OMB-CXL: A Micro-Benchmark Suite for Evaluating MPI Communication Utilizing Compute Express Link Memory Devices

Tu Tran
tran.839@osu.edu
The Ohio State University
Columbus, Ohio, USA

Mustafa Abduljabbar
abduljabbar.1@osu.edu
The Ohio State University
Columbus, Ohio, USA

Hooyoung Ahn
ahnhy@etri.re.kr
Electronics and Telecommunications Research Institute
South Korea

Seonyoung Kim
seonyoung8436@etri.re.kr
Electronics and Telecommunications Research Institute
South Korea

Yoomi Park
parkym@etri.re.kr
Electronics and Telecommunications Research Institute
South Korea

Woojong Han
woojong.han@etri.re.kr
Electronics and Telecommunications Research Institute
South Korea

Shinyoung Ahn
syahn@etri.re.kr
Electronics and Telecommunications Research Institute
South Korea

Hari Subramoni
subramoni.1@osu.edu
The Ohio State University
Columbus, Ohio, USA

Dhabaleswar K. Panda
panda@cse.ohio-state.edu
The Ohio State University
Columbus, Ohio, USA

ABSTRACT

Compute Express Link (CXL) is a promising technology providing connectivity between host processors and peripheral devices like accelerators or memory modules. Compute nodes are usually connected through a high-speed network like Ethernet or Infiniband. CXL provides another way for connectivity by having compute nodes connected through CXL switches. CXL devices can connect to the switches, granting resource pooling and sharing across nodes. Access latency to CXL memory is ~10x smaller than access to the memory of another node through network operations. This enables a more efficient way to communicate between nodes. Currently, no existing Message Passing Interface (MPI) libraries utilize CXL for inter-node communication. In this paper, we propose the usage of CXL memory devices to enhance message-passing communication across nodes. To demonstrate the benefit of CXL, we extend OSU Micro-Benchmark (OMB), a well-known MPI benchmark suite, to evaluate point-to-point communication going over CXL; the extended OMB is named OMB-CXL. As CXL technology is evolving and still under development, the availability of a CXL system with switches and memory devices is limited. We describe how to set up an emulated CXL system using QEMU, a virtualization software, for early experience. The experimental results show that communication over CXL is 15x better for small messages and 4x for larger ones in latency and bandwidth than over the network on average.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; **Network measurement**; **Network simulations**; • **Computer systems organization** → *Interconnection architectures*; • **Hardware** → *Networking hardware*; • **Computing methodologies** → *Parallel computing methodologies*; *Distributed computing methodologies.*

KEYWORDS

MPI, CXL, micro-benchmarks, memory devices

ACM Reference Format:
Tu Tran, Mustafa Abduljabbar, Hooyoung Ahn, Seonyoung Kim, Yoomi Park, Woojong Han, Shinyoung Ahn, Hari Subramoni, and Dhabaleswar K. Panda. 2024. OMB-CXL: A Micro-Benchmark Suite for Evaluating MPI Communication Utilizing Compute Express Link Memory Devices. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3626203.3670533

1 INTRODUCTION

Modern computing systems consist of compute nodes, with multi-core processors, connected through a high-speed network like Ethernet or InfiniBand. Such systems work under parallel and distributed computing schemes. Communication between processes is expressed in forms of memory access within or across nodes. While intra-node communication is done through memory copies over the internal memory bus of local memory, inter-node communication is done through network operations. Access latency to local memory is at least an order of magnitude faster than to remote memory through the network [7]. This leads to performance bottlenecks in inter-node communication. Compute Express Link (CXL) [6] is expected to bridge the latency gap between local memory and memory access over network; CXL serves as a cache coherent interconnect between compute nodes and peripheral devices like network adapters or accelerators. A preliminary study showed that access latency to CXL memory is 5x slower than local memory but 6x faster than remote memory over the network [10]. When access data can fit into cache, CXL memory has a similar latency to the one of local memory.

1.1 Motivation and Challenge

Compute nodes typically utilize high-speed networks such as Ethernet or InfiniBand for connectivity. CXL offers an alternative approach by connecting compute nodes through CXL switches. CXL devices can then connect to the switches, facilitating resource pooling and sharing across nodes. In addition, access latency to CXL memory is significantly lower, approximately one-tenth, compared to accessing the memory of another node via network operations. This enables more efficient inter-node communication using CXL rather the network.

Message Passing Interface (MPI) [18] is the de facto programming model for developing and executing applications in a parallel and distributed fashion. MPI provides a set of routines for expressing fundamental communication patterns; they can be classified into two main groups: point-to-point and collectives. While point-to-point involves send and receive operations between two processes, collectives allow us to perform group communication. MPI libraries utilize shared memory and network for intra-node and inter-node communication, respectively. To assess communication performance using MPI, the OSU Micro-Benchmark suite (OMB) [20] is widely recognized within the MPI community for this purpose. Currently, no existing libraries utilize CXL underneath for inter-node communication despite CXL delivering 10x lower latency than network [7]. This motivates us to extend OMB to evaluate the performance of MPI over CXL, a system software that we refer to as OMB-CXL.

CXL is a fresh evolving technology and still under development; most of the available CXL machines and devices are prototypes and commercial products. The limited availability of CXL-enabled systems poses a challenge in doing system-level performance studies on CXL environment. Software development is always one step behind hardware availability. Software for CXL cannot be fully developed until there is an actual CXL-ready system for study and development. This motivates us to adopt an emulated CXL system in QEMU for preliminary study evaluation, and development. QEMU (Quick EMUlator) [24] is an open-source machine emulator and virtualizer; it can emulate hardware components such as processors, memory, or input/output devices.

Despite the evaluation in this paper being conducted in an emulated CXL environment, OMB-CXL is compatible with actual CXL-enabled hardware systems and can work out of the box. In this paper, we propose a proof of concept, through benchmarking, on how CXL can be leveraged for inter-node communication with lower latency than network operations. The contributions of this paper are as follows:

- We propose OMB-CXL to evaluate MPI performance using CXL for internode data exchanges.
- We demonstrate how to utilize CXL for the implementations MPI send/recv operations.
- We enhance the performance of P2P communication in terms of latency, bandwidth, and bi-bandwidth when using CXL instead of network.
- We present a detailed description of how we set up an emulated CXL system.

To the best of our knowledge, OMB-CXL is the first MPI micro-benchmark suite that supports the evaluation of communication performance through CXL. The remainder of this paper is organized as follows: Section 2 gives an overview of the CXL technology with a focus on CXL memory devices. Section 3 describes (1) the architecture of a CXL system sharing memory devices and (2) the extension of OMB, called OMB-CXL, to benchmark communication over CXL. Section 4 presents how we can set up an emulated system in QEMU for evaluation with two compute nodes sharing a CXL memory device; the two nodes are also connected through Ethernet. Section 5 presents the performance evaluation of OMB-CXL with analysis. The literature of CXL is presented in Section 6. We conclude the work in Section 7.

2 COMPUTE EXPRESS LINK - CXL

In this section, an overview of Compute Express Link (CXL) is provided. Since the focus of this paper is on the usage of type-3 CXL memory devices instead of network channel to better facilitate message-passing communication, relevant information on CXL memory and other memory types is discussed. Finally, we report the status of CXL support and implementations for host CPUs and devices. We also mention relevant technologies to CXL.

CXL is an open standard interconnect for processors, devices such as accelerators, network adapters, and memory expansion devices [6]. In the simplest terms, CXL is an improved version of PCIe with cache coherency and memory semantics. Cache coherency provides a consistent view of shared data to host processors and CXL devices. Memory semantics in CXL enables a unified memory address space between host processors and devices. Processors can directly load/store from/to devices' memory and devices can do the same to host memory. CXL leverages the physical layer of PCIe and is built on top of it. A CXL device can directly plug in a PCIe slot and operate out of the box as long as both the host central processing unit (CPU) and the device support CXL protocols. CXL consists of three protocols: (1) CXL.io for device management such as device discovery, and configuration, (2) CXL.cache providing device access to host CPU memory, (3) CXL.memory providing host CPU access to device memory.

CXL defines three types of devices. The first type is a caching device; an example is a Network Interface Card (NIC) having a small amount of cache with no memory. The second type is an accelerator such as a Graphics Processing Unit (GPU) or Field-Programmable Gate Array (FPGA). In contrast to type 1 device, type 2 not only has cache but also memory. The third type is a memory device; it is mainly used for memory capacity or bandwidth expansion. It can be also used as a storage class memory, also known as non-volatile memory.

CXL has evolved through three generations since its first release in 2019. CXL 1.0 & 1.1 main focus on a single-machine case to provide cache coherence and memory semantics on top of PCIe. CXL 2.0 enables resource pooling between a small number of machines

Table 1: Characteristics of CXL memory, main memory, and disaggregated memory (memory accessed over a network), and latency increase factor compared to local DRAM access

Memory Type	Latency (ns)	Connection Type	Latency Increase Factor
Main memory	80-140	CPU-attached	1x
CXL	170-250	CPU-independent	~ 2 - 3x
Disaggregated memory	2000-4000	Network-attached	~ 25 - 50x

with a single CXL switch. CXL 3.0 and 3.1 focus on large-scale pooling and sharing with multiple CXL switches.

This paper focuses on the usage of type-3 CXL memory devices over network channels to facilitate communication across nodes for better performance. Characteristics of CXL and other memory types are demonstrated in Table 1 [7]. CXL memory is expected to bridge the gap between main memory and disaggregated memory and overcome their disadvantages. Main memory like DRAM is attached to CPU through the Double Data Rate (DDR) parallel interface. As compute capabilities grow exponentially, so does the demand for memory capacity and bandwidth. However, the DDR has not been able to keep up with the demand due to the requirement of a large number of signal pins. CXL leveraging the PCIe physical layer provides a promising solution. An x16 Gen5 PCIe port provides 256 GB/s with just 64 signal pins compared to 50 GB/s of DDR5-6400 with around 200 signal pins [26]. Disaggregated memory refers to a distributed system architecture in which memory resources are physically located on different compute nodes and connected through a high-speed network such as Ethernet or InfiniBand. Memory accesses happen over the network, hence this memory type is network-attached. Despite its advantages of having better resource utilization and being more scalable, it is roughly one order of magnitude slower than main memory or CXL.

For CXL to work, it requires support in both host CPUs and devices. For CPU, CXL 1.1 is supported in recent architectures like Intel Sapphire Rapids [1] and AMD EPYC Genoa [3]. On the device side, many IP vendors such as Synopsys, Cadence, PLDA/Rambus, Mobiveil, Samsung, Micron, SK Hynix, and others have announced their CXL-ready devices [26]. Intel Agilex7 FPGA [11] provides support for all three CXL protocols.

There are similar technologies to CXL that provide cache coherency such as NVLink [21], OpenCAPI (Coherent Accelerator Processor Interface) [22] and Gen-Z [9], CCIX (Cache coherent interconnect for accelerators) [5]. NVLink is a proprietary interconnect of Nvidia, providing connectivity between GPUs. OpenCAPI and Gen-Z are absorbed by CXL [22]. Many CPU vendors originally announced CCIX support. However, this support was never released.

3 DESIGNING OMB-CXL

3.1 Architecture of a CXL system

CXL memory devices can provide connectivity between multiple compute nodes besides their ability for memory capacity and bandwidth expansion. Figure 1 depicts a distributed system with two nodes connected through two channels: network and CXL. As a result, there are two ways to communicate between the nodes. CXL memory is interleaved and formed by multiple memory devices just like the main memory of a computer node is formed by multiple DRAM sticks. CXL memory is managed using "cxl" utility; more detail is discussed in Section 4. CXL supports both memory pooling and sharing. In this paper, we are interested in the usage of memory sharing to facilitate communication. In memory sharing, CXL memory becomes a shared memory region that compute nodes can directly access with load/store instructions.

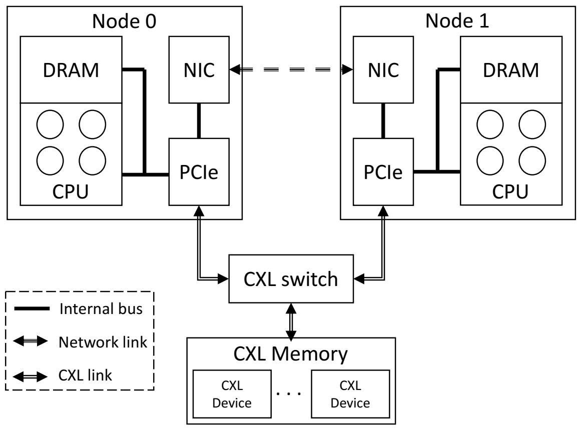


Figure 1: A distributed system connecting two nodes through a network and CXL memory.

3.2 Extending OMB to support CXL

In this paper, we are interested in benchmarking message-passing performance through the CXL channel since CXL provides one order of magnitude lower in latency than the network (Table 1). This enables a more efficient way to communicate across nodes. Currently, there is no existing libraries utilizing CXL under neath for inter-node communication. As a result, all inter-node communication goes through the network in MPI. To assess communication performance using MPI, the OSU Micro-Benchmark suite (OMB) [20] is widely recognized within the MPI community for this purpose. This motivates us to extend OMB to evaluate the performance of MPI over CXL. We refer to it as OMB-CXL.

Figure 2 describes how latency, bandwidth, and bi-bandwidth are benchmarked in OMB through MPI send and receive operations.

- The latency test measures the time it takes to send a message size M from one process to another. The test is performed in a ping-pong fashion. Process P0 sends a message size M to P1. P1 waits for the message before sending another message of the same size to P0. The time it takes for P0 to send and receive a message from P1 is called round-trip time (RTT). Latency is calculated by dividing RRT by half.

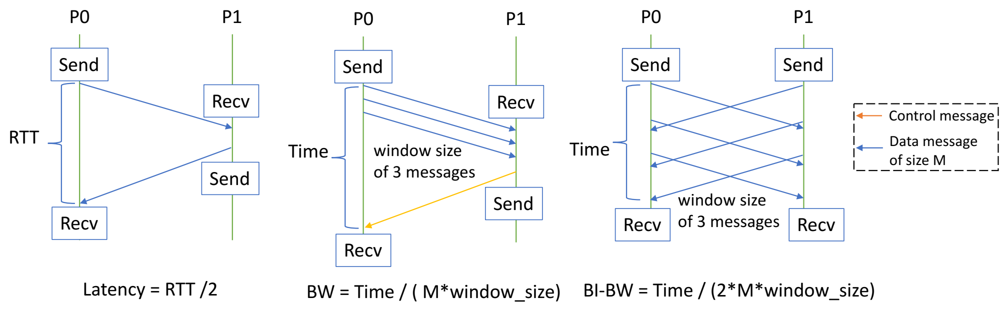

P0
P1
Send
Recv
RTT
Latency = RTT /2
Time
window size of 3 messages
BW = Time / (M*window_size)
BI-BW = Time / (2*M*window_size)
Control message
Data message of size M

Figure 2: Benchmarking latency, bandwidth, and bi-bandwidth with OMB.

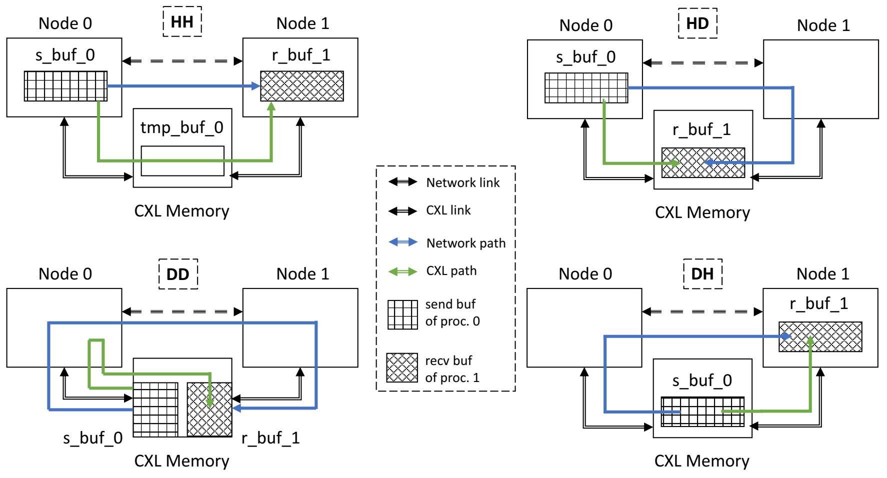


Figure 3: Comparison of different communication paths of a send operation through network and CXL channels under different buffer placement configurations.

- The bandwidth test is used to measure the data transfer rate or throughput between two endpoints or processes in a network. Process P0 sends multiple messages back-to-back in order to saturate the network bandwidth between two processes. The number of messages P0 sends is referred to as window size. After P1 receives all the messages of size M, P1 sends a control message of four bytes to P0 to indicate the successful receive of all messages. The time taken from the start of the first message P0 sent to the arrival of the control message is measured; it is used to calculate the unidirectional bandwidth from P0 to P1 of the network connecting two processes. The formula for the calculation is stated in Figure 2.
- The bi-bandwidth test is similar to the bandwidth test; it measures the bi-directional bandwidth of the network link between P0 to P1. Each process sends *window_size* messages of size M to each other. The time from the start of the first message being sent to the last one being received is measured in P0 for the evaluation of bi-directional bandwidth. The bi-directional bandwidth value is expected to be two times the value of the uni-directional bandwidth due to the full duplex property of the network.

We demonstrate the usage of CXL to improve point-to-point communication through the aforementioned tests: latency, bandwidth, and bi-bandwidth. Figure 3 demonstrates communication paths between send and receive processes on two compute nodes: node 0 and 1, respectively, using MPI over CXL and network channels. Send and receive buffers can be allocated on host or CXL memory, resulting in 4 variants for buffer placement: host-host(HH), device-device(DD), host-device (HD), and device-host (DH). When using the network channel, communication always goes through the network link between two nodes; depending on where send and receive buffers are allocated, read and write to local and CXL memory are also performed. For CXL, the communication path for each variant is as follows:

- HH: both send and receive buffers are allocated on the local memory of each node. In the CXL channel, the sender can put the data in a temporary buffer in the CXL memory for the receiver to copy out.
- DD: both send and receive buffers are allocated on the CXL shared memory. Either sender or receiver can invoke a memory copy to put data from the send buffer to receive buffer.

Compared to the network path, CXL path does not go through the network link and, hence shorter.

- HD: send and receive buffers are allocated on the local memory of node 0 and CXL memory, respectively. Sender can direct copy send buffer to receive buffer. CXL path does not go through the network link, hence shorter than network path.
- DH: send and receive buffers are allocated on CXL memory and the local memory of node 1, respectively. Receiver can direct copy send buffer to receive buffer. CXL path here is also shorter than network path.

Listing 1: A code snip of how to utilize CXL to send a message from process 0 to process 1 under DD buffer placement

```
// P0 sends to P1
memcpy(r_buf, s_buf, size);
// After data being copied, P0 sets a flag in the shared
    memory indicate data availability for P1 to use.
SET_DATA_FLAG(dataflag);

// P1 receives from P0
// P1 waits for data ready signal from P0
WAIT_TIL_DATA_FLAG_SET(dataflags);
```

Listing 1 demonstrates how to utilize CXL to implement a send from one process to another under DD buffer placement. The same idea is applied to other variants. Process P0 can simply invoke a memory copy to put data from send buffer to recv buffer. Once the operation is complete, P0 must set a flag in the shared CXL memory region indicating data availability for P1 to use. Before the arrival of the data, P1 will poll the status of the data flag to check for the ready signal. Because a process either reads or writes to the data flag, we must add appropriate memory fence instructions to avoid compiler reordering instructions and out-of-order execution.

3.3 OMB-CXL benchmarks usage

Based on what we discussed above, the point-to-point benchmarks are modified to support four variants of buffer placement running over CXL. Users now can specify running over CXL using '-C' option passed in as an argument to osu_latency, osu_bw, and osu_bibw. Following the option, users must specify where the send and recv buffers are allocated using 'H'- allocated on host memory or D - allocated on CXL device memory. As a result, there are four variants that we can specify after the '-C' option with a space: HH, HD, DD, or DH.

4 EMULATING A CXL SYSTEM WITH QEMU

Quick EMUlator (QEMU) [24] is an open-source machine emulator and virtualizer; it can emulate hardware components such as processors, memory, or input/output devices. QEMU supports CXL2.0 and above with the capability to create single-level switching and memory devices [25]. Willis and Price from Memverge demonstrated how to set up a CXL system with QEMU [17]. In this section, we discuss what is not covered in their instruction. Specifically, we discuss how to emulate a CXL system with two nodes connected through Ethernet and sharing a CXL memory device. We also discuss how to manage CXL memory devices in Linux with 'cxl' utility and make it a direct access (DAX) device for memory sharing.

Listing 2: QEMU configuration for constructing a machine connected to a CXL memory device and Ethernet

```
$ qemu-system-x86_64 \
...
-machine type=q35,cxl=on \
-device pxb-cxl,id=cxl.0,bus=pcie.0,bus_nr=52 \
-device cxl-rp,id=rp0,bus=cxl.0,chassis=0,port=0,slot=0 \
-device cxl-type3,bus=rp0,volatile-memdev=mem0,id=cxl-
    mem0 \
-object memory-backend-file,id=mem0,mem-path=/tmp/mem0,
    size=4G,share=true \
-M cxl-fmw.0.targets.0=cxl.0,cxl-fmw.0.size=4G \
-netdev socket,mcast=230.0.0.1:1234,id=net1 \
-device virtio-net-pci,mac=52:54:00:12:34:01,netdev=net1
```

Listing 2 shows the QEMU configuration for creating a machine connected to a CXL device of 4GB. The command instructs QEMU to create a CXL Host Bridge which has a CXL Root Port directly attaching to a CXL memory device. A CXL Root Port can also attach to a CXL switch by creating a single upstream port (cxl-upstream) and several downstream ports (cxl-downstream). The command also instructs to create an Ethernet interface. The command is run two times to create a CXL system with two nodes connected through Ethernet and sharing a CXL memory device.

Listing 3: Configuring a CXL memory device and it a DAX device

```
$ sudo cxl create-region -m -t ram -d decoder0.0 -w 1 -g
    4096 mem0
$ sudo daxctl reconfigure-device -m devdax dax0.0
```

CXL devices in Linux systems are managed through "cxl" command. It is similar to non-volatile device control (ndctl) utility; they are under the same project encompassing tools and libraries for managing and configuring CXL and non-volatile memory devices [23]. Listing 3 shows how to configure a CXL memory device using "cxl" and "daxctl" commands to make it Direct Access (DAX). The first command is used to create a memory region using CXL memory device "mem0" and decoder "decoder0.0". The "-w" and "-g" options allow us to specify the number of interleave ways and the interleave granularity. As a result, we can create a multi-way interleaved memory region across multiple CXL devices with a certain interleave granularity. The CXL memory region appears to Linux as a remote NUMA node with processors. The next command in the Listing 3 makes the CXL memory region DAX. This allows applications to directly access the memory as a memory-mapped file (use mmap() function), making it a shared memory region to which machines can have access.

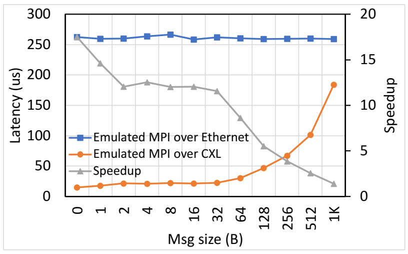
5 PERFORMANCE EVALUATION WITH OMB-CXL

5.1 Experimental environment

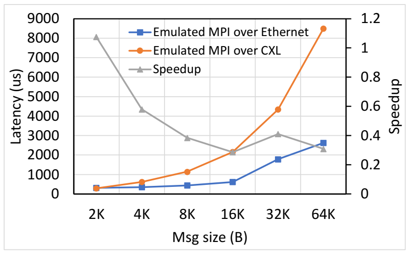
All the experiments are evaluated on an emulated CXL system with an architecture similar to Figure 1. The system has two x86_64 compute nodes, with 4 cores each, interconnected through an Ethernet adatper. The two nodes are also connected to a CXL memory device. The host machine used to emulate the CXL system is equipped with dual Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz CPU of 28

Table 2: Memory access latency and bandwidth over CXL and Ethernet under the emulated CXL system using QEMU

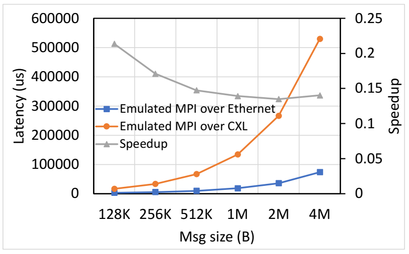
Memory type	Latency	Bandwidth
CXL memory	7us	7MB/s
Remote memory over Ethernet	150us	60MB/s



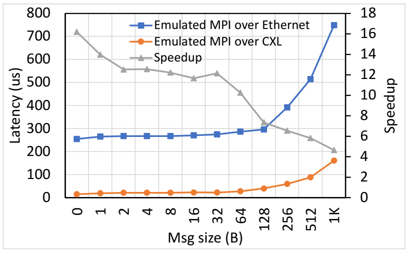
(a) HH - small message range



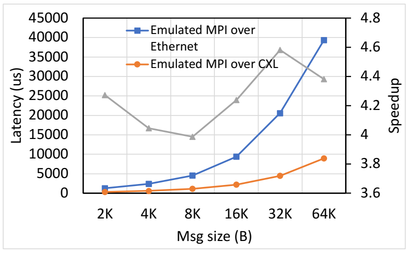
(b) HH - medium message range



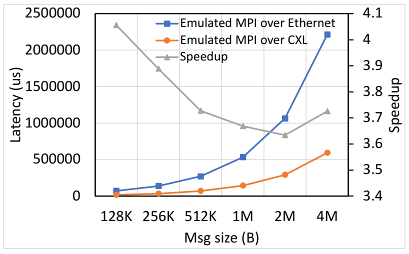
(c) HH - large message range



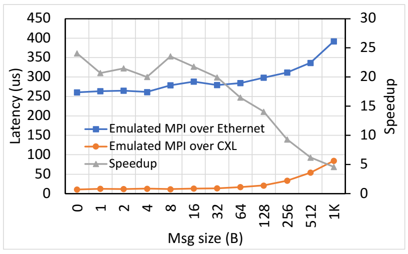
(d) DD - small message range



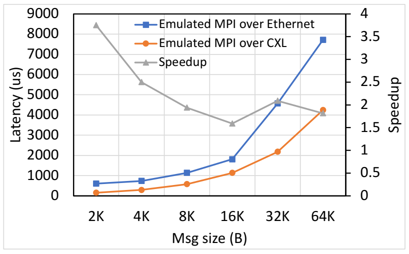
(e) DD - medium message range



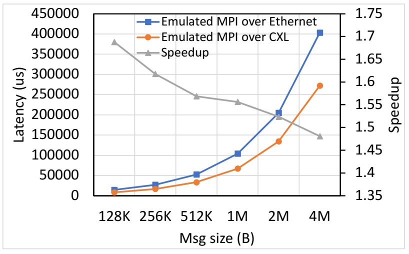
(f) DD - large message range



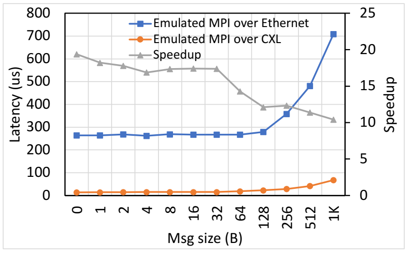
(g) HD - small message range



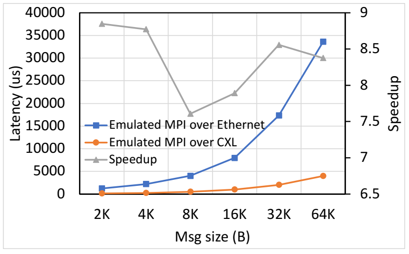
(h) HD - medium message range



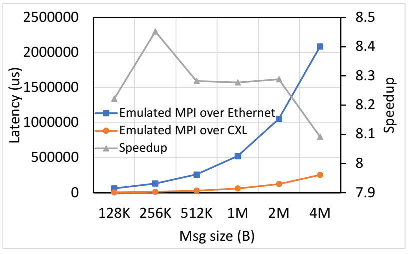
(i) HD - large message range



(j) DH - small message range



(k) DH - medium message range



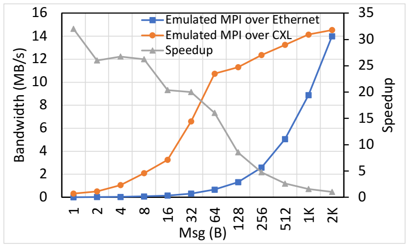
(l) DH - large message range

Figure 4: Inter-node latency test of different buffer placements of send buffer of process 0 and recv buffer of process 1. Send and recv buffers can be allocated on host(H) or device (D) memory, resulting in four configurations: HH, DD, HD, and DH.

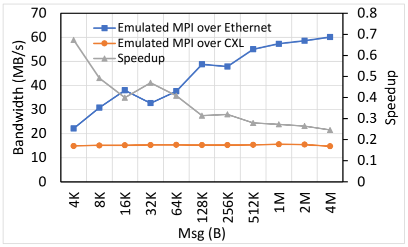
cores and 128 GB of DDR4 memory. The two emulated nodes run Fedora Linux 38. The MPI library used in this paper is MVAPICH v3.0-rc [19] along with UCX 1.15.0 [28], and the OMB-CXL we use for experimenting is extended from OMB v7.0 [20]. We run each experiment five times to remove any noise or fluctuation. Within each OMB-CXL run, each message has an average of 1000 iterations for small size and 100 iterations for large.

5.2 Performance evaluation

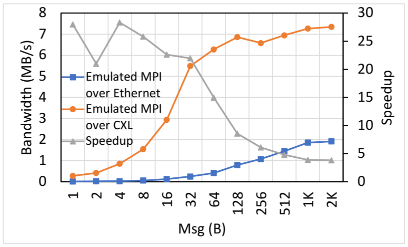
After setting up the emulated CXL system connected by Ethernet with two nodes sharing a CXL device, we perform basic low-level



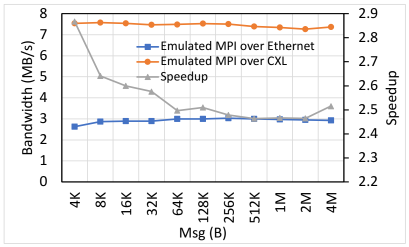
(a) HH - small message range



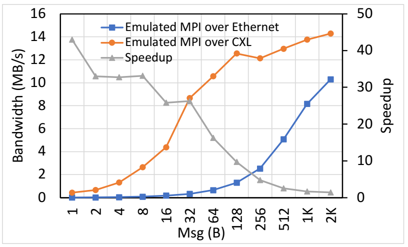
(b) HH - large message range



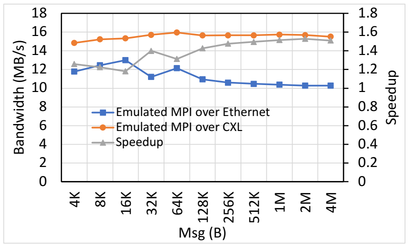
(c) DD - small message range



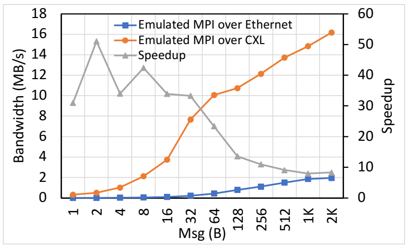
(d) DD - large message range



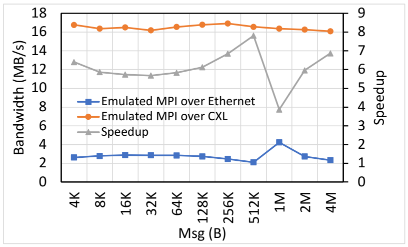
(e) HD - small message range



(f) HD - large message range



(g) DH - small message range



(h) DH - large message range

Figure 5: Inter-node bandwidth test of different buffer placements of send buffer of process 0 and recv buffer of process 1. Send and recv buffers can be allocated on host(H) or device (D) memory, resulting in four configurations: HH, DD, HD, and DH.

latency and bandwidth tests using the network and CXL channels to access memory. The results are reported in Table 2. CXL memory is tested with Intel® Memory Latency Checker v3.11 [12] reporting latency of 7us with 7MB/s of access bandwidth. The network channel to access a remote memory is tested with iPerf [13] reporting latency of 150us with 60MB/s of uni-directional bandwidth. The above performance values of the emulated system summarized in Table 2 are worse than the performance values a real system should have in Table 1 due to overhead for emulation. However, **the emulated CXL system maintains a latency ratio between the CXL-connected memory device and the remote memory accessed over the network that is nearly identical to theoretical estimates as shown in Table 1**.

We next perform latency, bandwidth, and bi-bandwidth tests of point-to-point communication with two MPI processes located on two nodes. Figures 4 and 5 show the performance comparison of emulated MPI over CXL and Ethernet channels under four variants of buffer placement. **The performance numbers are not to be interpreted in absolute values due to overhead for emulating the CXL environment. An actual system will have lower latency as presented in Table 1. A more reasonable metric for analysis should be based on the speedup of MPI over CXL and Ethernet.**

Among the MPI experiments, we first discuss the latency test illustrated in Figure 4. In the HH case, CXL has lower latency than Ethernet for the small message range with an average speedup of 9.5x. As the message size increases, the trend reverses with Ethernet performing better in latency than CXL due to the CXL channel having lower bandwidth than Ethernet in the emulated system. In an actual system, we expect that the bandwidth of a CXL channel should be at least equal to or better than any network, depending on the memory technology being used. CXL memory devices equipped with DDR4-2400 and DDR5-4800 are expected to deliver 19.2 GB/s and 38.4 GB/s per memory channel theoretically [27]; the bandwidth of a high-speed network is around 100 to 200 Gbps. In the DD, HD, and HD cases, CXL performs better in latency than Ethernet across all message ranges. The CXL communication path being shorter than one over the Ethernet leads to a speedup of about 9x. The detailed communication paths over CXL and Ethernet are illustrated in Figure 3.

In the bandwidth test depicted in Figure 5, we observe similar trends to the ones of the latency test. In the HH case, CXL performs better than MPI in bandwidth for small messages with an average speedup of 9x until CXL bandwidth is saturated. In other cases: DD, HD, and HD, CXL shows higher bandwidth than MPI with an average speedup of about 10x. Since the bi-bandwidth test shows the same trends as the bandwidth test, the charts for them are not shown here.

6 RELATED WORK

CXL is a promising technology providing cache coherency between host processors and devices like network adapters or accelerators. It also provides scale-out capability by having many compute nodes connected through a CXL network. Compared to a conventional way of scale-out using a high-speed network like Ethernet or InfiniBand, CXL is expected to deliver one order of magnitude lower in latency. The CXL technology has direct implications on memory. Ahm et al. have demonstrated the benefits of using CXL type 3 devices as a memory expansion for in-memory computing [2]. CXL memory devices can also be used as persistent [8] or disaggregated memory [10]. On a broader scale, CXL influences system-level operations. CXL enables memory pooling for better resource utilization in HPC systems [29]. In cloud systems, memory pooling is used to eliminate memory stranding [16]; free memory in a compute node with no available processors can not be allocated to users. CXL technology is still evolving and has its limitations in latency, scalability, and capital cost [15, 16]. Due to limited availability in CXL-ready machines, several studies have been done in an emulated or simulated environment [2, 4, 8, 14, 16, 29]. In the context of MPI, we have not been able to find any study that utilizes CXL for communication. To the best of our knowledge, our work is the first CXL-enabled MPI work that leverages CXL for faster inter-node communication compared to using network.

7 CONCLUSION

CXL provides a more efficient to connect compute nodes than a conventional high-speed network like Ethernet or Infiniband. The technology is expected to bridge the gap in latency between local memory and remote memory accessed over network operations. In this paper, we demonstrate the benefits of using CXL for inter-node communication. We propose OMB-CXL, a benchmark suite to evaluate point-to-point communication in MPI through CXL channel. As CXL technology is still under development, there are few CXL systems with switches and memory devices for usage. We show how to set up an emulated CXL system using QEMU for early experience. Experiments using OMB-CXL show that CXL delivers a promising performance improvement of 4-15x compared to network for latency and bandwidth tests.

ACKNOWLEDGMENTS

This research was supported by the Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (No. 23YS1700). It is also supported in part by NSF grants #1818253, #1854828, #2007991, #2018627, #2311830, #2312927, and XRAC grant #NCR-130002.

REFERENCES

[1] 4th Gen Intel Xeon Scalable Sapphire Rapids 2024. *4th Gen Intel Xeon Scalable Sapphire Rapids*. https://www.intel.com/content/www/us/en/developer/articles/technical/fourth-generation-xeon-scalable-family-overview.html. Accessed June 14, 2024.

[2] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. 2022. Enabling CXL memory expansion for in-memory database management systems. In *Proceedings of the 18th International Workshop on Data Management on New Hardware*. 1–5.

[3] AMD EPYC 9004 Genoa 2024. *AMD EPYC 9004 Genoa CXL Overview*. https://www.servethehome.com/amd-epyc-genoa-gaps-intel-xeon-in-stunning-fashion/amd-epyc-9004-genoa-cxl-overview/. Accessed June 14, 2024.

[4] Moiz Arif, Kevin Assogba, M Mustafa Rafique, and Sudharshan Vazhkudai. 2022. Exploiting CXL-based memory for distributed deep learning. In *Proceedings of the 51st International Conference on Parallel Processing*. 1–11.

[5] CCIX 2024. *CCIX*. https://www.ccixconsortium.com/. Accessed June 14, 2024.

[6] Compute Express Link 2024. *Compute Express Link*. https://computeexpresslink.org/. Accessed June 14, 2024.

[7] CXL Memory Hierarchy 2024. *The Expanding Cxl Memory Hierarchy Is Inevitable – And Good Enough*. https://www.nextplatform.com/2022/08/22/the-expanding-cxl-memory-hierarchy-is-inevitable-and-good-enough/. Accessed June 14, 2024.

[8] Yehonatan Fridman, Suprasad Mutalik Desai, Navneet Singh, Thomas Willhalm, and Gal Oren. 2023. CXL Memory as Persistent Memory for Disaggregated HPC: A Practical Approach. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. 983–994.

[9] Gen-Z 2024. *Finally, A Coherent Interconnect Strategy: CXL Absorbs Gen-Z*. https://www.nextplatform.com/2021/11/23/finally-a-coherent-interconnect-strategy-cxl-absorbs-gen-z/. Accessed June 14, 2024.

[10] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct access,{High-Performance} memory disaggregation with {DirectCXL}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 287–294.

[11] Intel Agilex® 7 FPGAs 2024. *Intel® FPGA Compute Express Link (CXL) IP*. https://www.intel.com/content/www/us/en/products/details/fpga/intellectual-property/interface-protocols/cxl-ip.html. Accessed June 14, 2024.

[12] Intel MLC 2024. *Intel® Memory Latency Checker*. https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html. Accessed June 14, 2024.

[13] iPerf 2024. *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. https://iperf.fr/. Accessed June 14, 2024.

[14] Mikhail Isaev, Nic McDonald, and Richard Vuduc. 2023. Scaling infrastructure to support multi-trillion parameter LLM training. In *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*.

[15] Philip Levis, Kun Lin, and Amy Tai. 2023. A Case Against CXL Memory Pooling. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. 18–24.

[16] 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.

[17] Memverge 2024. *Emulating CXL Shared Memory Devices in QEMU*. https://memverge.com/cxl-qemuemulating-cxl-shared-memory-devices-in-qemu/. Accessed June 14, 2024.

[18] Message Passing Interface Forum. 2021. *MPI: A Message-Passing Interface Standard Version 4.0*. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[19] Network-Based Computing Laboratory 2024. *MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE*. http://mvapich.cse.ohio-state.edu/. Accessed June 14, 2024.

[20] Network-Based Computing Laboratory 2024. *OSU Micro-Benchmarks*. https://mvapich.cse.ohio-state.edu/benchmarks/. Accessed June 14, 2024.

[21] NVLink 2024. *NVLink & NVSwitch: Fastest HPC Data Center Platform*. https://www.nvidia.com/en-us/data-center/nvlink/. Accessed June 14, 2024.

[22] OpenCAPI 2024. *OpenCAPI to Be Folded into CXL*. https://www.hpcwire.com/2022/08/01/opencapi-to-be-folded-into-cxl/. Accessed June 14, 2024.

[23] pmem/ndctl 2024. *A device memory enabling project encompassing tools and libraries for CXL, NVDIMMs, DAX, memory tiering and other platform memory device topics*. https://github.com/pmem/ndctl. Accessed June 14, 2024.

[24] QEMU 2024. *QEMU - A generic and open source machine emulator and virtualizer*. https://www.qemu.org. Accessed June 14, 2024.

[25] QEMU-CXL 2024. *Compute Express Link (CXL) - QEMU Documentation*. https://www.qemu.org/docs/master/system/devices/cxl.html. Accessed June 14, 2024.

[26] Debendra Das Sharma, Robert Blankenship, and Daniel S Berger. 2023. An Introduction to the Compute Express Link (CXL) Interconnect. *arXiv preprint arXiv:2306.11227* (2023).

[27] 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.

[28] UCX 2024. *Unified Communication X*. https://openucx.org/. Accessed June 14, 2024.

[29] 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.