CXLMemSim: A pure software simulated CXL.mem for performance characterization

Yiwei Yang, Pooneh Safayenikoo
University of California, Santa Cruz
Santa Cruz, California
{yyang363,psafyen}@ucsc.edu

Jiacheng Ma, Tanvir Ahmed Khan
University of Michigan
Ann Arbor, Michigan
{jcma,takh}@umich.edu

Andrew Quinn
University of California, Santa Cruz
Santa Cruz, California
aquinn1@ucsc.edu

ABSTRACT

The emerging CXL.mem standard provides a new type of byte-addressable remote memory with a variety of memory types and hierarchies. With CXL.mem, multiple layers of memory—e.g., local DRAM and CXL-attached remote memory at different locations—are exposed to operating systems and user applications, bringing new challenges and research opportunities. Unfortunately, since CXL.mem devices are not commercially available, it is difficult for researchers to conduct systems research that uses CXL.mem.

In this paper, we present our ongoing work, CXLMemSim, a fast and lightweight CXL.mem simulator for performance characterization. CXLMemSim uses a performance model driven using performance monitoring events, which are supported by most commodity processors. Specifically, CXLMemSim attaches to an existing, unmodified program, and divides the execution of the program into multiple epochs; once an epoch finishes, CXLMemSim collects performance monitoring events and calculates the simulated execution time of the epoch based on these events. Through this method, CXLMemSim avoids the performance overhead of a full-system simulator (e.g., Gem5) and allows the memory hierarchy and latency to be easily adjusted, enabling research such as memory scheduling for complex applications. Our preliminary evaluation shows that CXLMemSim slows down the execution of the attached program by 4.41x on average for real-world applications.

1 INTRODUCTION

DRAM is the dominant cost in building modern servers. To reduce cost, data-center operators strive to limit the quantity of DRAM required across their fleet of machines. Unfortunately, the traditional approach of deploying DRAM to be used by a single local machine wastes memory due to *memory stranding*, which refers to local memory on a machine that goes unused even while the machine is in use. A major cloud provider found that up to 25% of DRAM in their data centers is stranded [9].

CXL.mem offers a solution to the memory stranding problem. A data-center operator can use CXL.mem to move memory from servers into pools shared by multiple servers. Each server uses the load/store interface to access memory pools, which is more efficient than past shared memory pool designs (e.g., RDMA).

However, CXL memory pools are less efficient than local DRAM with respect to bandwidth and latency. Moreover, depending on the CXL.mem pool topology (i.e., the layout of which memory pools use which pools), CXL-attached memory may induce congestion or coherency performance problems.

Also, researchers cannot evaluate CXL.mem's performance impact because the devices are not commercially available. So, the research community needs a CXL.mem simulator. Unfortunately, existing simulators are insufficient for the task. Architectural simulators (e.g., Gem5 [2]) are too slow to analyze the applications that will use CXL.mem pools. Existing memory pooling solutions (e.g., those using RDMA) overestimate the overhead of CXL.mem since they do not access shared memory through the load/store interface. Finally, existing tiered memory deployments, such as NUMA [9] or NVM [11], cannot model many CXL.mem topologies.

In this project, we propose CXLMemSim, our ongoing work to accurately evaluate real-world applications that use CXL.mem pools. CXLMemSim executes an unmodified application on a traditional server, traces the performance events in the application, and inserts delays in the application to simulate the latency and bandwidth of a user-provided CXL.mem topology. Using existing hardware enables CXLMemSim to support a load/store interface with CXL.mem pools and to impose low overhead so that it can evaluate real-world applications on a variety of topologies.

CXLMemSim will also allow systems designers to evaluate new research. For example: the system will enable a comparison of software and hardware memory prefetching and migration. It will enable comparison of cache-line and page memory management. Finally, CXLMemSim will allow evaluation of the performance impact of CXL.mem pool coherency on applications that share memory across multiple servers.

2 CXL BACKGROUND

CXL is a set of protocols that operate over the serial PCIe bus to connect peripheral devices to host processors. CXL consists of three protocols, CXL.io, CXL.cache, and CXL.mem, which allow host processors to communicate with I/O devices, accelerators, and external memory, respectively. We focus on CXL.mem.

From the perspective of a host processor, a CXL.mem memory pool behaves equivalent to host memory. The host can issue load/-store instructions to the memory pool. The hosts can cache data from CXL-attached memory in their processor data caches; the CXL.mem protocol provides coherency across devices that cache data from the same CXL.mem memory pool.

CXL.mem's key hardware components are as follows. Hosts connect to CXL.mem memory pools using a CXL Root Complex (RC). Each RC can either connect to memory pools directly or through a CXL switch. A CXL switch allows a host to connect to multiple memory pools through the same link.

CXL switches allow a data center operator to deploy a variety of CXL.mem topologies. For example, Figure 1 shows a topology consisting of two network switches and three memory pools; we

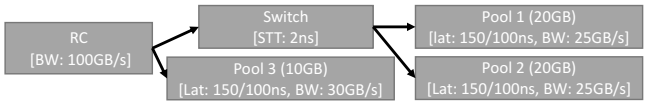

Figure 1: An example topology

annotate the bandwidth (BW), latency (Lat), and serial transmission time (STT) for each pool, switch, and the RC. Choosing a topology requires balancing memory stranding benefits with performance degradation. Memory pools that support more hosts decrease memory stranding but increase performance overhead since attaching more hosts to a pool requires employing a hierarchy of CXL switches. Moreover, each CXL switch can cause congestion, when multiple hosts use the switch at the same time, and limit bandwidth, when hosts exceed the bandwidth of the switch.

3 PROPOSED DESIGN

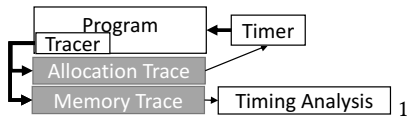

Figure 2: A system diagram of CXLMemSim

Figure 2 is the system diagram of CXLMemSim. It simulates CXL.mem on commodity hardware by attaching itself to a running user program. Its three components are: 1) a *Tracer*, 2) a *Timer*, and 3) a *Timing Analyzer*.

1–Tracer. CXLMemSim traces the memory operations of a program in two ways. First CXLMemSim uses eBPF [5, 8, 14] to trace memory allocation operations (e.g., munmap, sbrk and brk), which enables tracing memory allocation without modifying the program and allows simulating closed-source programs. Second, CXLMemSim uses performance counters (e.g., Intel PEBS [1, 6, 13]) to trace memory events (e.g., LLC misses and L2 stalls).

2–Timer. CXLMemSim divides the execution of the attached user program into *epochs* and sets up an epoch timer that periodically interrupts the attached program. When the program is interrupted, CXLMemSim uses the allocation trace to determine the corresponding memory pool of each memory access.

3–Timing Analyzer. While the program is paused, CXLMemSim uses the memory trace to calculate three types of timing delays that should be added to the execution time of each epoch: 1) latency delay, 2) congestion delay, and 3) bandwidth delay. CXLMemSim calculates the latency delay by multiplying the number of memory operations to each memory pool by the difference between the latency of the target memory pool and the latency of local DRAM. Then, CXLMemSim calculates the congestion delay by iterating over the memory trace to find events that use the same switch within a smaller interval than the switch's serial transmission time (STT); once such events are found, CXLMemSim injects the necessary delays. Finally, CXLMemSim determines the bandwidth delays. For each switch in the topology, CXLMemSim searches for events where the observed bandwidth—after the latency and congestion delays are added—exceeds the bandwidth of the switch, and adds delays for these events.

4 PRELIMINARY RESULTS

We implement and evaluate a proof-of-concept CXLMemSim on a computer with an Intel i9-12900K@5.0GHz processor with 96 GB of DDR5 4800MHz memory. Our platform has a 30 MB LLC and a memory latency of 88.9 ns. We simulate the memory topology in Figure 2.

Benchmarks. We evaluate CXLMemSim with both microbenchmarks and real-world applications. Specifically, we implement five microbenchmarks that allocate memory with different system calls (i.e., mmap_read, mmap_write, sbrk, malloc, and calloc) and perform sequential writes to the allocated memory. The working set of calloc is 10 GB; the working set of the rest is 100 MB. Additionally, we use mcf and wrf from spec2017 to evaluate CXLMemSim on real-world applications. We compare CXLMemSim's overhead to a Gem5 implementation of CXL.mem [3] using syscall emulation.

Benchmark	Native (s)	Gem5 (s)	CXLMemSim (s)
mmap_read	0.194	523.146	7.7967
mmap_write	0.118	426.361	6.6755
sbrk	0.174	381.597	6.0312
malloc	0.691	2359.973	97.7930
calloc	2.406	15.059	181.6472
spec2017 mcf	215.311	31537.609	1215.4854
spec2017 wrf	5.418	failed	17.3756

Table 1: Performance evaluation of CXLMemSim

Performance Overhead.

Table 1 shows our results. CXLMemSim slows execution by an average factor of 41.06. The system's slowdown is lower than Gem5 on all benchmarks except calloc; on average it is faster than Gem5 by a factor of 73. Thus, CXLMemSim's slowdown is significantly lower than architectural simulators.

5 RELATED WORK

We describe the two lines of memory simulator work from which CXLMemSim takes inspiration:

CXL simulation. Two prior works simulate CXL-attached memory: DirectCXL [4] and POND [10]. DirectCXL supports CXL 2.0 using FPGAs as external memory controllers. DirectCXL require investment in a memory hierarchy before profiling, whereas CXLMemSim allows data-center operators to evaluate potential topologies before procurement. POND uses NUMA as a mechanism to simulate multiple tiers of memory, which limits the hierarchies that POND can support and prevents POND for emulating CXL.Mem bandwidth.. In contrast, CXLMemSim can emulate arbitrary user-provided memory topologies.

Persistent Memory Simulation. CXLMemSim's design is similar to that of software-based persistent memory simulators including Quartz [12], MES [7], and LEEP [15]. These tools execute programs on existing software and simulate persistent memory read/write latency by tracing memory accesses and inserting periodic delays. CXLMemSim's design is different in that (1) CXLMemSim supports CXL-memory pool hierarchies, whereas prior tools only support single-level hierarchies of far memory, (2) CXLMemSim simulates

read/write bandwidth tracks congestion in the CXL switch in addition to latency.

REFERENCES

[1] D. Bakhvalov. Advanced profiling topics. pebs and lbr, 2018. URL https://easyperf.net/blog/2018/06/08/Advanced-profiling-topics-PEBS-and-LBR#processor-event-based-sampling-pebs.

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[3] fadedzipper. Gem5 cxl version, 2022. URL https://github.com/fadedzipper/gem5-cxl/compare/stable...cxl.mem-dev.

[4] 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)*, pages 287–294, 2022.

[5] B. Gregg. *BPF Performance Tools*. Addison-Wesley Professional, 2019.

[6] A. Kleen. Pmu tools, 2022. URL https://github.com/andikleen/pmu-tools.

[7] A. Koshiba, T. Hirofuchi, R. Takano, and M. Namiki. A software-based nvm emulator supporting read/write asymmetric latencies. *IEICE TRANSACTIONS on Information and Systems*, 102(12):2377–2388, 2019.

[8] K. Kourtis, A. Trivedi, and N. Ioannou. Safe and efficient remote application code execution on disaggregated nvm storage with ebpf. *arXiv preprint arXiv:2002.11528*, 2020.

[9] 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. 2023.

[10] H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, et al. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst*, 2023.

[11] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter. Hemem: Scalable tiered memory management for big data applications and real nvm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 392–407, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450387095. doi: 10.1145/3477132.3483550. URL https://doi.org/10.1145/3477132.3483550.

[12] H. Volos, G. Magalhaes, L. Cherkasova, and J. Li. Quartz: A lightweight performance emulator for persistent memory software. In *Proceedings of the 16th Annual Middleware Conference*, pages 37–49, 2015.

[13] V. M. Weaver. Advanced hardware profiling and sampling (pebs, ibs, etc.): creating a new papi sampling interface. Technical report, Technical Report UMAINE-VMWTR-PEBS-IBS-SAMPLING-2016-08. University of Maine, 2016.

[14] Y. Zhong, H. Wang, Y. J. Wu, A. Cidon, R. Stutsman, A. Tai, and J. Yang. Bpf for storage: an exokernel-inspired approach. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 128–135, 2021.

[15] G. Zhu, K. Lu, X. Wang, X. Zhou, and Z. Shi. Building emulation framework for non-volatile memory. *IEEE Access*, 5:21574–21584, 2017.