Enhancing MPI Collective Communication Performance Using CXL Shared Memory and an Intelligent CXL Switch

ETRI
Supercomputing System Research Section

Hooyoung Ahn

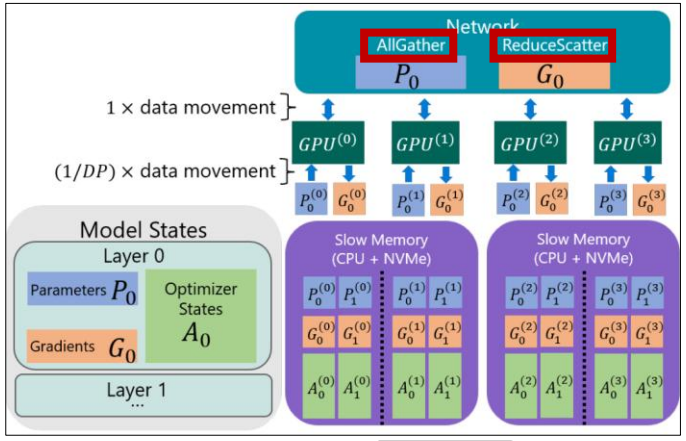
August 20, 2025

Contents

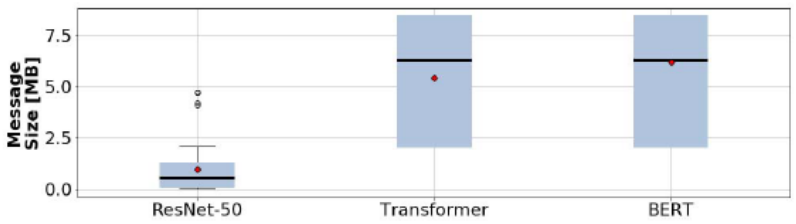
- Background
- Motivation
- Problem Definition
- Project Goals
- Roles of ETRI and OSU
- Our Approach
- Road Map
- Conclusion

Background

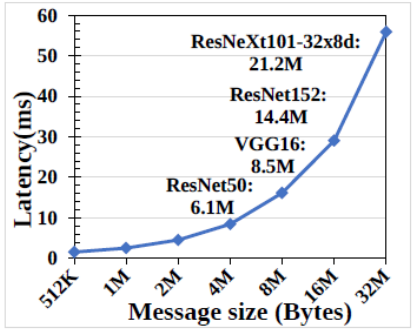
- Large-scale data-intensive applications in HPC and AI need distributed processing across multiple computing nodes
  - At this time, complex and frequent communication occurs among computing nodes
  - Providing enough memory for these applications is essential for performance improvement
- For example, LLM applications perform distributed training due to the large model and data sizes [1]
  - AllGather and ReduceScatter is used as the main collective communications
  - As the data and model size increases, the collective communication message size also increases [2]
  - However, collective communication suffers from increased latency when handling large messages [3]



A snapshot of ZeRO-Infinity training [1]



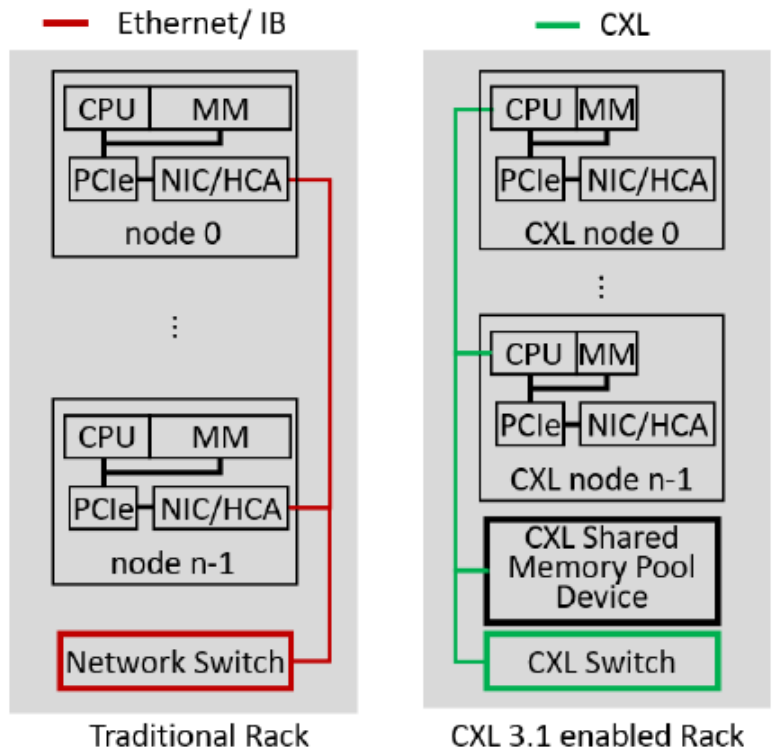
Message Size Distribution for various networks [2]



Message sizes of Allgather in PyTorch FSDP Training on 16 GPUs [3]

Motivation

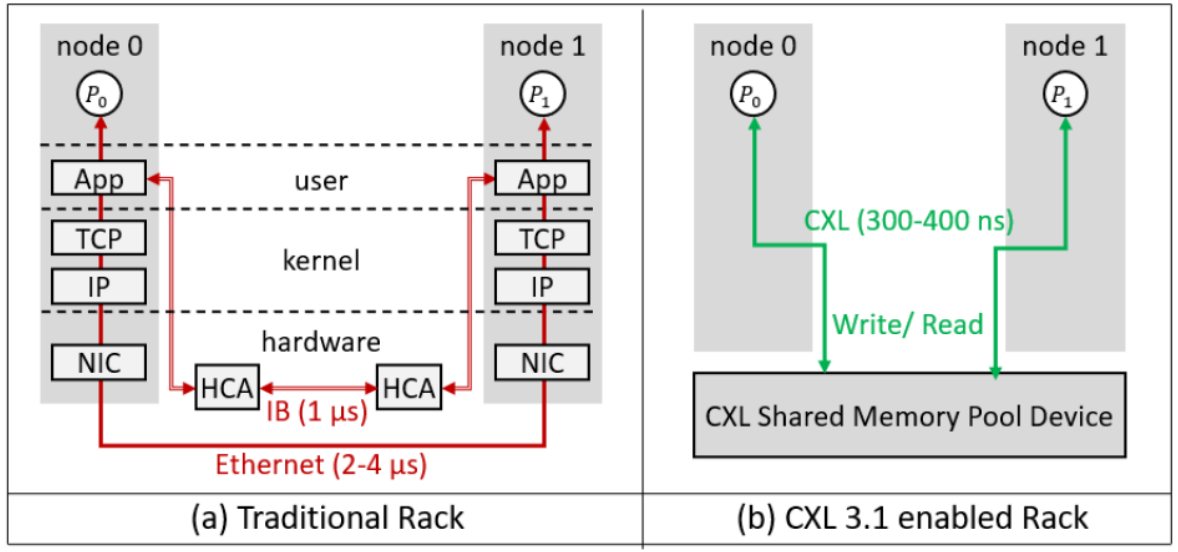
- In a Traditional rack
  - Computing nodes consists of CPUs, main memory, PCIe, Network Interface Cards (NICs), and Host Channel Adapters (HCAs)
  - They are connected to a **network switch** via **Ethernet or InfiniBand (IB)**
- In a CXL 3.1 enabled rack
  - CPUs that support CXL are directly connected to a CXL switch
  - Each node is equipped with significantly less main memory compared to nodes in a traditional rack
  - The limited memory capacity can be supplemented by the ***CXL shared memory pool device***, effectively addressing the problem of memory overprovisioning



Hardware Configuration of Traditional and CXL Racks

Motivation

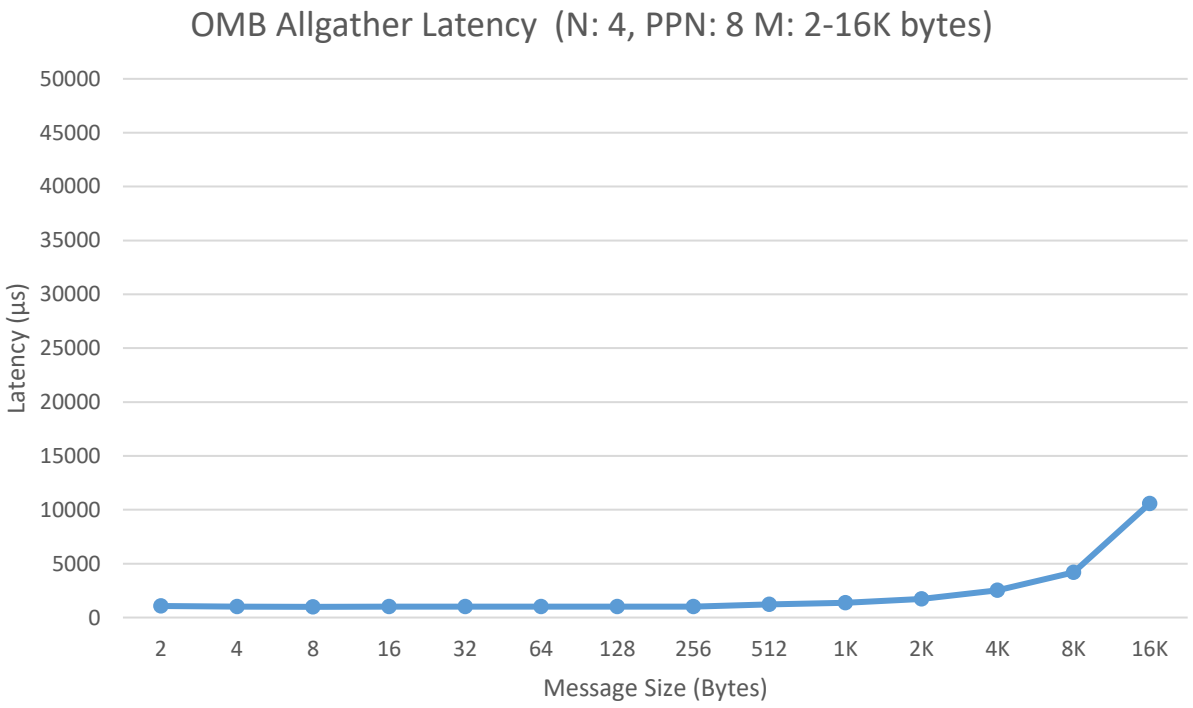
- In a traditional rack
  - When $P0$ communicates with $P1$ using **Ethernet**, the communication must traverse the user, kernel, and hardware layers, including the application layer, TCP/IP layer, and NIC, taking approximately **2-4 µs**
  - When using **IB** for communication, it involves the application layer and HCA, taking around **1µs**
- In a CXL 3.1 enabled rack
  - Inter-node communication is enabled through **read** and **write** operations on **the CXL shared memory pool device**, which maintains **cache coherence**
  - In this method, communication between $P0$ and $P1$ takes approximately **300-400 ns**

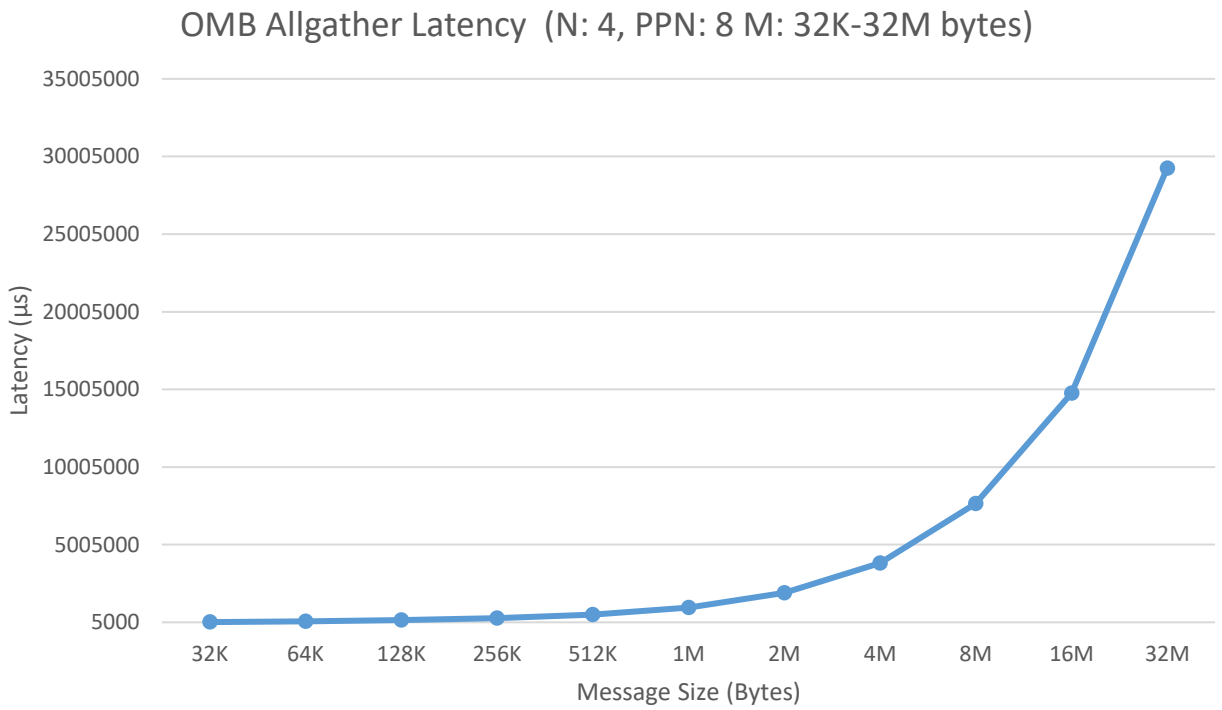


Comparison of Inter-Node Communication Latency

Problem Definition

- We observed that as the message size increases, the communication latency of traditional allgather also increases
- Specifically, the allgather latency increased significantly for large messages





Experimental Results on ETRI's QEMU-based 4 Computing Nodes

Project Goals

- The goal of this study is to **enhance the MPI Inter-Node collective communication performance** in a **multi-node environment connected by CXL**

- Two Specific Goals
  - **Goal 1**. Utilizing the **CXL shared memory pool** for **collective communication**
    → 1st phase: Sept. 2023 - Aug. 2024
  - **Goal 2**. Utilizing the **intelligent CXL switch** for **collective communication**
    → 2nd phase: Sept. 2024 - Aug. 2025

→ To achieve above goals, we proposed iMEX (intelligent Memory EXpander)

Project Goals

Main memory (512GB) | N Computing Node | : Data Movement
CXL memory pool (512GB) | MP Memory Pool | n : # of Processes

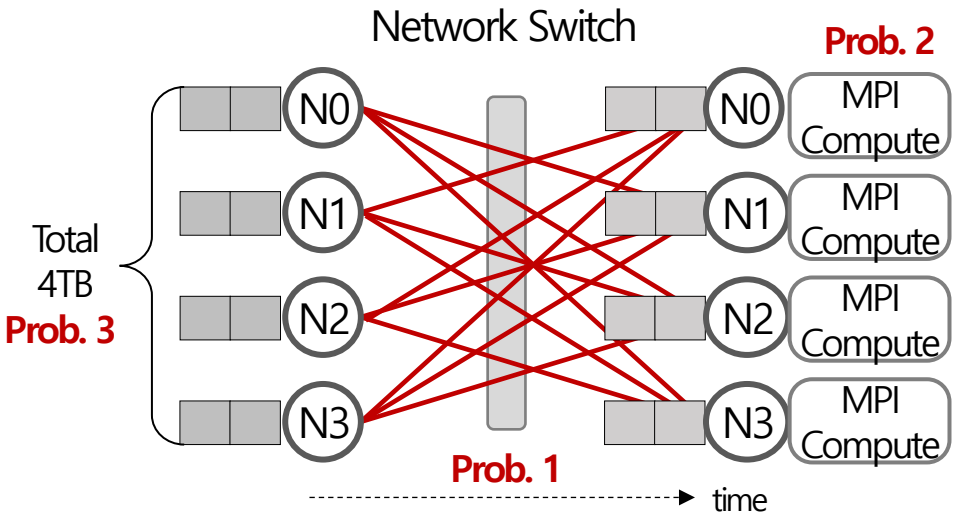
- Key Concept of iMEX

Data-Intensive Applications of **AI** and **HPC** fields

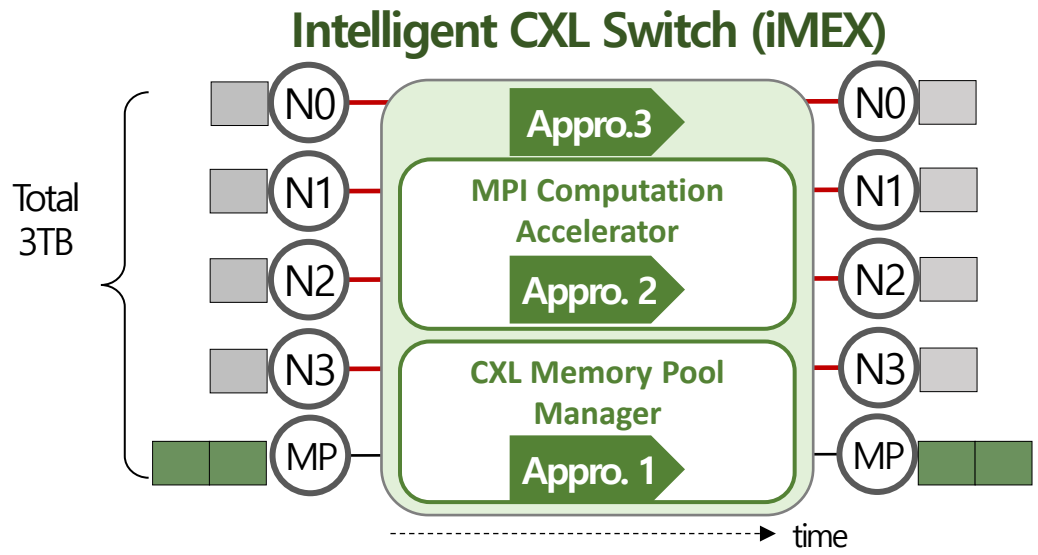
※ **Appro.3. MVAPICH2 optimized for IMEX**

All-Reduce

(AS-IS) Conventional Collective Communication



(TO-BE) Proposed Collective Communication



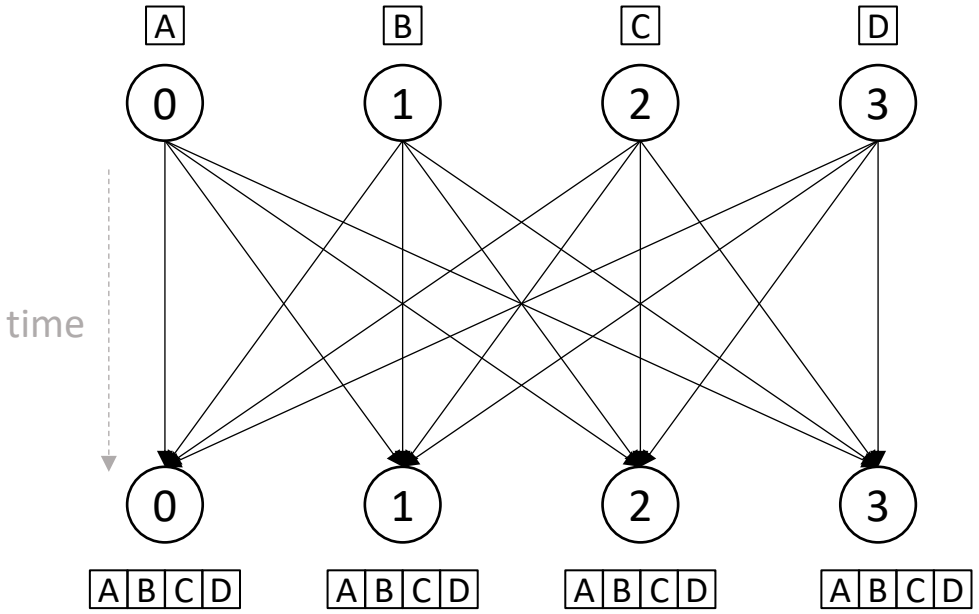
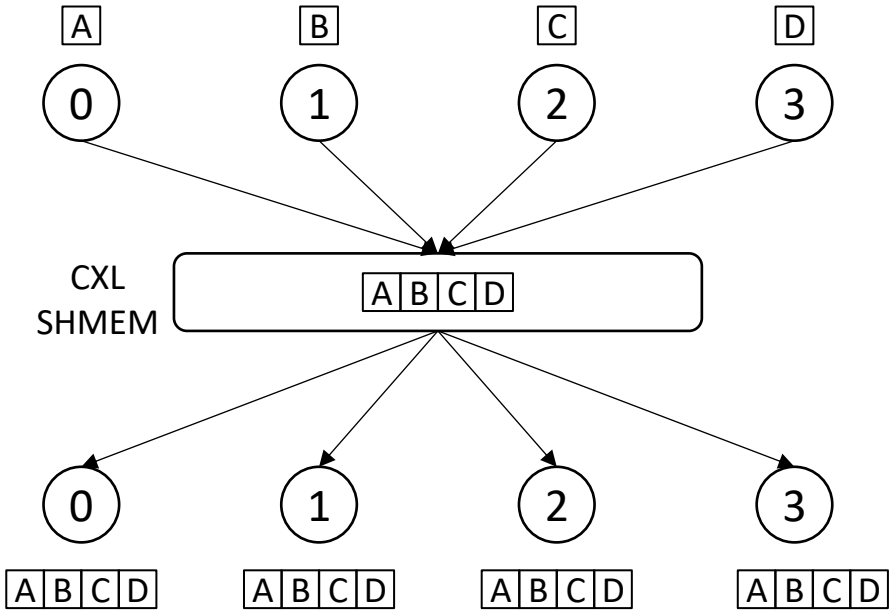
• # of Communication : n^2 • Communication **Latency** : 2-4 μs (100 GE)	Communication	• # of Communication : n • Communication **Latency** : 300-400 ns (CXL Switched DDR)
• MPI Computation on **CPU** • # of Computation : n	Computation	• MPI Computation on **Dedicated Accelerator** • # of Computation : **1**
• **Low**	Memory Utilization	• **High**

Roles of ETRI and OSU

	Research Area	Focus		Research Item
OSU	**Goal 1**	Beyond Rack-Scale CXL Memory Pool	1	Improving collective communication performance by utilizing the **beyond rack scale CXL memory pool device**
			2	Identify and **develop promising demonstration applications** to showcase the CXL-Based collective communication proposed in OSU's research item 1
ETRI	**Goal 1**	Single Rack-Scale CXL Memory Pool	1	**Proposed Approach 1**. CXL SHM-Based AllGather
	Goal 2	Intelligent CXL Switch	2	**Proposed Approach 2.** iMEX-Based Collective Communication (Intelligent CXL Switch-Based Collective Communication)

Proposed Approach for Goal 1

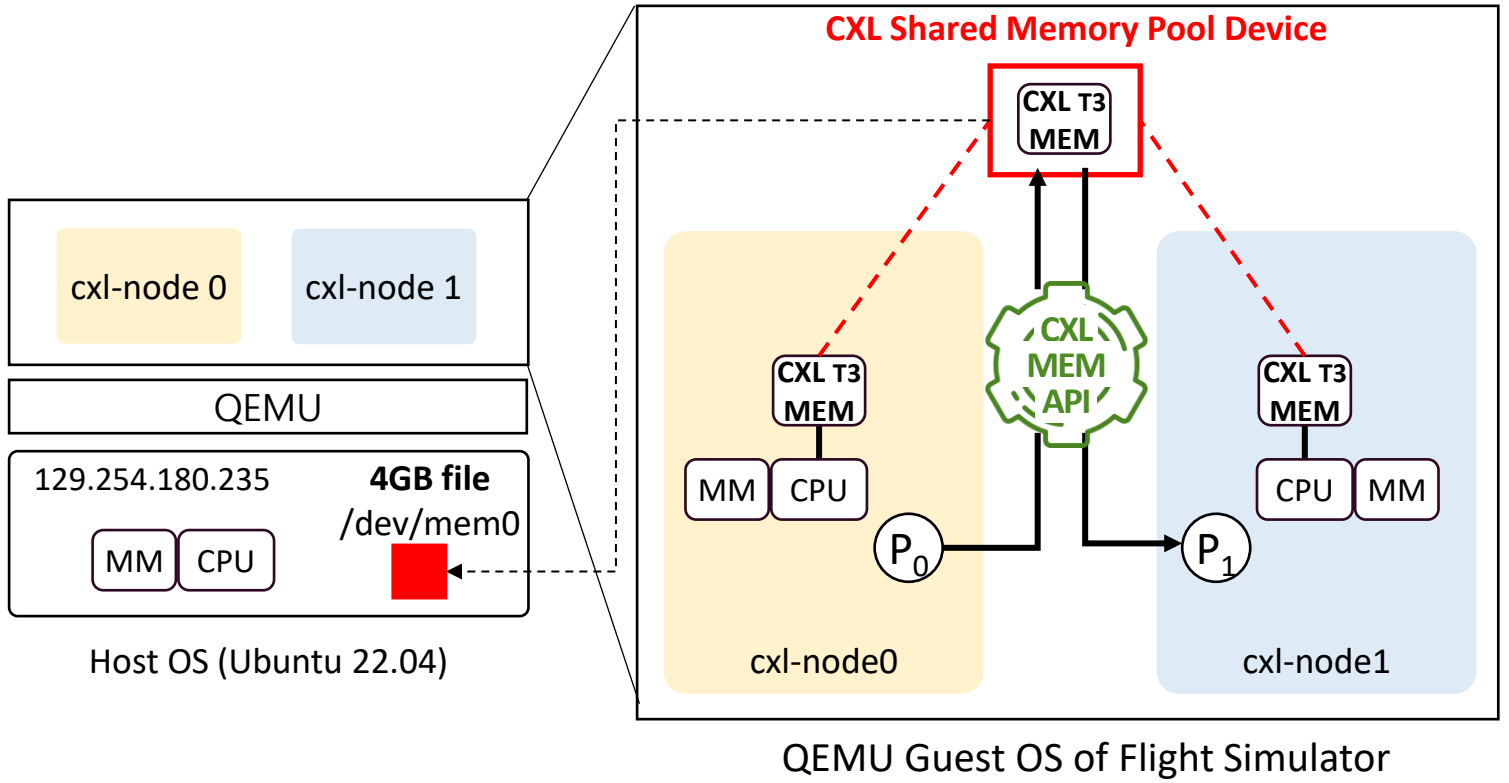
- CXL SHM-Based AllGather
  - Design and implement AllGather utilizing the CXL shared memory pool as the collective communication buffer
  - Measure Allgather latency with OMB for performance validation

Conventional. Network-Based AllGather	Proposed. CXL SHM-Based AllGather
	

Expect performance improvement by
- reducing the number of communications
- achieving performance gains with CXL over ethernet or IB

Implementation for CXL SHM-Based AllGather

- We developed five CXL memory APIs that are utilized for the CXL SHM-Based allgather
  - MPI ranks running on different computing nodes can utilize the CXL shared memory pool device as the communication buffer for collective communication

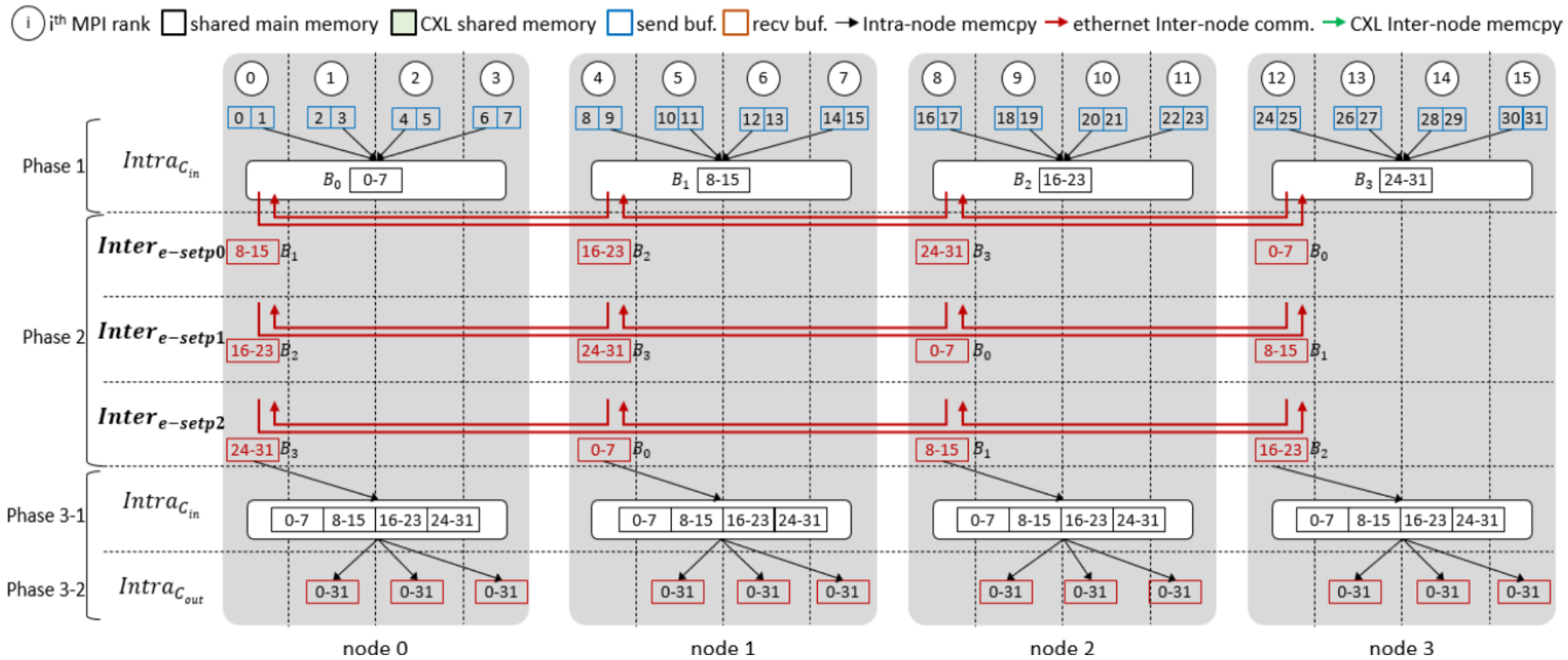


CXL Memory APIs

① cxlmemoryInitialize ()
② cxlmalloc (size)
③ cxlwrite (data, size, offset)
④ cxlread (data, size, offset)
⑤ cxlfree ()

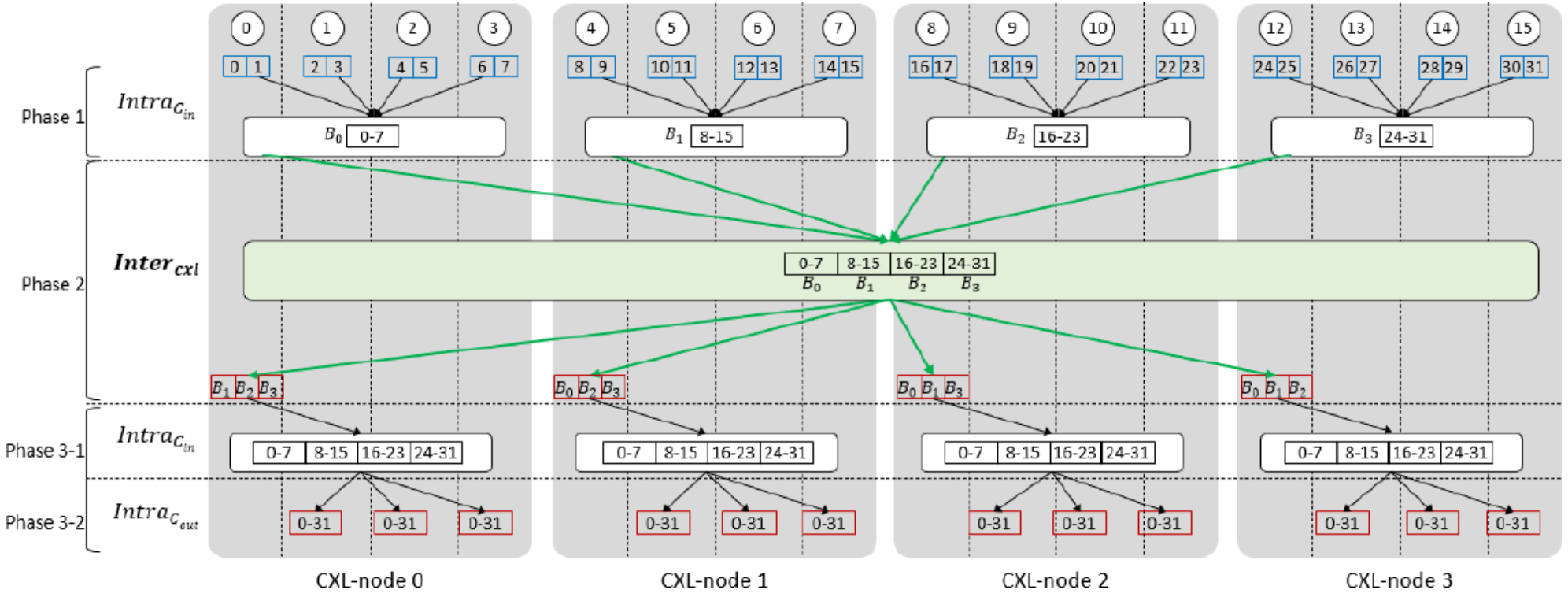
Process of the Traditional AllGather

- In Phase 1, ranks within each node copy data from the send buffer to shared main memory
- In Phase 2, node leaders transfer data to their left-hand neighbors using a ring algorithm over Ethernet
  - **This process must repeat $N - 1$ times for all rank**
- In Phase 3, the node leader copies data received from other nodes into the shared main memory. Then, each non-zero rank copies this data into its own receive buffer



Process of the CXL SHM–Based AllGather

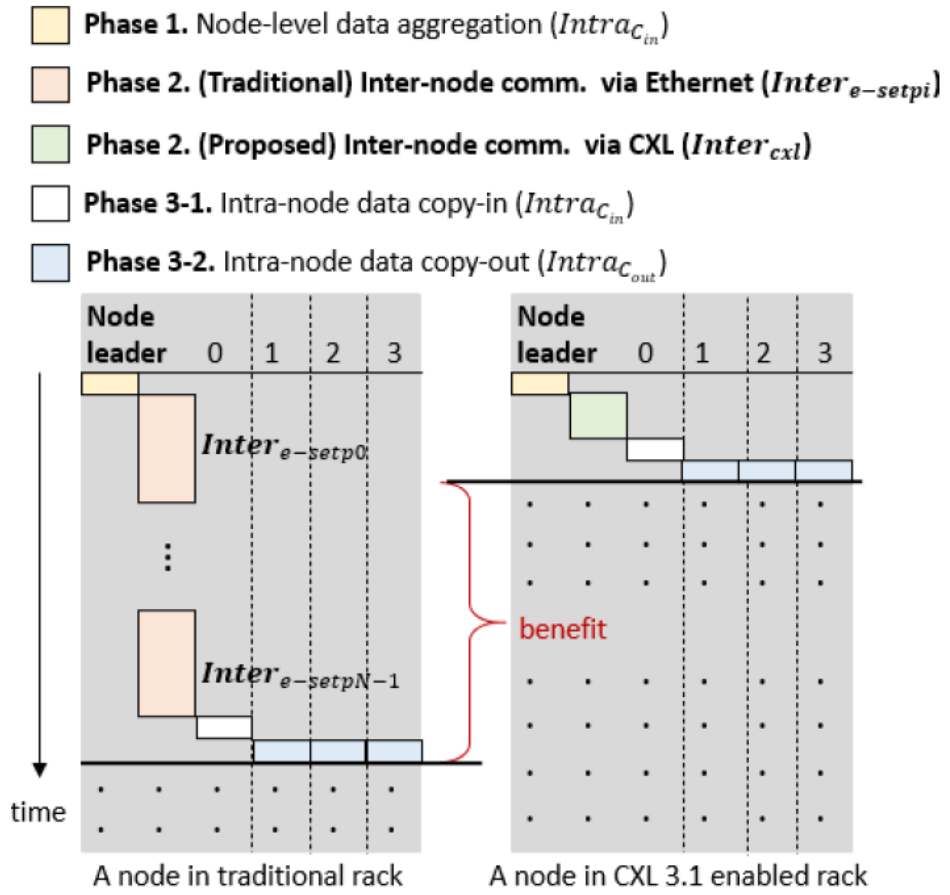
- Phases 1 and 3 are identical to those in traditional AllGather, while the **key difference** is in the inter-node communication method of **Phase 2**
- In Phase 2
  - Each node leader **writes** its send buffer data to the **CXL shared memory** in parallel
  - After all nodes' data is gathered in the CXL shared memory, each node leader **reads** the data from the **CXL shared memory**, excluding its own send buffer



Comparison of the Traditional and CXL SHM–Based AllGather

- In traditional Allgather
  - After Phase 1's node-level aggregation, **N – 1 inter-node communication via Ethernet** occur, followed by intra-node copy-in and copy-out
- In the proposed AllGather
  - Phases 1 and 3 are conducted as in traditional methods, but **Phase 2 involves only a single inter-node transfer via the CXL interconnect**
  - The frequency of this transfer is limited solely by the CXL bandwidth, independent of the number of nodes, achieving significantly lower latency compared to Ethernet-based communication

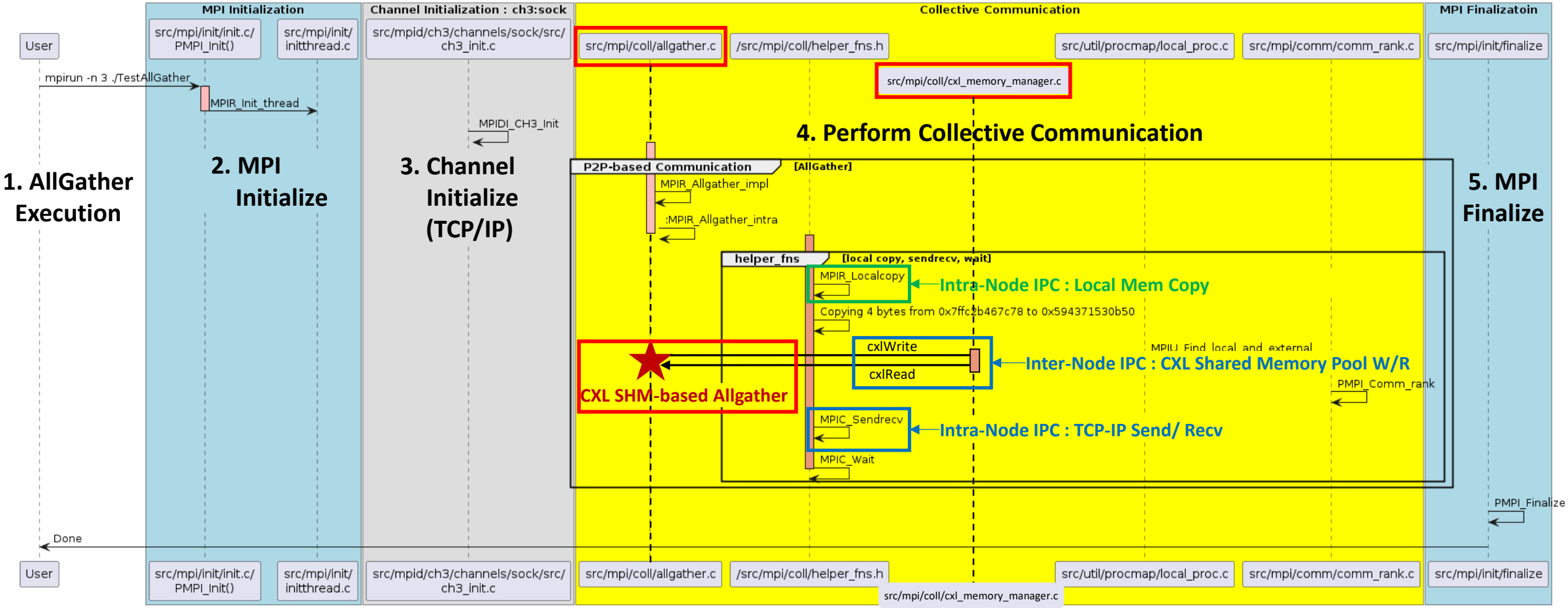
→ In summary, the proposed AllGather **reduces** both **the number of inter-node communication**s and **the latency**, as highlighted by the '**benefit**' shown in red.



Comparison of Communication Events within a Node in Traditional and Proposed Allgather

Implementation of the CXL SHM–Based AllGather

- We implemented the **CXL SHM-based allgather** and **cxl_memory_manager** in the MVAPICH2 2.3.7
- So, the CXL SHM-based allgather utilizes the **cxlwrite** and **cxlread** functions for inter-node communication



Experimental Setup for CXL SHM–Based AllGather

- Software emulator
  - Flight Simulator [5], which emulates the Multi-Node CXL 3.1 Shared Memory Pool Device in QEMU
- Experimental Environment
  - Host Machine
    - ✓ CPU : AMD EPYC 9754 128-Core Processor
    - ✓ Main memory : 792 GB
  - Guest Machine
    - ✓ QEMU cxl-2024-03-05 branch [6]
    - ✓ OS : fedora 38 (kernel version : vmlinuz-6.3.7-200.fc38.x86_64)
- Benchmark Suite
  - OSU Micro Benchmarks [7]

Experimental Results for CXL SHM-Based AllGather

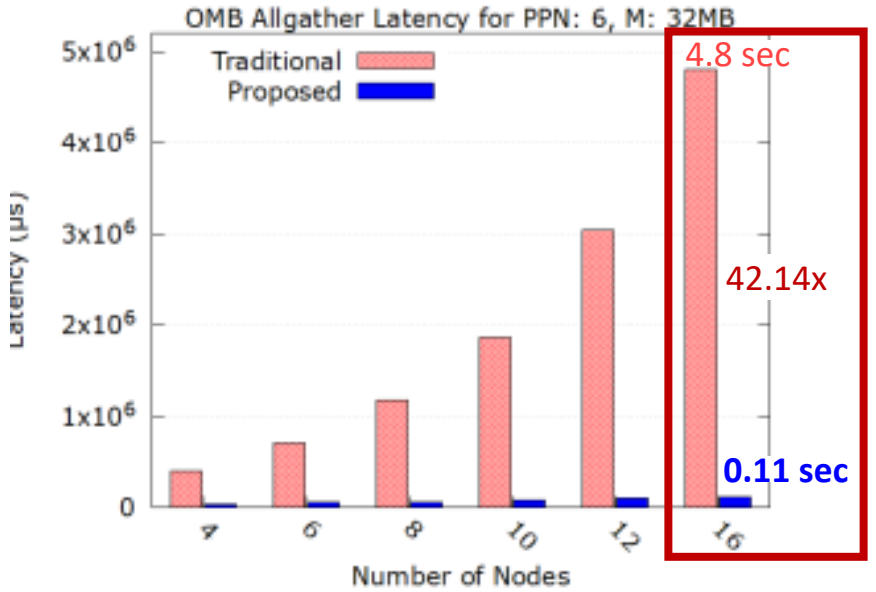
Published in IEEE BIgData'24

- The experimental items include the performance with
  - an increasing number of nodes, increasing number of processes per node (PPN), and increasing message size
- The proposed allgather significantly reduces communication latency compared to traditional allgather by up to 42.14x, with a minimum improvement of 2.91x

Performance Improvement of SHM-based Allgather over Traditional Allgather Across PPN Configurations (1 to 6 ppn)

No.	*N*	*M*	Performance Improvement			
			L = 1	*L* = 2	*L* = 4	*L* = 6
1	**8**	**512KB**	**2.91x**	6.84x	13.25x	20.15x
2	8	32MB	3.57x	6.68x	12.57x	19.2x
3	10	512KB	4.23x	7.66x	16.06x	24.12x
4	10	32MB	4.0x	8.29x	16.92x	22.93x
5	12	512KB	5.05x	9.01x	19.15x	28.71x
6	12	32MB	5.34x	10.28x	20.89x	30.29x
7	16	512KB	6.18x	12.55x	25.59x	38.22x
8	**16**	**32MB**	6.93x	13.69x	25.36x	**42.14x**

N : Number of Nodes *M* : Message Size *L* : Processes Per Node (PPN)



Performance Improvement of SHM-based Allgather Compared to Traditional Allgather Across the Number of Nodes (4 to 16)

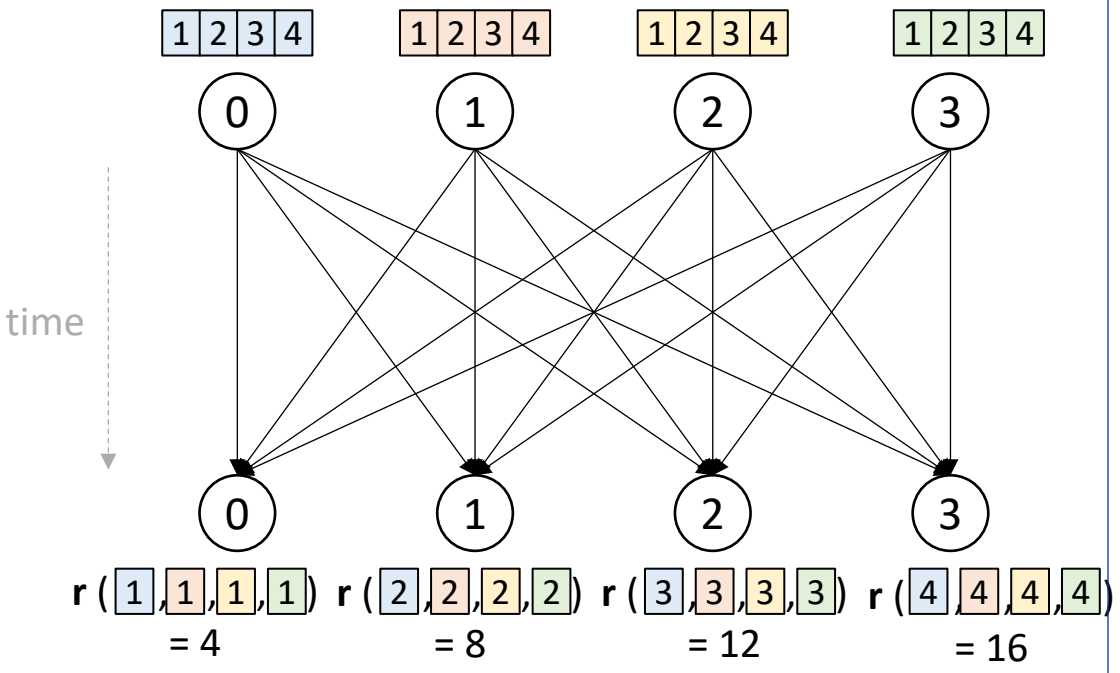
Proposed Approach for Goal 2

※ r : CPU reduction operation

※ sr : CXL switch reduction operation

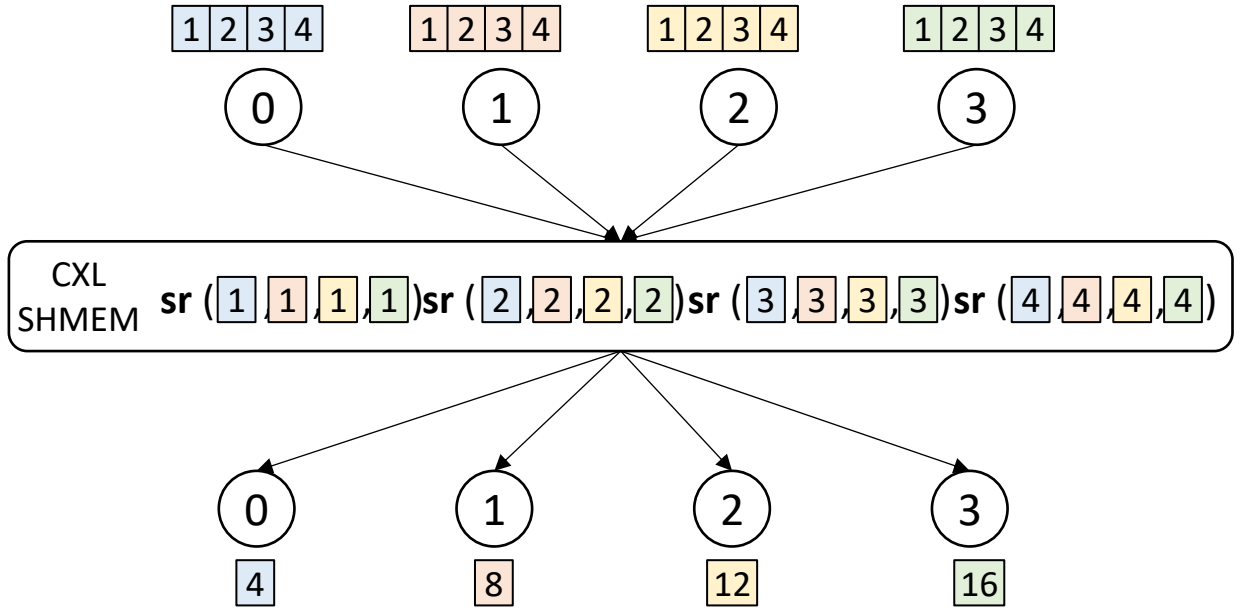
- iMEX-Based Collective Communication
  - Design and implement ReduceScatter and AllReduce utilizing iMEX for MPI communication and computation
  - Measure the latency of ReduceScatter and AllReduce using OMB for performance validation

Conventional. Network-Based Collective Communication



(e.g., ReduceScatter, sum)

Proposed. iMEX-Based Collective Communication



(e.g., ReduceScatter, sum)

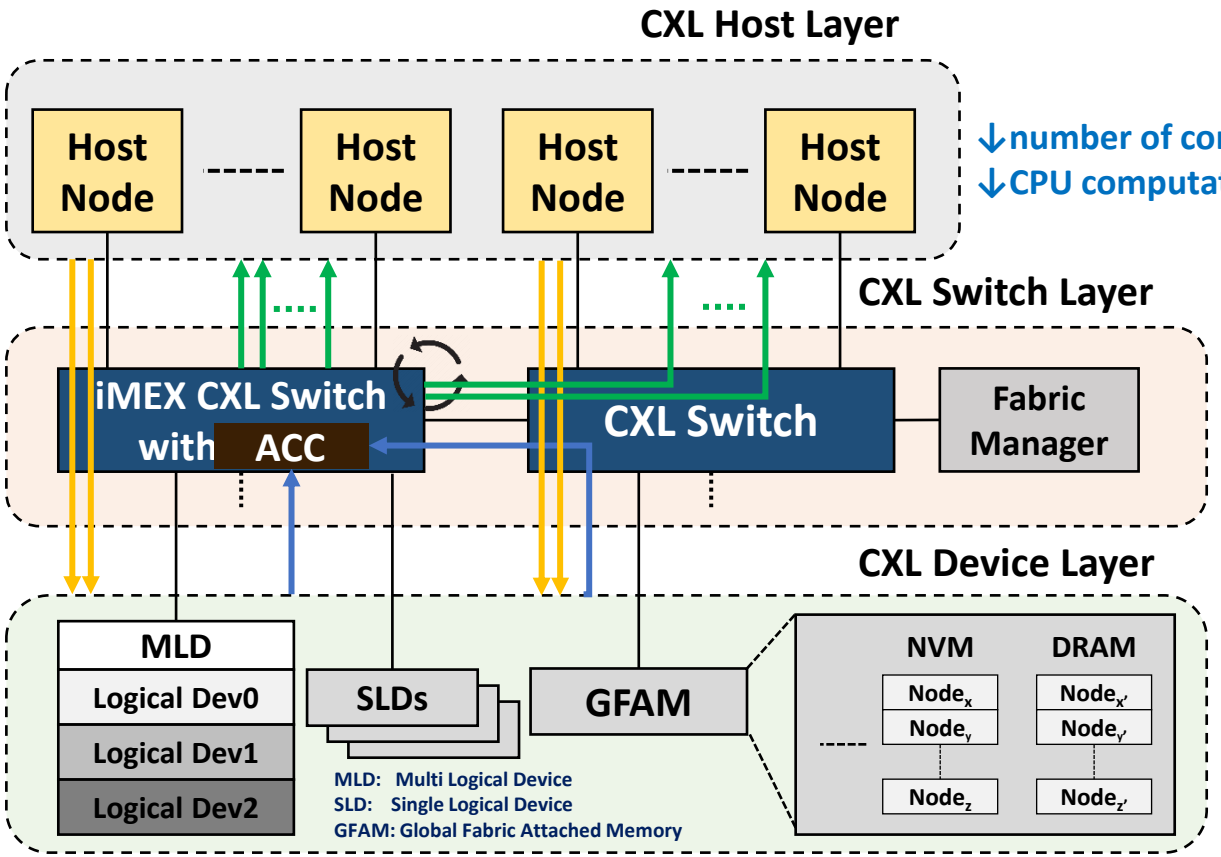
※ **sr**: intelligent CXL switch's reduction operation

Expect performance improvement by reducing
- the number of communications
- the CPU workload by offloading MPI computation to the switch's accelerator

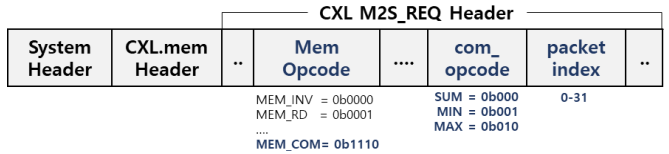
ETRI

Implementation for iMEX-Based Collective Communication

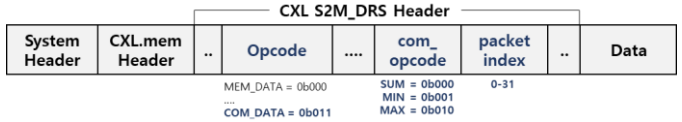
- We design and implement the iMEX CXL Switch, which consists of an MPI accelerator
- We define the Offloading and Result Packet Structures of the iMEX CXL switch
- We extend MVAPICH2 2.3.7 to leverage the iMEX CXL switch



iMEX-Based Collective Communication Architecture



Offloading Packet Structure



Result Packet Structure

Issuing offloading command

Gathering data into the accelerator

Collective operation on the accelerator (e.g., SUM/MIN/MAX)

Broadcasting collective operation result

ACC MPI communication and computation accelerator

Experiments for iMEX-Based Collective Communication

- Experimental Setup
  - Software emulator
    - ✓ OpenCIS (https://www.opencis.io/), which is an open-source SW simulator that models various CXL components in the qemu environment
  - Benchmark Suite
    - ✓ OSU Micro Benchmarks [7]
- Experimental Results
  - We plan to submit a paper, including the experimental results, to IPDPS 2026
  - We will compare two baselines and our proposed approach:
    - ✓ Baseline 1 is InfiniBand-based, Baseline 2 is CXL 3.1-based, and the proposed system is iMEX-based.

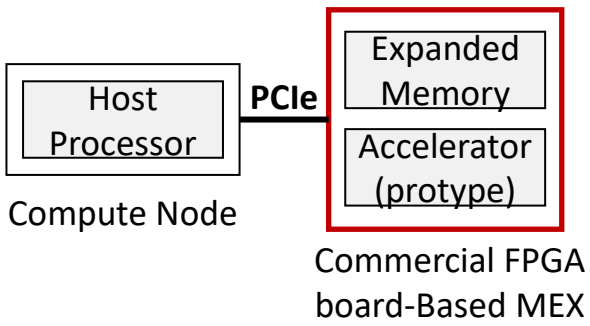
Road Map

- We aim to improve the performance of ***data-intensive applications*** in ***multi-node systems***

Now, we are here

Stage 1. MEX

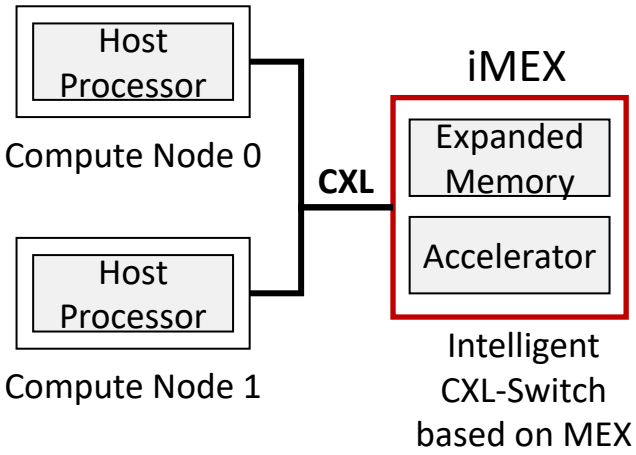
- Commercial FPGA board-Based MEX
- Up to 32GB expanded memory
- Prototype version of accelerator
- Support a **single node**



※ MEX (Memory EXpander)

Stage 2. iMEX

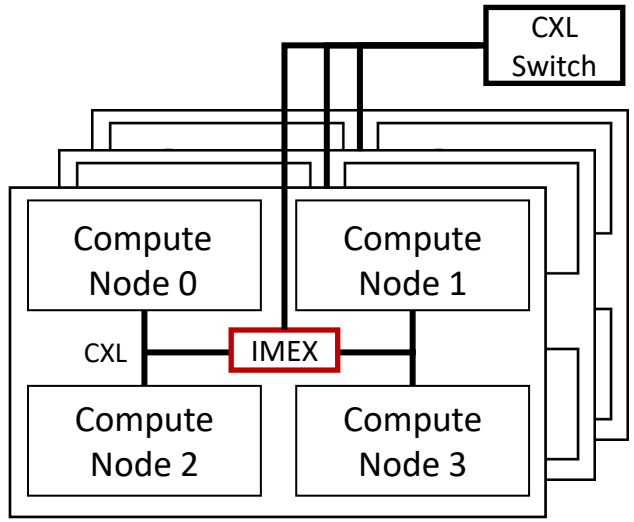
- Support **multi-node system** using CXL
- Accelerate MPI collective operation using **dedicated accelerator**
- Use **CXL** Memory Pool for **expanded memory** capacity



※ iMEX (intelligent MEX)

Stage 3

- Improvement the scalability of iMEX
- Multiple iMEX devices will be connected to a CXL spine Switch
- Support more complex topology



Conclusion

- **Improved memory utilization** for AI and HPC systems through the CXL Memory Pool as an MPI communication buffer
- **Enhanced collective communication performance** with iMEX's MPI Accelerator
- Reduced communication cost leading to **better AI and HPC application performance**

References

1. Rajbhandari, Samyam, et al. "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021.
2. KLENK, Benjamin, et al. An in-network architecture for accelerating shared-memory multiprocessor collectives. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2020. p. 996-1009.
3. Zhou, Qinghua, et al. "Accelerating distributed deep learning training with compression assisted allgather and reduce-scatter communication." 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2023.
4. "Enfabrica Scaling CXL Memory Using High Speed Networking ", https://www.youtube.com/watch?v=YdJWhqeT5DM
5. “MemVerge Flight simulator, ” https://memverge.com/cxl-qemuemulating-cxl-shared-memory-devices-in-qemu/
6. “QEMU-CXL branch,” https://gitlab.com/jic23/qemu
7. “OSU Micro-Benchmarks,” https://mvapich.cse.ohio-state.edu/benchmarks/
8. “OpenCIS,” https://www.opencis.io/
9. Qin, Ruoyu, et al. "Mooncake: Trading more storage for less computation—a {KVCache-centric} architecture for serving {LLM} chatbot." 23rd USENIX Conference on File and Storage Technologies (FAST 25). 2025.
10. **Lee, H., Kim, H., Park, J., & Son, J. (2025).** Disk-Based Shared KV Cache Management for Fast Inference in Multi-Instance LLM RAG Systems. *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD 2025)*
11. **Chen, S., Wu, Y., Wang, Z., Lin, M., Chen, Z., & Tang, J. (2025).** RetroInfer: A Vector-Storage Approach for Scalable Long-Context LLM Inference. *arXiv preprint arXiv:2505.02922.*
12. **Zhang, H., Xu, W., Liu, J., Wu, L., Sun, L., Wang, Z., … & Liu, X. (2025).** AlayaDB: The Data Foundation for Efficient and Effective Long-Context LLM Inference. *arXiv preprint arXiv:2504.10326.*
13. Kwon, Woosuk, et al. "Efficient memory management for large language model serving with pagedattention." Proceedings of the 29th symposium on operating systems principles. 2023.
14. “P2P NCCL Connector,” https://docs.vllm.ai/en/latest/design/p2p_nccl_connector.html#overall-process

Thank You!

Contacts : ahnhy@etri.re.kr