Performance Study of CXL Memory Topology

Jianbo Wu[†], Jie Liu[†], Gokcen Kestor[*], Roberto Gioiosa[*],
Dong Li[†], and Andres Marquez[*]
{jwu323,jliu279,dli35}@ucmerced.edu,{gokcen.kestor,roberto.gioiosa,andres.marquez}@pnnl.gov
[*]Pacific Northwest National Laboratory, Richland, Washington, USA
[†]University of California, Merced, Merced, California, USA

ABSTRACT

This paper presents a comprehensive evaluation of the performance impact of various Compute Express Link (CXL) memory topologies, with a particular emphasis on CXL switches, in the context of High-Performance Computing (HPC) and Large Language Model (LLM) inference workloads. Our study unveils significant performance variations across different topologies, demonstrating that certain configurations yield superior performance for specific workloads. These findings underscore the critical importance of tailored topology selection in optimizing system performance. Additionally, we address the inherent challenges associated with integrating CXL switches, including overhead considerations and routing complexities. Our research highlights the necessity for thorough evaluation methodologies to fully leverage CXL technology's potential in contemporary computing environments. These insights provide valuable guidance for system architects and data center operators in designing and optimizing CXL-based infrastructures for diverse workload requirements.

CCS CONCEPTS

• **Computer systems organization → Heterogeneous (hybrid) systems**; **Interconnection architectures**; • **Computing methodologies → Simulation evaluation**.

KEYWORDS

CXL.mem and CXL switch, CXL.mem topologies, CXL.mem Emulation

1 INTRODUCTION

The exponential growth of data and the increasing complexity of applications, from traditional High-Performance Computing (HPC) [5, 22] to rising Large Language Model (LLM) [13, 21], in modern computing environments have driven the need for efficient and highly scalable memory architectures. Traditional memory architectures [15] often struggle to keep up with the demands for higher bandwidth and lower latency, leading to bottlenecks that hinder overall system performance.

Compute Express Link (CXL) [9, 14, 16] has emerged as a pivotal interconnect technology designed to address these challenges by providing high-bandwidth, low-latency connections between processors and memory devices. CXL enables a more flexible and dynamic allocation of memory resources, making it an attractive solution for heterogeneous computing environments where various types of processors, such as CPUs, GPUs, and FPGAs, need to share and access memory efficiently. The CXL standard [14] includes three main protocols: `CXL.io`, `CXL.cache`, and `CXL.memory` (also known as `CXL.mem`), dynamically multiplexed on PCIe physical layer, which together support coherent memory sharing and efficient data transfer between connected devices.

Despite the promise of CXL, there are significant challenges in optimizing its implementation. Traditional memory topologies can lead to suboptimal resource utilization e.g. memory stranding [8] and scalability issues. For instance, direct-attached memory configurations [9] may not fully leverage the capabilities of CXL, resulting in inefficient memory usage and higher latency. Moreover, existing CXL implementations [5, 16] often lack the ability to effectively balance the load across multiple memory devices, further limiting performance improvements.

CXL switches [4] have been proposed as a solution to address these limitations. CXL switches facilitate the connection of multiple memory devices to multiple hosts, thereby enhancing overall memory bandwidth and reducing latency. By enabling a more distributed and balanced memory architecture, CXL switches can potentially overcome the scalability and resource utilization issues inherent in traditional memory configurations [9, 16].

Challenges. However, the implementation of CXL switches introduces its own set of challenges. Designing and integrating CXL switches into existing systems requires careful consideration of compatibility and performance optimization. The overhead [4] associated with switch management and the complexity of routing memory requests through the switch can also impact overall system performance. Furthermore, the effectiveness of CXL switches in different computing scenarios needs to be thoroughly evaluated to ensure they provide the expected benefits [9].

These challenges underscore the need for a comprehensive evaluation of CXL memory topologies, particularly CXL switches, to understand their performance implications fully. Addressing these challenges is critical to unlocking the full potential of CXL technology in modern computing environments.

Solutions. In this paper, we conduct a detailed performance study of various CXL memory topologies, with a specific focus on CXL switches. Our study aims to evaluate the performance benefits and trade-offs associated with using CXL switches in different computing scenarios. By analyzing key performance metrics such as runtime, we provide insights into the effectiveness of CXL switches in improving system performance.

We summarize the major contributions as follows.

- We reveal performance of different CXL memory topologies in HPC and LLM inference, guiding the selection of memory topology.
- We verify the simulation results with real CXL devices.
- We undertake a comprehensive cost-benefit analysis and develop a latency-based cost model to evaluate how CXL

memory could substantially reduce real-world applications' TCO (Total Cost of Ownership).

2 BACKGROUND

Era of Data-intensive Applications. HPC and LLM represent two critical perspectives that necessitate advancements in memory and switch technologies like CXL. In HPC, the exponential growth of data and computational demands requires ultra-fast, low-latency memory solutions to efficiently process and analyze vast datasets, thereby enhancing overall system performance and scalability [5, 20]. On the other hand, LLMs like LLaMA [10, 18, 19] and Mistral [7], with their billion-level model weights and extensive key-value caches (to store context information within sentences), demand high-bandwidth memory access to manage and retrieve massive amounts of data swiftly, ensuring smooth training and inference processes [3]. The integration of CXL memory and switches offers promising solutions to these challenges, providing a unified, high-speed interconnect that significantly improves data throughput and reduces latency across diverse computational workloads.

`CXL.mem` protocol. CXL is an emerging interconnect standard designed to enhance communication between CPUs, accelerators, and memory, providing high-speed, low-latency connections to improve overall system performance. The `CXL.mem` protocol, a crucial component of this standard, facilitates memory expansion and sharing, allowing for more flexible and efficient use of memory resources. This is particularly beneficial in data-intensive applications, where the demand for memory bandwidth and capacity is constantly increasing. By enabling direct memory access by accelerators and other devices, `CXL.mem` reduces bottlenecks and enhances the system's ability to handle large datasets.

CXL switches further augment the capabilities of the CXL ecosystem. CXL switches [4] allow multiple CXL devices to connect to a single processor, significantly increasing the scalability and flexibility of the system architecture. With CXL switches, it is possible to dynamically allocate resources, optimize workload distribution, and improve resource utilization across diverse computing environments. This flexibility is essential for modern data centers and high-performance computing systems, where efficient resource management directly translates to improved performance and cost-effectiveness. Together, `CXL.mem` and CXL switches represent a transformative advancement in interconnect technology, paving the way for next-generation computing infrastructures.

`CXL.mem` Emulator. The recent advancements in memory disaggregation have introduced the `CXL.mem` standard, which allows for efficient shared memory pools across servers using a load/store interface. As commercial `CXL.mem` devices are yet to be released, Yang et al. [23] have addressed the research gap with the development of *CXLMemSim*, a lightweight simulation tool that emulates `CXL.mem`'s performance characteristics. *CXLMemSim* simulates the perspective of the CPU by incorporating the target latency, taking into account the Reorder Buffer (ROB) and various cache line states. This simulation assesses the impact of these factors on application-level performance. *CXLMemSim* enables the profiling of memory access patterns and the insertion of realistic latency and bandwidth delays within unmodified applications. This tool facilitates the evaluation of diverse `CXL.mem` topologies and memory management techniques, providing valuable insights into the performance implications of this emerging technology. With a modest performance overhead, *CXLMemSim* offers a critical resource for researchers and system architects to optimize memory-centric applications in anticipation of the widespread adoption of `CXL.mem` in data centers.

3 ANALYSIS

3.1 Setup

Our hardware details are shown in Table 1. We use *CXLMemSim* [23] emulator (allocate local memory firstly, then interleavely allocate the remote memory within the topology according to the inverse proportion of each remote memory device's latency while memory access sequentially) with NPB-OMP [11] benchmark (detailed in Table 2) to evaluate performance of different CXL topologies in HPC and IPEX-LLM [1] (Intel LLM library, which accelerates local LLM inference on Intel CPU) to evaluate performance of different CXL topologies in LLM inference using Mistral-7B [7] with 256 max tokens to generate for 5 times (stable execution time of each iteration). Meanwhile, we increase number of OpenMP threads (x axis) and use numactl [2] to bind application process to a specific NUMA node in order to observe changes in performance.

We have designed four different topologies: (1) CXL direct-attached, (2) CXL Switch-attached, (3) CXL Switch-attached with mixed paths, and (4) CXL Switch-attached with one switch connecting multiple CXLs. Figure 1 reveals our topologies. We mainly change the value of the *topology* argument as suggested in [23] to simulate behaviors of different topologies using Newick tree syntax (values of other arguments are default used in the simulator).

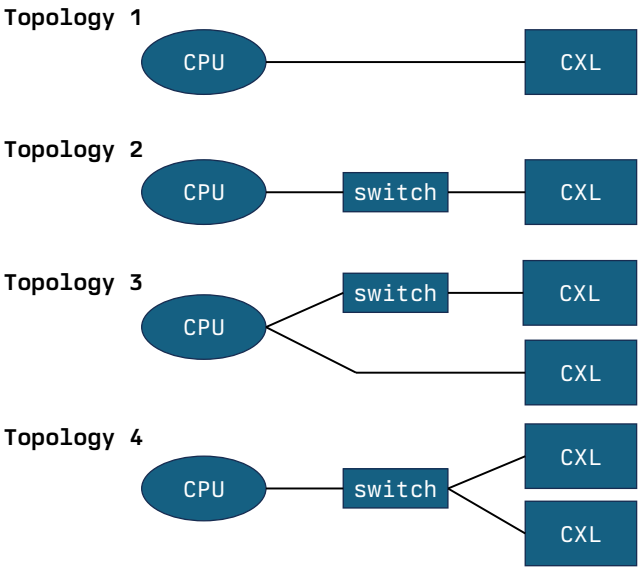

Figure 1: Different CXL Memory Topologies.

3.2 Simulation Results

3.2.1 CXL Switch Congestion and its Impact on HPC Performance. Figure 2 illustrates the performance of BT and LU workloads across

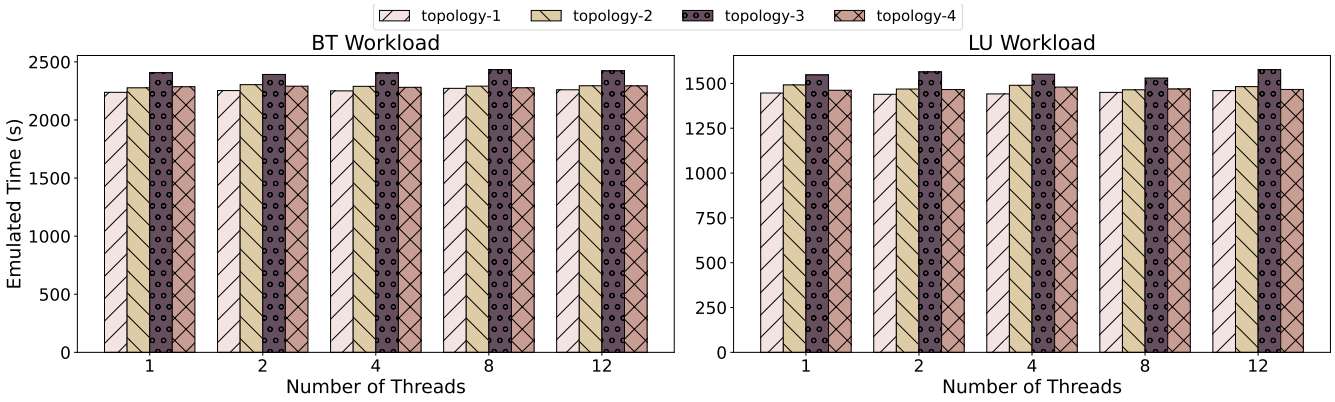

Figure 2: HPC Results of Different CXL Memory Topologies.

Table 1: Hardware details of our evaluation platform.

Component	Specification
OS	Ubuntu 22.04 with kernel 5.15.0-101-generic
CPU	Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz 24 cores in total
Memory	two NUMA nodes, each with ~90 GB memory

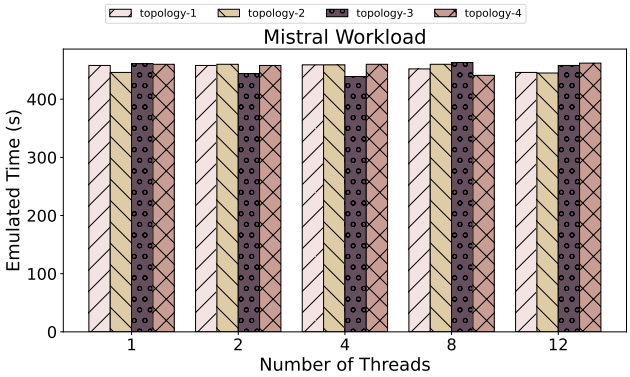

Figure 3: LLM Results of Different CXL Memory Topologies.

various thread counts and CXL switch topologies, revealing notable congestion effects in high-performance computing (HPC) scenarios. For the BT workload, topology-3 shows the most pronounced congestion effects, with execution time increasing from 2406.86 seconds at 1 thread to 2422.27 seconds at 12 threads, peaking at 2433.29 seconds with 8 threads. This suggests that the benefits of increased parallelism are outweighed by congestion in this topology. Similarly, the LU workload on topology-3 demonstrates significant impact, with execution time fluctuating but generally increasing from 1547.13 seconds at 1 thread to 1576.79 seconds at 12 threads. These patterns across topologies demonstrate that CXL switch congestion impacts HPC performance in complex ways as parallelism increases.

The non-linear nature of performance changes across different topologies and workloads underscores the challenges in optimizing CXL-connected systems for HPC applications. While some configurations show resilience to increased parallelism, others clearly demonstrate the limitations imposed by CXL switch congestion. This data highlights the importance of careful analysis when deploying HPC workloads in such systems, as the balance between parallelism benefits and CXL switch bottlenecks can significantly impact efficiency, especially in multi-threaded, large-scale computations. The research emphasizes the need for topology-aware scheduling and resource allocation strategies to maximize performance for HPC workloads in CXL-connected infrastructures.

3.2.2 CXL Switch Congestion and its Impact on LLM Performance. Figure 3 illustrates the performance of Mistral workloads across various thread counts and CXL switch topologies, revealing notable congestion effects. The impact of CXL switch congestion becomes evident as thread count increases, though in a non-linear fashion. In topology-3, we observe an intriguing performance pattern: improving from 461 seconds at 1 thread to 439 seconds at 4 threads, but then degrading to 458 seconds at 12 threads. This suggests initial parallelism benefits followed by congestion effects at higher thread counts. Similarly, topology-2 shows immediate congestion effects, with performance worsening from 446 seconds at 1 thread to 460 seconds at 2 threads, and remaining elevated at 459-460 seconds for higher thread counts.

These patterns across topologies demonstrate that CXL switch congestion impacts LLM performance in complex ways as parallelism increases. The non-linear nature of these performance changes suggests a nuanced interplay between thread count, CXL topology, and workload characteristics. While the Mistral LLM shows some resilience and even benefits from initial parallelism, the eventual performance degradation underscores the critical need for optimizing CXL-connected environments. This data highlights the importance of careful analysis when deploying LLMs in such systems, as the balance between parallelism benefits and CXL switch bottlenecks can significantly impact efficiency, especially in multi-threaded, large-scale deployments.

Table 2: NPB-OMP workloads for evaluation. "BW" stands for "bandwidth".

	Type	Workload Characterization	Input Problem	Mem. footprint
BT	Dense linear algebra	Unit-strided memory accesses from dense matrices	Class D	49 GB
LU	Sparse linear algebra	Indexed loads and stores from compressed matrices	Class D	53 GB

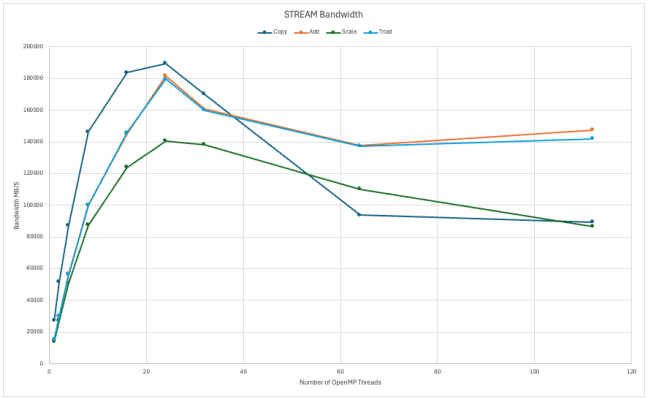

Figure 4: CXL Results of Stream Benchmark (x-axis represents number of OpenMP threads, y-axis represents Bandwidth).

3.3 Machine-Ready Results

To verify real performance difference between different CXL memory topologies, we have conducted some experiments on real CXL devices with topology-4 using Stream benchmark [12].

In Figure 4, the x-axis represents the number of OpenMP threads used in the STREAM benchmark, ranging from 1 to 128 threads, while the y-axis represents the bandwidth. As the number of threads increases, the memory bandwidth initially shows a nearly linear improvement for all operations (Copy, Add, Scale and Triad). However, after reaching approximately 32 threads, we also observe the *congestion* phenomenon where bandwidth plateaus for Add and Triad operations and even declines for Copy and Scale operations. This suggests that beyond a certain thread count, there are diminishing returns or even negative effects on memory bandwidth performance when using CXL switch.

3.4 Latency-Based Cost Model

Inspired by [17], we adopt a similar way to estimate TCO saving and use a capacity-bound application (LLM inference [1]) as an example to demonstrate how we develop our latency-based cost model for CXL switch scenario, but our methodology can be extended to other types of workloads as well. For LLM inference [1], the additional capacity enabled by CXL memory reduces the amount of data spilled to SSD and results in higher performance (less runtime). This means fewer servers will be needed to meet the same performance target.

Given that the workload maintains a relatively consistent memory footprint (the size of the active dataset) during execution, we can approximate the execution time of the workload by dividing it into three distinct segments:

(1) The segment processed using data stored in Main Memory (MMEM).
(2) The segment processed using data stored in CXL memory.
(3) The segment processed using data that has been offloaded to SSD storage.

We first make these measurements from microbenchmarks on a single server:

- **Baseline performance (L_s)**: Measure the latency when (almost) all working set is spilled to SSD. The absolute number is not used in our cost model. Instead, we then normalize it to 1 in our cost model.
- **Relative performance when the entire working set is in MMEM (RL_d)**: Using the same workload, we measure the latency when the entire working set is in MMEM and normalize it to L_s to get the relative performance (i.e., how much faster compared to the baseline).
- **Relative performance when the entire working set is in CXL memory (RL_c)**: Using the same workload, we measure the latency when the entire working set is in CXL memory, and normalize it to L_s to get the relative performance.

We then formulate our cost model using the parameters outlined in Table 3. For a working set size of W, the execution time of the baseline cluster could be approximated as the sum of two segments:

(1) The segment that is executed with data in MMEM,
(2) The segment that is executed with data spilled onto SSD.

$$T_{\text{baseline}} = N_{\text{baseline}} D \cdot RL_d + (W - N_{\text{baseline}} D)$$

The execution time of the cluster with CXL memory could be approximated in a similar way. It includes the segment that is executed with data in main memory, in CXL memory, and spilled to SSD respectively. We also consider congestion brought by using CXL switch:

$$T_{cxl} = N_{cxl} D \cdot RL_d + \frac{N_{cxl} D \cdot C \cdot RL_c}{min(N_{congest}, N_{threads})} + \left(W - N_{cxl} D - \frac{N_{cxl} D}{C}\right)$$

To meet the same performance target, $T_{\text{baseline}} = T_{cxl}$:

$$N_{\text{baseline}} D \cdot RL_d - N_{\text{baseline}} D = N_{cxl} D \cdot RL_d + \frac{N_{cxl} D \cdot C \cdot RL_c}{min(N_{congest}, N_{threads})} - N_{cxl} D - \frac{N_{cxl} D}{C}$$

With some simple transformations, we get the ratio between N_{cxl} and N_{baseline}:

Table 3: Parameters of our Latency-Based Cost Model.

Parameter	Description	Example Value
L_s	Latency when (almost) entire working set is spilled to SSD on a server. Normalized to 1 in the cost model.	1
RL_d	Relative latency when the entire working set is in main memory on a server, normalized to P_s.	0.1
RL_c	Relative latency when the entire working set is in CXL memory on a server, Normalized to P_s.	0.125
D	The MMEM capacity allocated to each server. For completeness only, not used in cost model.	
C	The ratio of main memory to CXL capacity on a CXL server. E.g. 2 means the server has 2x MMEM capacity than CXL memory.	2
$N_{baseline}$	Number of servers in the baseline cluster.	
N_{cxl}	Number of servers in the cluster with CXL memory to deliver the same performance as the baseline.	
R_t	Relative TCO comparing a server equipped with CXL memory vs. baseline server. E.g. If a server with CXL memory costs 10% more than the baseline server, this parameter is 1.1.	1.1
$N_{threads}$	Number of CPU threads used.	12
$N_{congest}$	Number of CPU threads used to cause congestion.	8

$$\frac{N_{\text{cxl}}}{N_{\text{baseline}}} = \frac{RL_d - 1}{RL_d + \frac{C \cdot RL_C}{min(N_{congest}, N_{threads})} - 1 - \frac{1}{C}}$$

TCO saving can then be formulated as follows:

$$TCO_{\text{saving}} = 1 - \frac{TCO_{cxl}}{TCO_{\text{baseline}}} = 1 - \frac{N_{cxl} R_t}{N_{\text{baseline}}}$$

For example, suppose $RL_d = 0.1$, $RL_c = 0.125$, $C = 2$, $N_{congest} = 8$, $N_{threads} = 12$, we get

$$\frac{N_{cxl}}{N_{\text{baseline}}} = 65.75\%$$

from the cost model. This means that by using CXL memory, we may reduce the number of servers by 34.25%. And if we further assume $R_t = 1.1$ (a server with CXL memory costs 10% more than the baseline server), the TCO saving is estimated to be 27.67%.

Our cost model provides an easy and accessible way to estimate the benefit from using CXL switch, providing important guidance to the design of the next-generation infrastructure.

4 RELATED WORK

Liu et al. [9] have conducted a detailed performance analysis of CXL memory-expansion cards, revealing that CXL memory exhibits latency comparable to remote DRAM but with lower bandwidth, as evidenced by latency increases of 2.8×, 2.4×, and 2.2× on three evaluated systems, respectively, compared to local DRAM. Notably, their study indicates that CXL memory can serve as a unique NUMA node with latencies similar to remote DRAM under heavy system load, suggesting the potential for CXL switches to manage memory resources efficiently in complex memory hierarchies.

The evolution of CXL technology has brought forth new paradigms in memory sharing and interconnect solutions. In a recent study by Jain et al. [6], the authors address the challenges and potential of memory sharing within CXL-enabled systems. Their findings on the coherency mechanisms introduced by CXL 3.0, such as Back Invalidation, are particularly relevant to the design of CXL switches. This work provides a framework for understanding the interplay between hardware and software in managing shared memory resources, which is critical when considering the role of CXL switches in maintaining data consistency and system performance.

5 CONCLUSION

This paper conducted a comprehensive performance study of various Compute Express Link (CXL) memory topologies, particularly focusing on CXL switches. Our findings indicate significant performance variations across different topologies, highlighting the importance of selecting the right configuration based on specific workload requirements.

While CXL switches offer promising benefits in terms of improved bandwidth and reduced latency, their integration poses challenges such as compatibility issues and routing complexity. Nonetheless, the potential for enhanced scalability and dynamic resource allocation makes CXL a transformative technology for modern computing environments.

Overall, our study underscores the necessity of thorough evaluation and optimization of CXL memory topologies to maximize performance gains. Future research should continue exploring CXL standards and solutions to address the remaining challenges, ensuring that CXL technology can meet the increasing demands of contemporary high-performance computing and large language model workloads.

ACKNOWLEDGMENTS

This work is supported by the US DOE Office of Science project “Advanced Memory to Support Artificial Intelligence for Science" at PNNL. PNNL is operated by Battelle Memorial Institute under Contract DEAC06-76RL01830.

REFERENCES

[1] Intel Analytics. 2023. IPEX-LLM: Accelerate local LLM inference and finetuning on Intel CPU and GPU. https://github.com/intel-analytics/ipex-llm.

[2] Andi Kleen (SUSE Labs). [n. d.]. NUMA Support for Linux. https://github.com/numactl/numactl.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Tam Do and Sanketh Srinivas. 2023. *CXL Memory Expansion, Pooling, Sharing, FAM Enablement, and Switching*. Open Compute Project. https://www.youtube.com/watch?v=VCYSzBFCBnQ Presented by Tam Do (Microchip) and Sanketh Srinivas (Microchip), Open Compute Project.

[5] Yehonatan Fridman, Suprasad Mutalik Desai, Navneet Singh, Thomas Willhalm, and Gal Oren. 2023. CXL Memory as Persistent Memory for Disaggregated HPC: A Practical Approach. arXiv:2308.10714 [cs.DC]

[6] Sunita Jain, Nagaradhesh Yeleswarapu, Hasan Al Maruf, and Rita Gupta. 2024. Memory Sharing with CXL: Hardware and Software Design Approaches. arXiv:2404.03245 [cs.ET]

[7] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[8] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 2023. 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*. 574–587.

[9] Jie Liu, Xi Wang, Jianbo Wu, Shuangyan Yang, Jie Ren, Bhanu Shankar, and Dong Li. 2024. Exploring and Evaluating Real-world CXL: Use Cases and System Adoption. arXiv:2405.14209 [cs.PF]

[10] Meta Llama. 2024. Llama3 Model Card. https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md. Accessed: 2024-05-31.

[11] Júnior Löff, Dalvan Griebler, Gabriele Mencagli, Gabriell Araujo, Massimo Torquati, Marco Danelutto, and Luiz Gustavo Fernandes. 2021. The NAS parallel benchmarks for evaluating C++ parallel programming frameworks on shared-memory architectures. *Future Generation Computer Systems* 125 (2021), 743–757.

[12] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.

[13] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 551–564.

[14] Debendra Das Sharma, Robert Blankenship, and Daniel S. Berger. 2024. An Introduction to the Compute Express Link (CXL) Interconnect. arXiv:2306.11227 [cs.AR]

[15] Richard Solomon. 2023. CXL: How an Accelerator Link Caused a Memory Revolution. *Synopsys IP Technical Bulletin* (2023). https://www.synopsys.com/designware-ip/technical-bulletin/cxl-in-memory-solutions.html

[16] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. 2023. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 105–121.

[17] Yupeng Tang, Ping Zhou, Wenhui Zhang, Henry Hu, Qirui Yang, Hao Xiang, Tongping Liu, Jiaxin Shan, Ruoyun Huang, Cheng Zhao, et al. 2024. Exploring Performance and Cost Optimization with ASIC-Based CXL Memory. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 818–833.

[18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[19] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[20] Jacob Wahlgren, Maya Gokhale, and Ivy B Peng. 2022. Evaluating emerging CXL-enabled memory pooling for HPC systems. In *2022 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 11–20.

[21] Jianbo Wu, Jie Ren, Shuangyan Yang, Konstantinos Parasyris, Giorgis Georgakoudis, Ignacio Laguna, and Dong Li. [n. d.]. LM-Offload: Performance Model-Guided Generative Inference of Large Language Models with Parallelism Control. ([n. d.]).

[22] Shuangyan Yang, Minjia Zhang, Wenqian Dong, and Dong Li. 2023. Betty: Enabling large-scale gnn training with batch-level graph partitioning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 103–117.

[23] Yiwei Yang, Pooneh Safayenikoo, Jiacheng Ma, Tanvir Ahmed Khan, and Andrew Quinn. 2023. CXLMemSim: A pure software simulated CXL. mem for performance characterization. *arXiv preprint arXiv:2303.06153* (2023).

Received 24 June 2024; accepted 5 Aug 2024