

CXLSim: A Simulator for CXL Memory Expander

Seongbeom Kim^{†‡}, Jintaek Kang[†], Kyunghoon Kim[†], Seungwook Lee[†], and Beomseok Nam[‡]
 Samsung Electronics[†], Sungkyunkwan University[‡]

Abstract—Compute Express Link (CXL), a cache-coherent interconnect protocol for CPUs and PCIe devices, has emerged as a promising solution for memory expansion. In this work, we develop a CXL simulator built upon GEM5 and QEMU to assess the system-wide performance impacts based on varying attributes of CXL devices, such as memory capacity and latency. The simulator directs memory traffic to either system memory or the CXL memory expander, tracks cache hit/miss ratios, and projects application performance. Our simulation study demonstrates that the simulator accurately predicts application performance.

Index Terms—CXL, memory pool, software simulator

I. INTRODUCTION

Data centers have been experiencing a growing demand for memory-intensive tasks [1]–[8]. However, the scalability of the memory bus remains limited. To address this challenge, CXL (Compute Express Link), a cache-coherent interconnect protocol for CPUs and PCIe devices, has emerged as a new solution for memory expansion [9].

Currently, various semiconductor companies are actively developing CXL prototype devices in accordance with the CXL standard. However, it is not known how CXL devices will affect the performance of applications, as not all design aspects that could influence performance are detailed in the CXL standard.

Simulation studies are essential when evaluating the intricacies of CXL-based systems and the performance demands of applications. By simulating design choices before actual device fabrication, it ensures the design functions as desired, allowing for the identification and correction of any possible issues or deficiencies. Specifically, making changes to designs after manufacturing actual devices can lead to significant expenses for system designers or application users seeking adjustments. For instance, the lack of a precise simulator for Intel’s Optane DC Persistent Memory (DCPM) led to multiple challenges after the device had been released to the public, with issues such as those arising from XPLine granularity and high read latency. As a result, DCPM was not successful in its business.

Samsung’s latest CXL Memory Expander uses an x8 PCIe 5.0 interface to connect to the CPU. This early CXL prototype device indicates an increase in latency by approximately 200–250 nanoseconds, which is similar to the latency of DCPM. While Intel discontinued its business, DCPM demonstrated the potential of byte-addressable tiered memory systems, making the latency of a few hundred nanoseconds negligible for various applications [10], [11]. Analyzing the impact of capacities and latencies of tiered memory systems on the performance of applications is important for memory manufacturers. However,

CXL simulators such as Flight Simulator [12] do not provide functionality to control latencies of tiered memory systems, and simulators such as CXLMemSim [13] use the NUMA effect to increase the latency of CXL memory. This emulation approach is limited in that it does not offer a wide range of latency variations.

In this study, we develop *CXLSim*, a CXL simulator built upon GEM5 and QEMU, neither of which currently supports CXL memory expander capabilities. The main goal of *CXLSim* is to assess the system-wide performance impacts as the attributes of CXL devices, particularly in terms of memory capacity and latency, vary. *CXLSim* allows memory traffic to be directed to either system memory or the CXL memory expander. It also tracks cache hit/miss ratios and projects application performance. Our simulated results demonstrate that the simulator can accurately predict application performance.

II. BACKGROUND

A. CXL and Memory Expander

CXL (Compute Express Link) is an open standard for CPU-to-device and CPU-to-memory interconnect built on the PCIe physical layer. CXL provides three sub-protocols for the connection between the host CPU and connected devices (such as accelerators, memory expansion devices, etc.), namely *CXL.io*, *CXL.cache*, and *CXL.mem*. *CXL.io* is used for discovering, enumerating, configuring, and managing CXL devices connected to the CXL host. *CXL.cache* is used when a CXL device accesses processor memory, while *CXL.mem* is used when the processor accesses CXL device memory.

The CXL devices are categorized into three types. Type 1 devices are accelerators or SmartNICs. Type 1 CXL devices do not have device memory. Instead, they access the host memory via *CXL.cache* transactions, maintaining coherency between host memory and device-local cache. Type 2 devices are accelerators such as GPUs or FPGAs that have host managed device memory (HDM). Similar to Type 1 devices, they can directly access host memory. Type 3 CXL devices are memory expansion devices allowing host processors to access them cache coherently through *CXL.mem* transactions. CXL Type 3 devices could be used for memory density and bandwidth expansion.

In this study, we focus on Type 3 memory expanders, which do not consider cache coherence. Typically, increasing a system’s memory size and bandwidth requires adding more CPU memory channels, leading to higher costs and greater engineering complexity. However, CXL Type 3 allows for increasing memory size and bandwidth without adding memory channels. This approach introduces higher latency compared to

memory directly connected to the host CPU. Memory directly connected to the CPU has a latency of 80 to 140 ns, whereas CXL memory has a latency of 170 to 250 ns. Users of CXL memory expanders are expected to employ them in scenarios where the benefits of significantly increased memory size outweigh the increased latency. Consequently, there is a need for a simulator that can finely adjust the latency of CXL links, routers, and other components.

B. Processor Support for CXL

To use CXL devices, a processor that supports CXL is required. Sapphire Rapids and Genoa, respectively, are the code names for Intel’s 4th generation processor and AMD’s next-generation processor, both of which support the CXL protocol. For accurate simulation of CXL, it is crucial to provide the simulator with as much precise architectural information, such as *FLIT* (Flow Control Unit), as possible. The CXL.cache and CXL.mem protocols share common link and transaction layers, utilizing a fixed-width 528-bit (66-byte) FLIT. Each FLIT consists of four 16-byte data slots and a two-byte CRC, transmitting data across the Cache/Mem Protocol Interface (CPI) in one clock cycle. FLIT ensures low latency and efficient bandwidth use, supports error detection and recovery, and is compatible with various communication protocols. Accurate simulation of FLIT is crucial for CXL.

C. SMDK for CXL

The Scalable Memory Development Kit (SMDK) is specifically designed for CXL memory expander devices, providing a comprehensive software-defined memory system. It serves as an intermediate layer between applications and hardware, supporting various CXL usage scenarios. SMDK achieves transparent memory management by extending the Linux process/virtual memory manager design. Without SMDK, the simulator would need to rewrite everything from scratch, including device drivers, tiered memory management, memory management policies, and memory usage by applications. Additionally, it is worth noting that the policy for using CXL devices is also implemented within SMDK.

D. Gem5 and Garnet

The reliability of simulation results is the most crucial factor in selecting a simulator. Gem5 is renowned for its highly reliable results and is consequently one of the most widely used simulators. Gem5 accurately models and simulates computer architecture behavior across various levels, from individual instructions to full systems.

Gem5 provides Garnet, a network-on-chip (NoC) model, which includes a cycle-accurate micro-architectural implementation of an on-chip network router. In particular, Garnet supports FLIT-based communication, allowing simulation of data-divided transfers. This capability allows for the dynamic adjustment of latency in CXL switches and links. CXL switching enables multiple CXL devices to be connected to a host processor, with each device being able to connect to multiple host processors. Additionally, CXL switches can be

configured with multiple levels, allowing for the construction of hierarchical trees, meshes, or ring topologies. Therefore, the simulator should be capable of adjusting the switching latency of CXL. Ideally, latency should be adjusted separately for each switch and link. However, since the architecture of Sapphire Rapid is not publicly available, CXLSim does not distinguish between switches and links, applying a single latency.

III. CXLSIM DESIGN AND IMPLEMENTATION

CXLSim is a trace-driven CXL simulator. Trace-driven simulation is widely used because it allows for iterative evaluation of various architecture designs using the same input traces obtained from a reference system.

In this section, we first explain how to extract memory access traces from workloads, and then describe how CXLSim simulates the performance of a system equipped with CXL memory expander.

A. Memory Access Trace

CXLSim utilizes QEMU to extract memory access traces. QEMU is an emulator that supports various architectures, including x86, ARM, and others. It is commonly paired with Linux’s Kernel-based Virtual Machine (KVM). KVM operates directly on hardware, allowing QEMU-KVM to access and utilize actual memory devices without the need for additional virtualization layers.

Memory Access Time: To extract memory access traces, we modified the `glue()` function within the `memory_ldst.c.inc` file of QEMU. The `glue()` function facilitates the access and validation of values at memory addresses. Therefore, the code inserted into the `glue()` function measures the memory access time, denoted as A_i in Figure 1.

Computation Time: By subtracting the total memory access time from the overall execution time, we obtain the computation time, excluding memory access, i.e., $\sum C_i = T_{QEMU} - \sum A_i$, where C_i represents the computation time.

Cache Effect: Unfortunately, the memory access times measured in QEMU differ from actual memory access times. This discrepancy arises because CPU cache effects are not accurately reflected in the QEMU trace. That is, cache misses occurring during QEMU emulation are captured while cache misses from the workload are not accounted for.

The total execution time T_{ref} measured in the CXL reference system is larger than the time measured in QEMU. This difference can be attributed to two factors: (i) overhead generated during QEMU emulation and (ii) the lack of reflection of cache misses in memory access times, resulting in longer memory access times.

To consider cache effects, we multiply the total memory access time by the cache miss rate (X) and add the computation time to estimate the total time, i.e., $T_{GEM5} = \sum C_i + \sum A_i \cdot X$. We adjust the cache miss rate until the time matches the total execution time in the reference system. We note that if the cache miss ratio exceeds 0.03, the execution time in QEMU exceeds that in the reference system. Therefore, we vary the cache miss ratio from 0.01, 0.02, to 0.03.

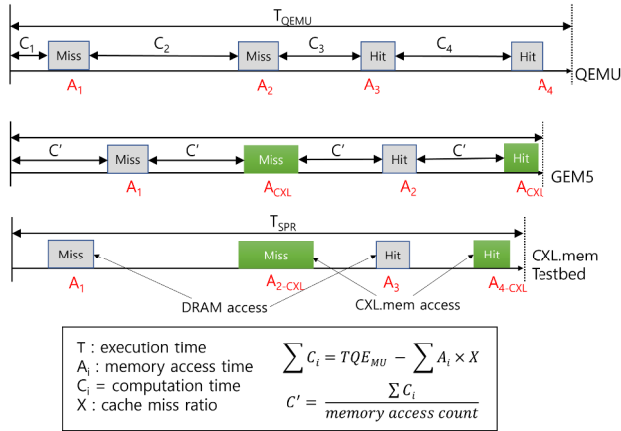


Fig. 1. Memory Traffic Injection

B. Adjusting Simulation Parameters Using Reference System

CXL memory latency: QEMU currently does not support CXL memory extension. As a result, the memory trace obtained from QEMU does not distinguish between CXL memory accesses and host memory accesses.

To obtain memory access statistics, including data stored in CXL memory expanders during application execution, we analyzed memory access statistics on a reference Sapphire Rapids machine equipped with an actual CXL memory expander. We measured the ratio of memory traffic into CXL and DRAM using `numastat`, which provides statistics about memory usage, page allocation, etc., for each NUMA node. By configuring DRAM and CXL memory expanders on separate nodes and utilizing `numastat`, we obtained CXL memory usage statistics.

C. Architecture Modeled by CXLSim

The architecture of the Sapphire Rapids system, modeled by CXLSim, consists of N nodes sharing one memory expander per system board. Each node in this architecture comprises four chiplets, each containing N CPUs and one DMA controller. It's important to note that the number of CPUs significantly affects simulation time, so increasing the number of CPUs requires careful consideration.

D. Running Simulation

To ensure that simulations using CXLSim produce results consistent with experiments using the actual Sapphire Rapids board and CXL memory expander, we obtained the workload execution time and the ratio of CXL to DRAM usage from the real reference system.

After measuring the execution time and the CXL/DRAM ratio, and collecting memory traffic information from QEMU, we provide this data to GEM5 for simulation. When processing memory access traces, GEM5 determines memory access latency based on the probability of CXL memory access.

The number of concurrent memory access requests also significantly impacts simulation results. While the maximum

number of concurrent memory access requests varies depending on the size of the reorder buffer, the average number of concurrent memory access requests depends on the characteristics of the application. Therefore, the average number of concurrent memory access requests should be experimentally measured and used for each individual application workload.

IV. EXPERIMENTS

In this section, we evaluate the accuracy of CXLSim in simulating the performance effects of a real CXL memory expander using two memory-intensive applications. We also evaluate how application performance changes as the latency of the CXL memory expander varies. The objective of this simulation study is to determine whether applications can still benefit from the CXL memory expander despite increases in the latency of tiered memory.

We run experiments on a server that features two Intel Xeon Platinum 8468 CPUs (2.10GHz, 48 cores) from the Intel Sapphire Rapids ES2/QS family, along with Samsung DDR5 DIMMs (2x 64GB) operating at 4800MT/s. Additionally, the server features a Samsung 128GB CXL memory expander FPGA prototype to expand memory capacity. The server runs the Ubuntu 20.04 LTS operating system with the SMDK kernel version 5.17.0-rc5-smdk.

For the workloads, we run the Memtier benchmark using the memory-intensive Memcached, which stores 60 GB of data entirely in memory. We set the number of threads to 24 and the number of clients to 50. For the less memory-intensive Cassandra, which stores data both in memory and on disks, we run the YCSB Load workload.

In the experiment shown in Figure 2, we observed that 65% of the data accessed by Memcached in the CXL reference system was allocated in DRAM, while the remaining 35% was allocated in the CXL memory expander. Furthermore, when performing the YCSB Load workload using Cassandra on the same CXL reference system, we observed that Cassandra had much lower memory access frequency compared to Memcached, accessing approximately 22% of in-memory data relative to the total memory accesses of Memcached.

We calibrated the CXLSim using this DRAM/CXL access ratios for each workload. It should be noted that the usage ratio of DRAM and CXL memory measured through `numastat` includes not only the workload but also the usage from background applications and the operating system, leading to a higher proportion of DRAM usage.

In the experiments shown in Figure 3, we vary the latency of the CXL memory expander to 170, 250, and 600 nsec. The latency of the Samsung CXL memory expander FPGA prototype is 600 nsec based on actual measurements, while the latency of the Samsung CXL memory expander ASIC prototype is expected to be around 170 nsec.

When all memory accesses were directed to DRAM (i.e., CXL Memory Ratio = 0%), Memcached takes about 1 second to perform the workload. When 65% of memory accesses are directed to CXL.mem (i.e., CXL Memory Ratio = 65%), as measured in Figure 2, the reference system with FPGA CXL

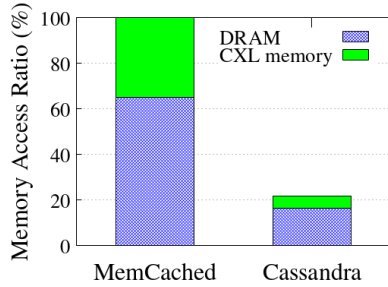


Fig. 2. CXL memory/DRAM ratio

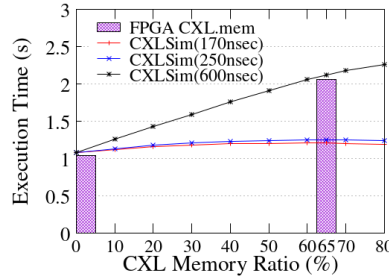


Fig. 3. Memcached Simulation result

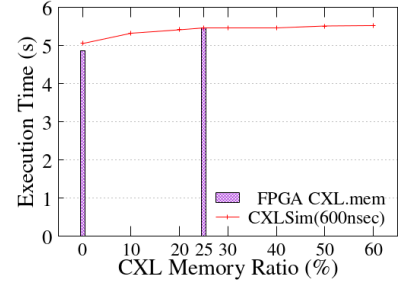


Fig. 4. Cassandra Simulation result

memory takes about 2 seconds. When aligning the CXL.mem latency in CXLSim with the FPGA performance of 600 nsec, the obtained curve matches the actual execution time. After calibrating the CXLSim, we vary the CXL.mem latency. When aligning the CXL.mem latency with the ASIC performance, even if 65% or 80% of memory accesses are directed to CXL.mem, the execution time only increases by less than 10%.

In Figure 4, we use the similarly calibrated CXLSim parameters to measure the performance of another application, Cassandra. As in Figure 2, we adjust the ratio of DRAM accessed by Cassandra to the CXL memory. When all memory accesses were directed to DRAM (i.e., CXL Memory Ratio = 0%), the simulator measures an execution time of 5.044 seconds. In comparison, it actually takes 5.13 seconds on the reference CXL memory system, yielding an error margin of approximately 1.7%. When 25% accesses were directed to CXL memory, the simulator measures an execution time of 6.278 seconds, while the reference system actually spends 6.144 seconds. This results in an error margin of about 2%. In summary, CXLSim simulator can accurately predict the performance of application processes running on the actual reference system with very low error.

V. CONCLUSION

In this work, we developed a CXL simulator using QEMU and Gem5. Leveraging Gem5, the simulator accurately replicates the attributes of CXL architecture, such as FLIT and Garnet, and assesses how application performance fluctuates with varying CXL.mem latencies. Through empirical experiments, we verified how the performance of memory-intensive applications changes relative to the access ratio between CXL and DRAM memory, yielding results consistent with those from real reference systems.

However, CXLSim is subject to several limitations that necessitate future improvements. Currently, it solely accommodates the CXL memory expander (CXL.mem). Expanding its scope to encompass CXL.io and CXL.cache—representing storage, accelerators, and switches—is essential. Additionally, a significant drawback lies in CXLSim’s reliance on execution time data from real-world CXL systems for precise application calibration. Addressing these shortcomings will demand substantial research efforts to enhance the simulator’s efficacy.

REFERENCES

- [1] C. Delimitrou and C. Kozyrakis, “Quasar: Resource-Efficient and QoS-Aware Cluster Management,” in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages (ASPLOS)*, 2014, p. 127–144.
- [2] J. Wang and M. Balazinska, “Elastic Memory Management for Cloud Data Analytics,” in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2017, pp. 745–758.
- [3] C. A. Waldspurger, “Memory Resource Management in VMware ESX Server,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 181–194, 2002.
- [4] M. J. Feeley, W. E. Morgan, E. P. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath, “Implementing Global Memory Management in a Workstation Cluster,” in *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, 1995, p. 201–212.
- [5] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, “Disaggregated Memory for Expansion and Sharing in Blade Servers,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, 2009, p. 267–278.
- [6] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, “System-level Implications of Disaggregated Memory,” in *Proceedings of the 18th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2012, pp. 1–12.
- [7] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, “Efficient Memory Disaggregation with Infiniswap,” in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 649–667.
- [8] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker, “Can Far Memory Improve Job Throughput?” in *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*, 2020.
- [9] “Compute Express Link CXL Latency How Much is Added at HC34,” <https://www.servethehome.com/compute-express-link-cxl-latency-how-much-is-added-at-hc34/>.
- [10] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, “Better i/o through byte-addressable, persistent memory,” in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 133–146. [Online]. Available: <https://doi.org/10.1145/1629575.1629589>
- [11] A. Baldassin, J. Barreto, D. Castro, and P. Romano, “Persistent Memory: A Survey of Programming Support and Implementations,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–37, 2021.
- [12] “Flight Simulator,” <https://memverge.com/memverge-and-sk-hynix-accelerate-memory-pooling-and-sharing-software-development-with-cxl-flight-simulator/>.
- [13] Y. Yang, P. Safayenikoo, J. Ma, T. A. Khan, and A. Quinn, “CXLMemSim: A pure software simulated CXL.mem for performance characterization,” in *The fifth Young Architect Workshop (YArch’23)*, Mar. 2023.