of gem5 as a component to SST-based simulations without fast-forwarding or checkpointing.

Disaggregated Memory System Performance Profiling. Sun *et al.* have characterized the performance of CXL 1.1 devices [36]. They highlight the difference between CXL 1.1 devices and the NUMA-based emulations. The work is limited to understanding the performance characteristics of the CXL 1.1 protocol and related devices. Our work captures the performance characteristics by modeling essential characteristics of CXL devices, such as CXL latency, while keeping the implementation simple to enable disaggregated memory system exploration.

6. Conclusion

We present CXL-ClusterSim, a full system modeling and simulation infrastructure for performance evaluation and design space exploration of CXL-based disaggregated memory. The proposed system has some limitations. Dynamic memory management via memory hot-plugging, and the details of CXL's coherence specification are not incorporated yet. These constitute our plans for future work. One can take advantage of the framework infrastructures and extend to evaluate the `CXL.io`/`CXL.cache`-based systems. Even with these limitations, we expect CXL-ClusterSim to be a valuable asset to the computer architecture research community to benchmark and evaluate hardware/software co-design and optimization ideas in this rapidly evolving research topic.

Acknowledgments

We thank Ayaz Akram, Bobby Bruce, Mahyar Samani, William Shaddix, Carolina Minami Oguchi and the other

members of DArchR who provided feedback on earlier versions of this work and contributed to the development of the gem5-SST integration. This work was supported in part by the National Science Foundation under grant numbers 1925724, 2144883, 2311888 and 2225882, Department of Energy under grant number DE-SC0021149, and the Laboratory for Physical Sciences.

LLMs were used for editorial purposes and code generation in this manuscript. All outputs were inspected by the authors to ensure accuracy and originality.

References

[1] "CXL® 4.0 Specification." [Online]. Available: https://computeexpresslink.org/cxl-specification/"

[2] "CXL® Specification." [Online]. Available: https://computeexpresslink.org/cxl-specification/"

[3] "GPT-4 architecture, datasets, costs and more leaked," 2020. [Online]. Available: https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/"

[4] "Compute Express Link (CXL): All you need to know," 2024. [Online]. Available: https://www.rambus.com/blogs/compute-express-link/"

[5] M. K. Aguilera, E. Amaro, N. Amit, E. Hunhoff, A. Yelam, and G. Zellweger, "Memory disaggregation: why now and what are the challenges," *SIGOPS Oper. Syst. Rev.*, vol. 57, no. 1, p. 38–46, jun 2023. [Online]. Available: https://doi.org/10.1145/3606557.3606563

[6] M. Alian, D. Kim, and N. Sung Kim, " pd-gem5: Simulation Infrastructure for Parallel/Distributed Computer Systems ," *IEEE Computer Architecture Letters*, vol. 15, no. 01, pp. 41–44, Jan. 2016. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/LCA.2015.2438295

[7] Y. An, S. Yi, B. Mao, Q. Li, M. Zhang, D. Zhou, K. Zhou, N. Xiao, G. Sun, Y. Luo, and J. Zhang, "Xerxes: Extensive exploration of scalable hardware systems with CXL-Based simulation framework," in *24th USENIX Conference on File and Storage Technologies (FAST 26)*. Santa Clara, CA: USENIX Association, Feb. 2026, pp. 329–345. [Online]. Available: https://www.usenix.org/conference/fast26/presentation/an

[8] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The NAS Parallel Benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[9] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," 2017. [Online]. Available: https://arxiv.org/abs/1508.03619

[10] B. R. Bruce, A. Akram, H. Nguyen, K. Roarty, M. Samani, M. Friborz, T. Reddy, M. D. Sinclair, and J. Lowe-Power, "Enabling reproducible and agile full-system simulation," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 183–193.

[11] A. Cho and A. Daglis, "Starnuma: Mitigating numa challenges with memory pooling," in *Proceedings of the 2024 57th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '24. IEEE Press, 2024, p. 997–1012. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00077

[12] J. Evans, "A scalable concurrent malloc (3) implementation for freebsd," in *Proc. of the bsdcan conference, ottawa, canada*, vol. 2, no. 3.1, 2006, p. 5.

[13] S. Ghemawat and P. Menage, "TCMalloc: Thread-Caching Malloc." [Online]. Available: https://goog-perftools.sourceforge.net/doc/

[14] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "Ai and memory wall," *IEEE Micro*, 2024.

[15] 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 22)*, 2022, pp. 287–294.

[16] A. Hansson, N. Agarwal, A. Kolli, T. F. Wenisch, and A. N. Udipi, "Simulating DRAM controllers for future system architecture exploration," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014, Monterey, CA, USA, March 23-25, 2014*. IEEE Computer Society, 2014, pp. 201–210. [Online]. Available: https://doi.org/10.1109/ISPASS.2014.6844484

[17] M. Hsieh, K. Pedretti, J. Meng, A. Coskun, M. Levenhagen, and A. Rodrigues, "Sst + gem5 = a scalable simulation infrastructure for high performance computing," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS '12. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, p. 196–201.

[18] IBM, "OpenCAPI (Open Coherent Accelerator Processor Interface)," 2014. [Online]. Available: https://docs.kernel.org/userspace-api/accelerators/ocxl.html

[19] G.-Z. Interconnect, "Gen-z zmmu and memory interleave,," Gen-Z Consortium, Tech. Rep., July 2017. [Online]. Available: https://genzconsortium.org/wp-content/uploads/2018/05/Gen-Z-MMUand-Memory-Interleave.pdf

[20] 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, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 574–587. [Online]. Available: https://doi.org/10.1145/3575693.3578835

[21] 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 Comp Architecture (HPCA)*. IEEE, 2012, pp. 1–12.

[22] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020.

[23] S.-L. Lu, T. Karnik, G. Srinivasa, K.-Y. Chao, D. Carmean, and J. Held, "Scaling the "memory wall"," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 271–272. [Online]. Available: https://doi.org/10.1145/2429384.2429437

[24] 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, Volume 3*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 742–755. [Online]. Available: https://doi.org/10.1145/3582016.3582063

[25] J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," University of Virginia, Charlottesville, Virginia, Tech. Rep., 1991-2007, a continually updated technical report. http://www.cs.virginia.edu/stream/. [Online]. Available: http://www.cs.virginia.edu/stream/

[26] ——, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[27] MICRON, "Famfs Shared Memory Filesystem Framework - User Space Repo," Tech. Rep., 2024. [Online]. Available: https://github.com/cxl-micron-reskit/famfs

[28] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. 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.

[29] H. Nguyen and J. Lowe-Power, "gem5/sst integration 2021: Scaling full-system simulations." [Online]. Available: https://www.gem5.org/events/isca-2022#:~:text=gem5/SST%20Integration%202021%3A%20Scaling%20Full%2Dsystem%20Simulations

[30] A. Puri, K. Bellamkonda, K. Narreddy, J. Jose, V. Tamarapalli, and V. Narayanan, "Dracksim: Simulating cxl-enabled large-scale disaggregated memory systems," in *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 3–14. [Online]. Available: https://doi.org/10.1145/3615979.3656059

[31] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, p. 37–42, mar 2011. [Online]. Available: https://doi.org/10.1145/1964218.1964225

[32] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[33] D. D. Sharma, "Compute express link," *CXL Consortium White Paper*, 2019.

[34] ——, "Compute express link®: An open industry-standard interconnect enabling heterogeneous data-centric computing," in *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2022, pp. 5–12.

[35] D. D. Sharma, R. Blankenship, and D. S. Berger, "An introduction to the compute express link (cxl) interconnect," 2024.

[36] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong, R. Wang, J. H. Ahn, T. Xu, and N. S. Kim, "Demystifying cxl memory with genuine cxl-ready systems and devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 105–121. [Online]. Available: https://doi.org/10.1145/3613424.3614256

[37] N. Tampouratzis, I. Papaefstathiou, A. Nikitakis, A. Brokalakis, S. Andrianakis, A. Dollas, M. Marcon, and E. Plebani, "A novel, highly integrated simulator for parallel and distributed systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 17, no. 1, pp. 1–28, 2020.

[38] L. Wang, X. Zhang, S. Wang, Z. Jiang, T. Lu, M. Chen, S. Luo, and K. Huang, "Asynchronous memory access unit: Exploiting massive parallelism for far memory access," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 3, Sep. 2024. [Online]. Available: https://doi.org/10.1145/3663479

[39] Y. Wang, L. Wu, S. Gao, Y. Tang, J. Luo, Z. Wang, Y. Ou, D. Dong, N. Xiao, and M. Lai, "Cohet: A cxl-driven coherent heterogeneous computing framework with hardware-calibrated full-system simulation," 2026. [Online]. Available: https://arxiv.org/abs/2511.23011

[40] Y. Wang, L. Wu, W. Hong, Y. Ou, Z. Wang, S. Gao, J. Zhang, S. Ma, D. Dong, X. Qi, M. Lai, and N. Xiao, "Cxl-dmsim: A full-system cxl disaggregated memory simulator with comprehensive silicon validation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 45, no. 4, pp. 1787–1801, 2026.

Appendix

1. Abstract

CXL-ClusterSim is a full-system modeling and simulation framework for CXL-based disaggregated memory clusters. It couples the gem5 simulator, which provides detailed, full-system-fidelity timing models for individual compute hosts (CPU, cache hierarchy, OS), with the Structural Simulation Toolkit (SST), which drives multiple gem5 host instances as MPI processes and models the shared, disaggregated "remote memory" device/pool that those hosts access over a CXL.mem-like interconnect. The artifact comprises the full source (a gem5 fork augmented with disaggregated-memory boards,cache-hierarchies, memory components in `disaggregated_memory/`, and a gem5 - SST bridge plus SST component configuration scripts in `ext/sst/`). Together with a single launcher script and a set of JSON "joblists" that reproduce the paper's memory-pooling experiments (STREAM bandwidth under different NUMA policies, mixed-ISA pooling across ARM and RISC-V hosts, NAS Parallel Benchmarks) and memory-sharing experiment (GAPBS with a single-writer/multiple-reader access pattern). This submission targets the **Available** badge: the artifact is publicly hosted, archived with a persistent identifier, and documented sufficiently to support reproducing the paper's setup.

2. Artifact check-list (meta-information)

- **Algorithm:** N/A (full-system cycle-level/event-driven simulator, not an algorithmic artifact).
- **Program:** gem5 (C++/Python, full-system CPU + memory hierarchy simulator); SST-Core and SST-Elements (C++/Python, parallel discrete-event simulation engine and `memHierarchy` components); guest workloads: STREAM, NAS Parallel Benchmarks (NPB), GAPBS.
- **Compilation:** SCons (gem5); GNU Autotools + `make` (SST-Core, SST-Elements); GNU `make` (the gem5↔SST bridge library in `ext/sst/`).
- **Transformations:** N/A.
- **Binary:** Not distributed; built from source (`gem5.opt`, `libgem5_opt.so`, `bin/sst`, `libgem5.so` bridge).
- **Model:** N/A.
- **Data set:** None beyond guest boot media (Linux kernel + disk image), built separately; see "Data sets" below.
- **Run-time environment:** Linux; Python 3; OpenMPI; no root/container requirements beyond KVM capable (requires both X86 and ARM hosts) device access for fast-forward boot.
- **Hardware:** X86_64 and ARM hosts; KVM virtualization support strongly recommended (used to fast-forward guest boot); a multi-core host is recommended since SST launches `(hosts + 1)` (up to 17 threads) MPI processes per run.
- **Run-time state:** None.
- **Execution:** Command line:

```
python3 disaggregated_memory/unified_run.py \
  --count=<num-hosts> \
  --exp-name=<output-dir> \
  --joblist=<path/to/a/JSON/config/script>
```

Listing 1: Command line parameters

The joblist for all the experiments in the paper are located in `disaggregated_memory/joblist` directory.

- **Metrics:** Memory bandwidth (STREAM); IPC / execution time (NPB); throughput (GAPBS). Reported via gem5's per-host `stats.txt` and SST's `sst-output.txt`.
- **Output:** gem5 statistics files (one per simulated host) and an SST statistics file per experiment directory.
- **Experiments:** Memory pooling under varying remote-memory latency and NUMA policy (STREAM); mixed-ISA pooling (ARM + RISC-V); multiprogrammed NPB with a memory-stranding case study; memory sharing across hosts (GAPBS).
- **How much disk space required (approximately)?:** 21 GiB
- **How much time is needed to prepare workflow (approximately)?:** For building on a 96-core X86-based AMD EPYC system, it approximately takes about 30 minutes to build gem5, libgem5_opt.so, SST-Core, SST-Elements and the gem5-SST bridge. Additional 60 minutes are needed to build the Linux kernels and prepare the gem5 disk images for full-system workloads.
- **How much time is needed to complete experiments (approximately)?:** For the STREAM experiments, it typically takes from 2 to up to 16 hours until completion. NPB workloads were evaluated on an ARM Neoverse machine, which takes up to a week. GAPBS with shared memory graphs takes 14 hours to complete. KVM-capable CPUs are highly recommended as every experiment boots with systemd enabled and NPB/GAPBS work with data sizes of > 10 GiB.
- **Publicly available?:** Yes: https://github.com/darchr/cxl-clustersim.
- **Code licenses (if publicly available)?:** BSD 3-Clause "New" or "Revised" License (inherited from the upstream gem5 project this repository extends).
- **Data licenses (if publicly available)?:** N/A – guest kernel/disk images are not bundled in this repository; they are built separately per the instructions in `README-DM.md`.
- **Workflow automation framework used?:** None; a single Python launcher script (`unified_run.py`) drives both simulator phases.
- **Archived (provide DOI)?:** 10.5281/zenodo.21844298

3. Description

3.1. How to access. The artifact is the `cxl-clustersim` repository is upstreamed on Github (branch: `iiswc-artifact`). This repository is a fork of gem5; the code specific to this paper is confined to two directories: (a) `disaggregated_memory/` (gem5-side composable-memory boards, cache hierarchies, and the `unified_run.py` launcher), and (b) `ext/sst/` (the gem5 - SST bridge and SST-side component/topology configuration scripts, under `ext/sst/sst/`). All other directories are the unmodified upstream gem5 tree. A DOI-archived snapshot is available at 10.5281/zenodo.21844298

3.2. Hardware dependencies. Experiments were conducted on ARM and X86 machine. KVM virtualization support (`/dev/kvm`) is strongly recommended to fast-forward guest boot in reasonable time. Without it, boot must fall back to gem5's atomic CPU model, which is substantially slower. A multi-core host is required: SST launches up to 17 MPI processes (`(number of hosts (16) + 1)`). Approximately 512 GiB of system memory is recommended.

3.3. Software dependencies. Linux; Python 3.8+ with the packages listed in gem5's "requirements.txt"; SCons; GNU Autotools and `make`; a C++17 compiler; OpenMPI; SST-Core and SST-Elements v14.0.0 (built from source, see `ext/sst/INSTALL.md`). No containerized environment is provided; all dependencies are installed directly on the host.

3.4. Data sets. None are bundled. Guest boot media (a Linux kernel and a disk image with NUMA-aware userspace tooling and the STREAM/NPB/GAPBS binaries installed) must be built separately. Disk images and the kernels can be compiled via:

```
git clone https://github.com/kaustav-goswami/
    gem5-resources
cd gem5-resources
git checkout disaggregated
cd src/stream
./build-x86.sh 24.04
cd ../src/shared-gapbs
./build-x86.sh 24.04
cd ../src/npb-cxl
./build-arm.sh 24.04
cd ../kernels
# see README.md on how to build the kernel
git clone https://git.kernel.org/pub/scm/linux/
    kernel/git/stable/linux.git
cd linux
git checkout v6.9.9
cp ../linux-configs/config.x86.6.9.9 .config
make -j`nproc`
```

Listing 2: Creating gem5-resources

3.5. Models. N/A.

4. Installation

CXL-ClusterSim can be compiled via:

```
git clone https://github.com/darchr/cxl-
    clustersim.git
cd cxl-clustersim
git checkout iiswc-artifact
scons build/ALL/gem5.opt -j16
scons build/ALL/libgem5_opt.so -j16 \
  --without-tcmalloc --duplicate-sources
cd ext/sst
export SST_CORE_HOME=`pwd`
wget https://github.com/sstsimulator/sst-core/
    releases/download/v14.0.0_Final/sstcore
    -14.0.0.tar.gz
tar xf sstcore-14.0.0.tar.gz
cd sstcore-14.0.0
./configure --prefix=$SST_CORE_HOME --with-
    python=/usr/bin/python3-config
make all -j16 && make install
cd ..
wget https://github.com/sstsimulator/sst-
    elements/releases/download/v14.0.0_Final/
    sstelements-14.0.0.tar.gz
tar xf sstelements-14.0.0.tar.gz
cd sst-elements-library-14.0.0
./configure --prefix=$SST_CORE_HOME --with-
    python=/usr/bin/python3-config \
```

```
    --with-sst-core=$SST_CORE_HOME
make all -j16 && make install
cd ..
export PKG_CONFIG_PATH=`pwd`/lib/pkgconfig
mv Makefile.linux Makefile
make ARCH=ALL -j4
```

Listing 3: Compiling CXL-ClusterSim

5. Experiment workflow

Every experiment follows the same two-phase workflow, orchestrated by `disaggregated_memory/unified_run.py`: (1) each simulated host boots independently under plain gem5 (KVM or atomic-fast-forwarded) up to the start of its region of interest, where a checkpoint is taken; (2) `unified_run.py` then launches all hosts' checkpoints together under SST via `mpirun`, restoring each host into a timing-accurate CPU model while SST simulates the shared remote-memory device/pool and the interconnect between hosts and that pool. A JSON "joblist" (one entry per host) fully describes each host's CPU/cache/memory configuration, remote-memory address range and link latency, and boot workitem; sample joblists reproducing every experiment in the paper are provided under `disaggregated_memory/joblist/`.

6. Evaluation and expected results

Each experiment reports gem5/SST statistics (memory bandwidth for STREAM, IPC/simSeconds for NPB, IPC for GAPBS) that should reproduce the trends shown in the paper's corresponding figures (memory-pooling bandwidth vs. NUMA policy and remote-memory latency; mixed-ISA pooling; NPB memory-stranding case study; GAPBS

- `ext/sst/sst/unified_sst.py` is the topology used to produce every result in the paper and is the well-exercised path for this artifact.

memory sharing). Exact reproduction of published values is not expected to be bit-identical across different host CPUs, kernel versions, or SST/gem5 patch levels; qualitative trends and relative comparisons between configurations are the intended reproducibility target for this artifact.

7. Experiment customization

New configurations are expressed purely as joblist JSON files and no code changes are required to vary the number of hosts, per-host ISA (a mix of ARM, RISC-V, and x86 hosts is supported in the same run), cache hierarchy, local/remote memory sizes, remote-memory latency, or whether remote memory is pooled (disjoint per-host ranges) or shared (multiple hosts mapped to the same range). `README-DM.md` documents the full joblist schema and how to add new host- or device-side simulation logic in gem5 or SST respectively.

8. Notes

SST-side scripts are split into three configurations to simulate direct links and different network topologies:

- `unified_sst_router.py` is used to simulate a single hop star topology.
- `unified_sst_router_tree.py` is used to simulate a tree topology.

The network script is automatically selected from the top script, *i.e.* `unified_run.py` script and can load a checkpoint taken with a different topology script. The repository's `README-DM.md`'s "Known Limitations" section documents this and one other known functional/timing race in gem5's block-device model. CXL-ClusterSim is an actively developed research fork of gem5. We expect the repository to be updated frequently.

9. Methodology

This submission only targets the "Available" badge.