# Designing a ROCm-Aware MPI Library for AMD GPUs: Early Experiences

Kawthar Shafie Khorassani, Jahanzeb Hashmi(✉), Ching-Hsiang Chu(✉), Chen-Chun Chen(✉), Hari Subramoni(✉), and Dhabaleswar K. Panda(✉)

The Ohio State University, Columbus, OH 43210, USA
{shafiekhorassani.1,hashmi.29,chu.368,chen.10252}@osu.edu, {subramon,panda}@cse.ohio-state.edu

**Abstract.** Due to the emergence of AMD GPUs and their adoption in upcoming exascale systems (e.g. Frontier), it is pertinent to have scientific applications and communication middlewares ported and optimized for these systems. Radeon Open Compute (ROCm) platform is an open-source suite of libraries tailored towards writing high-performance software for AMD GPUs. GPU-aware MPI, has been the de-facto standard for accelerating HPC applications on GPU clusters. The state-of-the-art GPU-aware MPI libraries have evolved over the years to support NVIDIA CUDA platforms. Due to the recent emergence of AMD GPUs, it is equally important to add support for AMD ROCm platforms. Existing MPI libraries do not have native support for ROCm-aware communication. In this paper, we take up the challenge of designing a ROCm-aware MPI runtime within the MVAPICH2-GDR library. We design an abstract communication layer to interface with CUDA and ROCm runtimes. We exploit hardware features such as PeerDirect, ROCm IPC, and large-BAR mapped memory to orchestrate efficient GPU-based communication. We further augment these mechanisms by designing software-based schemes yielding optimized communication performance. We evaluate the performance of MPI-level point-to-point and collective operations with our proposed ROCm-aware MPI Library and Open MPI with UCX on a cluster of AMD GPUs. We demonstrate 3–6× and 2× higher bandwidth for intra- and inter-node communication, respectively. With the rocHPCG application, we demonstrate approximately 2.2× higher GFLOPs/s. To the best of our knowledge, this is the first research work that studies the tradeoffs involved in designing a ROCm-aware MPI library for AMD GPUs.

**Keywords:** ROCm · AMD GPUs · MPI

## 1 Introduction

Modern High-Performance Computing (HPC) systems are equipped with state-of-the-art accelerators including Graphics Processing Units (GPUs).

This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, and XRAC grant #NCR-130002.

B. L. Chamberlain et al. (Eds.): ISC High Performance 2021, LNCS 12728, pp. 118–136, 2021.
https://doi.org/10.1007/978-3-030-78713-4_7

Such systems are currently fueling the next generation of Artificial Intelligence (AI) and scientific applications. This trend is timely to avert the challenges presented by the end of Moore's law [15], which sustained performance growth for the last several decades. The use of GPUs is prevalent in many modern HPC and cloud systems for driving scientific applications and Machine Learning workloads. However, it is important to innovate further by evolving and expanding the support provided by GPUs to meet the ever-increasing computational requirements of next-generation applications.

The landscape of accelerator-based computing is currently dominated by NVIDIA GPUs. However, other alternatives like Radeon Instinct devices (GPUs) from AMD and Xe GPU from Intel have recently started emerging. In particular, AMD GPUs offer a promising platform that has been adopted by upcoming next-generation exascale systems such as Frontier [3] and El Capitan [6]. In addition to these up-and-coming systems, a current compute platform, the Corona cluster at Lawrence Livermore National Laboratory [2], is also equipped with 291 nodes consisting of AMD Mi50 and AMD Mi60 GPUs. Of these nodes, 123 of them are equipped with AMD EPYC 7002 series CPU nodes, with each node consisting of 8 AMD Radeon Instinct MI50 GPU accelerators.

Prior to the emergence of AMD GPUs, NVIDIA GPU platforms have been the *defacto* standard for exploiting GPUs within applications for communication and computation tasks. NVIDIA GPUs rely on NVIDIAs in-house toolkit called Compute Unified Device Architecture (CUDA) to support GPU-accelerated high-performance applications. In the past, applications wanting to use AMD GPUs often had to rely on the OpenCL library, which made it difficult to port applications while CUDA as a programming system was much more developed. Recent efforts by AMD has resulted in Radeon Open Compute (ROCm) software stacks that offer seamless support for high-performance libraries required for efficient computation and communication on modern AMD GPU hardware. ROCm is an open-source toolkit provided by AMD consisting of libraries, profilers, and APIs used in the development of high-performance software for AMD GPUs. An important feature offered by ROCm is HIP [14]—a C++ Runtime API and kernel language that allows developers to create portable applications for AMD and NVIDIA GPUs. In most cases, HIP offers one-to-one mappings of API calls between CUDA and ROCm and provides tools for automatic translation from CUDA to HIP code. This source-to-source translation, also referred to as *hipification*, has helped in seamlessly porting application codes to AMD hardware.

The Message Passing Interface (MPI) standard, is considered a *defacto* API for writing parallel programs on modern HPC systems. In order to accelerate large-scale high-performance applications on GPU clusters, GPU-aware MPI has been the widely adopted programming model in use. The state-of-the-art MPI libraries have evolved over the years to incorporate GPU-aware communication support at the MPI layer. This is also referred to as CUDA-aware MPI as it entails support for NVIDIA CUDA platforms due to the dominance of NVIDIA GPUs in the hardware configuration of GPU-enhanced clusters. The emergence of AMD GPUs and their adoption in upcoming exascale systems makes it important to have support for AMD ROCm platforms in modern MPI libraries. Current MPI implementations do not have native support for direct communication between device resident data on AMD GPUs or ROCm-aware communication. In order to accelerate scientific applications, Machine Learning workloads, and to have parallel applications ready to scale on next-generation exascale systems with AMD GPU hardware configurations, it is crucial to have the appropriate support at the middleware level by designing a ROCm-aware MPI runtime.

**In this paper, our goal is to design a ROCM-aware MPI runtime which brings about the following challenges: 1)** How can we design an abstract and extensible communication layer for MPI libraries that interfaces with both the CUDA and the ROCm run-times? **2)** Can we appropriately make use of the various features supported by ROCm including ROCm IPC, ROCm-RDMA (PeerDirect), etc., and identify the ranges in which each of these features is optimal for data transfer? **3)** How can we utilize unified memory and AMD's Large Bar mapped memory feature to optimize the performance of MPI operations?

## 1.1 Contributions

In this paper, we design a ROCm-aware implementation of MPI developed over MVAPICH2-GDR by delving into the details and challenges of utilizing existing hardware and software. The challenge here is to properly extend the native support within the MPI library to run with ROCm on AMD GPUs. We design a communication layer that is able to interface with both CUDA for NVIDIA GPUs and ROCm for AMD GPUs and derive MPI operations seamlessly. We evaluate the proposed ROCm-aware MPI implementation against Open MPI with UCX as the ROCm-aware communication backed on the Corona Cluster at the benchmark-level and with ROCm-enabled applications. In summary, the paper incorporates the following contributions:

- Design an abstract and extensible communication layer in the MPI runtime to interface with both CUDA and ROCm run-times to drive MPI communication.
- Identify challenges with utilizing existing hardware and software configurations for enabling ROCm-aware MPI communication.
- Propose new designs in the MPI library to exploit AMD GPUs using ROCm libraries and features e.g., ROCm PeerDirect, ROCm IPC, and unified memory.
- Incorporate tuning based selection of ROCm designs for MPI protocols (e.g., eager vs. rendezvous) for appropriate message ranges.
- Comprehensive evaluation of MPI point-to-point and collective operations for GPU resident data and comparing our proposed ROCm-aware MPI implementation against state-of-the-art communication libraries (Open MPI + UCX).

– Evaluate the efficacy of our proposed ROCm-aware MPI using various applications such as a 3DStencil, and HPCG and compare the performance against Open MPI + UCX on the LLNL Corona cluster.

**To the best of our knowledge, this is the first research work that studies and analyzes the tradeoffs involved in designing a ROCm-aware MPI library for AMD GPUs.**

## 2 Background

### 2.1 Radeon Open Compute (ROCm)

ROCm [5] is an open-source software platform tailored towards high-performance computing and Machine Learning on AMD GPUs. It consists of tools for development on GPUs, APIs, and drivers that support AMD GPUs. ROCm also has support for various programming models such as OpenMP, OpenCL, and HIP. It has recently been integrated with many scientific applications such as HPCG, NAMD, GRID, and GROMACS and Machine Learning frameworks including TensorFlow, Pytorch, RAJA, and Kokkos.

### 2.2 ROCm Remote Direct Memory Access (RDMA)

ROCm RDMA enables third-party devices such as the Mellanox Infiniband HCA device to have a direct peer-to-peer data path with GPU memory. This removes CPU intervention from communications between GPUs across the network, further enhancing communication latency between GPU-GPU transfers.

### 2.3 Inter-Process Communication (IPC)

IPC is used to address overheads associated with data transfer between GPUs within a node. The ROCm platform has support for the IPC interface allowing a process to expose its GPU buffer to a remote process, optimizing the movement of data between GPUs. This allows for directly implementing an MPI call over GPU device memory. A remote process could directly call *deviceMemCpy* on the exposed IPC handle from the sender process leading to optimized data transfer between GPUs.

### 2.4 Message Passing Interface (MPI)

The Message Passing Interface (MPI) is a programming paradigm used to enable communication amongst processes for parallel applications. There are multiple communication primitives within MPI including one-sided, point-to-point, and collective operations. One-sided communication, also referred to as remote memory access (RMA), involves one process communicating with another without any intervention from the remote process. A process sends data to a receiving process without requiring synchronization, eliminating this step from the data transfer process. Point-to-point operations involve direct communication between a sender process and a receiver process, and unlike the non-blocking nature of one-sided communication, it requires some synchronization between the two processes involved. Collective communication refers to multiple processes communicating with one or many processes.

In this work, we focus on point-to-point and collective communication and what factors to consider when making these operations ROCm-aware. Through CUDA-aware MPI functionality in various MPI libraries such as MVAPICH2-GDR [16] and Open MPI [11], these operations can be run on NVIDIA GPUs. They utilize various schemes to optimize and enhance the GPU-based communication through GPUDirect RDMA and IPC. In order to extend these operations to run on AMD GPUs, we require a ROCm-aware implementation of MPI. We evaluate Open MPI + Unified Communication X (UCX) [7] against our proposed development, where UCX is used as the ROCm-aware communication backend to support AMD GPU runs since Open MPI is not a stand-alone ROCm-aware MPI library.

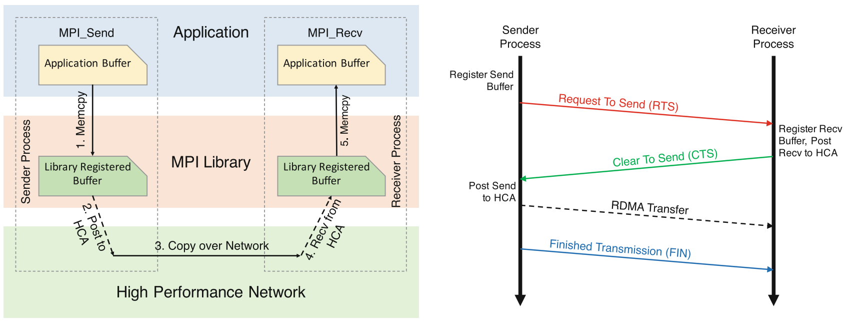
### 2.5 Protocols for High-Performance Communication in MPI

Figures 1(b) and 1(b) depict how the eager and rendezvous protocol respectively are typically implemented. The eager protocol consists of four steps—1) copying the data from the application buffer to buffers internal to the MPI library, 2) initiating the data transfer to the remote process, 3) detecting the reception of data in buffers internal to the MPI library and, 4) copying the data back to the application buffer. With most high-performance networks like InfiniBand, the network itself takes care of the actual data transfer. Thus, initiating the data transfer at the sender and detecting the reception of the data at the receiver are low overhead tasks. So, apart from the time to transfer data over the network, the main costs involved in an eager transfer are the memory copies at the sender/receiver. Note that steps #1 and #2 happen inside the send function call itself. With a rendezvous protocol on the other hand (Fig. 1(b)), MPI designers take advantage of the RDMA feature that high-performance interconnects like InfiniBand offers and transfers data directly from the source application buffer to the target application buffer (with the appropriate exchange of control information), thereby avoiding the extra large memory copies from the application buffer to internal communication buffers within the library.

## 3 Designing and Implementation of ROCm-Aware MPI

### 3.1 Overview of Technologies Offered by NVIDIA and AMD for GPU Based Communication

As discussed earlier, the state-of-the-art GPU-aware MPI libraries have supported NVIDIA GPUs and hence, the communication designs employed by these MPI libraries were highly CUDA specific. For example, two popular GPU-aware



(a) Eager Protocol (b) Rendezvous Protocol

**Fig. 1.** Point-to-point communication protocols in MPI

MPI libraries MVAPICH2-GDR and Open MPI supported NVIDIA's GPUDirect technology for communicating GPU resident data over RDMA networks. However, with the newer AMD GPU and ROCm stacks offering similar technologies, the MPI libraries have to go through the same route by evaluating various technologies offered by ROCm. We summarize the key communication technologies offered by both the vendors and their similarities/differences in Table 1 below:

**Table 1.** Similarities and differences between CUDA and ROCm communication features used by GPU-aware MPI libraries

| | Technology | | Dependency | |
|---|---|---|---|---|
| | NVIDIA | AMD | NVIDIA | AMD |
| RDMA Support | GPUDirect RDMA | ROCmRDMA (PeerDirect) | `nv_peer_mem` Kernel Module | ROCm Driver (no kernel module) |
| Peer-to-peer | CUDA IPC | ROCm IPC | CUDA Runtime | ROCr Runtime |
| Mapped Copy | GDRCopy BAR1 | Large BAR Feature | GDRCopy Kernel Module | ROCm Driver (no kernel module) |

As demonstrated, most of the features provided by AMD are integrated into the ROCm driver or the ROCr runtime while NVIDIA often requires separate modules to enable features like GDRCopy and GPUDirect RDMA. The unified package offered by AMD is advantageous since most HPC centric clusters often do not install separate kernel modules due to security concerns.

## 3.2 Designing Unified Device Abstraction Interface for Accelerator-Aware MPI

In order to avoid the duplication of efforts when designing ROCm-aware MPI, we propose a unified device abstraction interface (UDI) in the MPI runtime. The purpose of this abstraction is to seamlessly interface with vendor-specific APIs without requiring the change in the MPI level designs. There are mainly two approaches used to interface with AMD GPUs; 1) low-level HSA APIs, and 2) high-level HIP APIs. Due to the similarities and one-to-one mappings between CUDA APIs and HIP APIs, we used HIP in our UDI layer to interface with AMD GPUs. Figure 2 shows the high-level architecture of our GPU-aware MPI runtime with UDI abstraction layer. We move all the protocol level advanced designs in the UDI layer. For instance, one of the major designs employed by our MPI library for peer-to-peer IPC transfers is to amortize the overheads of registering handles by caching the registered handles for subsequent communications. By moving this design to UDI, we avoid redundancy and the same designs are used for both CUDA IPC as well as ROCm IPC. As later shown in Fig. 4 these designs lead to significant performance improvement for both CUDA-aware and ROCm-aware MPI communication.

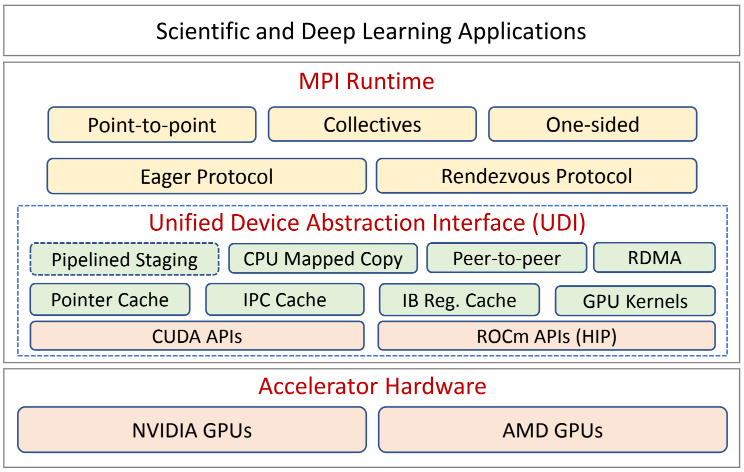
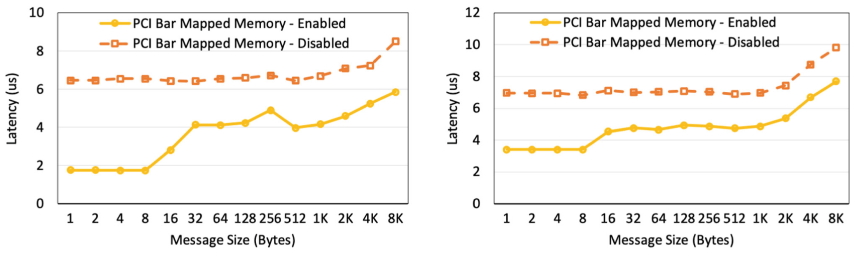


**Fig. 2.** A high-level overview of the proposed designs in accelerator-aware MPI. We propose a Unified Device abstraction Interface (UDI) layer in MPI that abstracts the common operations in a GPU-aware MPI runtime. The modular design makes it easy to interface with vendor-specific backend implementations such as CUDA or ROCm (HIP) APIs.

## 3.3 PeerDirect

Network adapters can directly access device-resident data through ROCm-RDMA, enabling direct memory access for 3rd party PCIe devices. For AMD

GPUs, ROCmRDMA, or PeerDirect support, is available through the ROCm driver. AMD GPUs large-bar configuration allows for the entire GPU memory to be exposed, enabling peer-to-peer DMA access. The ROCm-aware PeerDirect approach in the proposed ROCm-aware MPI library works with the rendezvous protocol for data transfer. The source process and the remote process exchange the addresses of their buffers. An RDMA operation is then used to issue the data transfer. A registration cache is used here to keep a record of the reused buffers to reduce the overhead of repeated registration of GPU memory.



(a) Intra-Node Latency (b) Inter-Node Latency

**Fig. 3.** Utilizing PCI bar mapped memory for small message inter-node and intra-node communication

## 3.4 CPU-Driven GPU Mapped Memory Copy Based Design

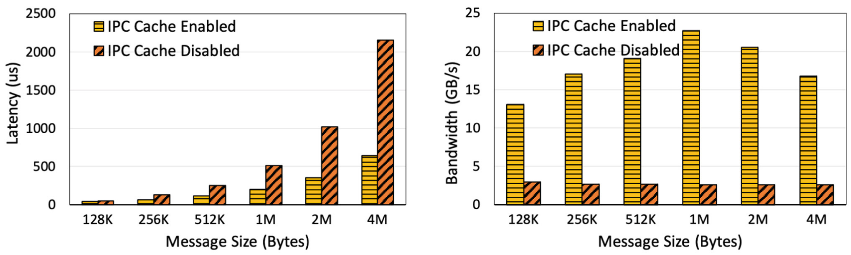
In order to enhance small message performance on NVIDIA GPUs, GDRCOPY, a low-overhead CPU driven copy that allows for the CPU to map the GPU memory, is used. It utilizes a specific API and the GDRCopy kernel module to pin the device buffer using PCI BAR1 memory. The host CPU treats this mapped memory just like any other host memory and derives the communication. On AMD GPUs, no such kernel module is required, and instead, it offers support for Large BAR (Base Address Register) feature that maps entire GPU memory to host address space. We exploit large Bar features to provide similar small message performance enhancements. In Fig. 3, we see the impact of utilizing the mapped copy through the Large BAR feature of AMD GPUs by evaluating the performance difference when it is enabled compared to when it is disabled. In the small message range where these designs would have the most impact between the range of 1B to 8 KB, we see up to 3× lower latency in utilizing the PCI Bar Mapped Memory copy for intra-node point-to-point communication and up to 2× better performance for inter-node communication. We evaluate the impact of the added PCI Bar Mapped Memory copy on intra-node and inter-node point-to-point performance in Fig. 5(a) and Fig. 6(a), respectively.

## 3.5 ROCm IPC Based Design

The simplest approach to designing a rendezvous based transfer for large message sizes between GPUs on a node would be to implement a staging based design where the data transfer involves staging to the host memory. The source would copy data from the device to the shared host memory region between the two processes, and the destination would then copy from the host to its device memory. This would incur an added cost for large message sizes where the performance would be impacted by the overhead of these additional copies. Inter-Process Communication (IPC) provides a peer-to-peer mechanism that allows for direct MPI calls over device memory, facilitating a copy between processes on different GPUs within a node, while entirely bypassing the host memory. However, peer-to-peer support is only available when two devices share the same PCIe switch in the system topology (e.g., devices are the same socket). Earlier work presenting the benefits of utilizing IPC on NVIDIA GPUs [17] shows enhanced performance in allowing direct MPI calls over device memory for large message sizes. We apply IPC-based data transfer mechanisms for ROCm-aware communication through enabling direct access to the AMD GPU memory between processes on the same node and sharing the same PCI root complex. The design is detailed as follows: A process will use $deviceIPCMemHandle$ (abstract call in UDI) to generate an IPC handle on its device buffer and send this handle to the remote process. This will expose its device buffer, allowing for it to be mapped by the remote process into its own address space and then directly issue a $deviceMemCpy$ call on the addressable buffer. In utilizing ROCm IPC with the rendezvous protocol, this exchange happens during the handshake between the source process and the remote process. When the source is sending a Request to Send (RTS) message, it will also exchange the IPC handle generated. The remote process will map this handle and directly copy from the device memory of the source.

Historically, IPC usage has shown overhead due to generating and exchanging IPC handles repeatedly. This added cost makes the performance gain, that would be obtained through bypassing host memory, negligible. In order to eliminate this added overhead and to demonstrate benefit from the IPC designs, we utilize an IPC Cache in the proposed ROCm-aware MPI. This implements caching of IPC handles at the source and destination, allowing for direct data movement whenever a cache hit is encountered on the handle. This IPC cache exists in the UDI layer where it is utilized for the ROCm run-time in order to have the handles cached for subsequent communication. Figure 4(a) demonstrates the difference in latency for large message intra-node point-to-point communication between 128 KB and 4 MB with IPC cache enabled compared to IPC cache disabled. We see 2–3$\times$ lower latency when IPC cache is enabled and approximately 10$\times$ higher bandwidth (Fig. 4(b)).

We integrate the ROCm-IPC design into the proposed ROCm-aware MPI library to improve intra-node performance for data transfer between GPUs on the same node. We evaluate designs for large message ranges to determine the appropriate range of use on AMD GPUs. We present intra-node latency, bandwidth, and bi-directional bandwidth with ROCm-IPC being used for message sizes $> 8$ KB in Fig. 5.



(a) Intra-Node Latency (b) Intra-Node Bandwidth

**Fig. 4.** Utilizing IPC cache for large message intra-node communication

## 4 Performance Evaluation

In this section, we detail the hardware and software configurations of the compute platform used for the evaluation. We also present a detailed evaluation of the proposed ROCm-aware MPI implementation against Open MPI 4.1.0 + UCX 1.10.0 (details of the configuration provided in Table 2). We report a comparison of latency, bandwidth, and bi-directional bandwidth for MPI point-to-point operations and MPI collective operations. We then delve into the application level benchmarks by evaluating hipified versions of 3D Stencil, and HPCG (rocHPCG) with ROCm support.

### 4.1 Experimental Setup

The evaluation was conducted on the **Corona Cluster**, deployed at Lawrence Livermore National Laboratory [2]. It consists of 291 AMD EPYC 7002 series CPU nodes: 82 nodes are equipped with 4 MI50 AMD GPUs per node, 82 nodes have 4 MI60 AMD GPUs per node, and 123 nodes consisting of 8 MI50 AMD GPUs per node. The MI50 AMD GPUs have 32 GB HBM, with single-precision peak theoretical floating-point performance of up 13.3 teraFLOPS.

Each node is equipped with dual-socket Mellanox IB HDR-200, with AMD EPYC 7402 24-Core Processor running Mellanox OFED 5.0, and ROCm version 4.1.0. In our evaluation, we utilized the nodes with 8 MI50 GPUs per node to evaluate the performance of dense GPU nodes.

**Peak Achievable Performance of Interconnects**–To evaluate the performance of the proposed ROCm-aware MPI and OpenMPI + UCX compared to the peak achievable performance, we utilized the following tests:

- rocm_bandwidth_test: We utilized this test to evaluate the performance between two GPUs on a node (displays the peak achievable bandwidth by performing a uni/bi-directional copy involving the two devices [1]).
- Infiniband Perftest: We utilized the *ib_read_bw* and *ib_read_lat* provided by the Infiniband Perftest package to measure the peak achievable bandwidth and minimum achievable latency of communicating data across two nodes [4].

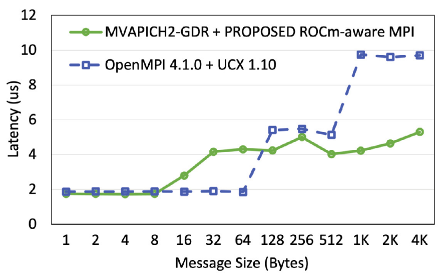
**Table 2.** Experimental setup of OpenMPI 4.1.0 and UCX 1.10.0

| | |
|---|---|
| Configure UCX | –with-rocm=<path-to-rocm>–without-knem–without-cuda–enable-optimizations |
| Configure OpenMPI | –with-ucx=<path-to-ucx>–without-verbs |
| Run-time parameters | -mca btl "ˆopenib" -mca pml ucx<br>**ROCm UCX Parameters:** rocm, rocm_copy, rocm_ipc |

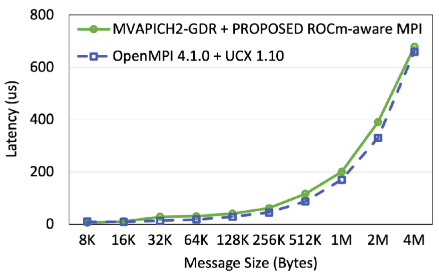
## 4.2 Micro-Benchmark Evaluation

In order to develop a comprehensive evaluation of various point-to-point and collective MPI operations, we utilized the OSU Micro-Benchmarks (OMB) suite version 5.7 that has support for AMD GPUs via the HIP interface. These micro-benchmarks are used in evaluating MPI operations across different MPI libraries on the CPU and support for CUDA-aware operations on the GPU for point-to-point, one-sided, and collective communication. The metrics reported represent measures of latency, bandwidth, or bi-directional bandwidth. We utilize OMB with added support for ROCm-aware MPI operations (through HIP) to evaluate our proposed ROCm-aware MPI implementation against Open MPI + UCX.

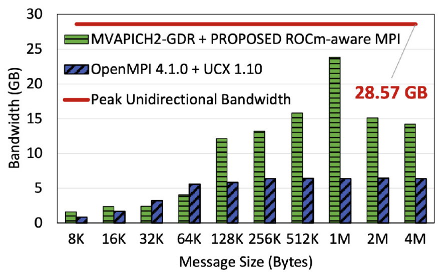
**Intra-Node Point-to-Point**—We evaluate the most common configuration for binding MPI processes to GPUs for an MPI+GPU run where one MPI process utilizes a single GPU. We evaluate intra-node point-to-point communication with two processes bound to two GPUs on the same node. Figure 5 depicts the results of evaluating this on MI50 GPUs on the Corona system for latency, bandwidth, and bi-directional bandwidth performance. Two GPUs on the same socket within the node (i.e. GPU 0 and GPU 1) share the same PCIe switch and can have peer-to-peer access enabled. The proposed ROCm-aware MPI demonstrates as low as 1.74 µs latency (Fig. 4(a)) for 8 Bytes. For small message intra-node communication, we utilize the PCI Bar Mapped Memory approach proposed in Sect. 3.4 in order to obtain the latency presented. As demonstrated in Fig. 3(a), within the range of 1 B to 8 KB we see between 16–66% benefit based on message size by enabling PCI Bar Mapped Memory for this range as opposed to disabling it. This PCI Bar Mapped Memory approach improves the latency of the proposed ROCm-aware MPI from ∼6.54 µs to ∼1.80 µs within the range of 1 B to 16 Bytes. Within the range of 128 Bytes to 8 KB, we see a vast performance difference between the proposed ROCm-aware MPI and Open MPI + UCX. This gap in performance between the two MPI libraries can be attributed to the use of Loopback designs in conjunction with the PCI Bar Mapped Memory in the proposed designs. The loopback design utilizes ROCm RDMA and the PCI Bar Mapped Memory to avoid expensive copy operations by relying on IB verbs to initiate the transfer between the host and device [20]. In Fig. 5(c), we see that the bandwidth of the proposed ROCm-aware MPI is about 3× higher than that of Open MPI + UCX and between 3–6× higher for bi-directional bandwidth. In this range, the proposed ROCm-aware MPI utilizes the ROCm-IPC cache proposed in Sect. 3.5 to deliver 23.8 GB/s bandwidth,



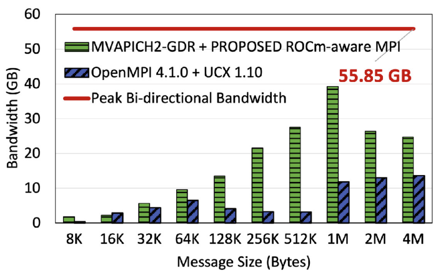
(a) Latency (Small Messages)



(b) Latency (Large Messages)



(c) Bandwidth



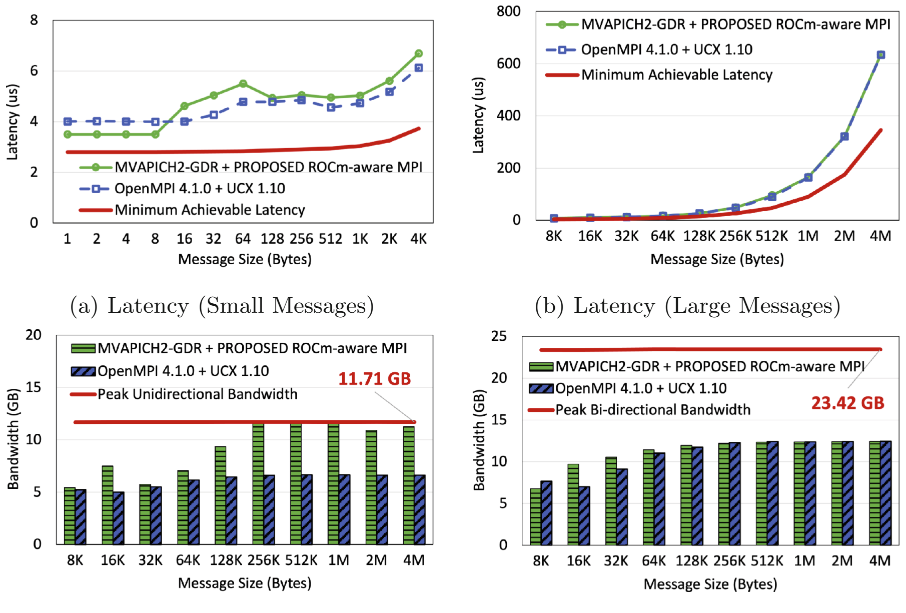
(d) Bi-Directional Bandwidth

**Fig. 5.** Comparison of intra-node MPI point-to-point operations between proposed ROCm-aware MPI Library and Open MPI + UCX on the Corona system

and 39.2 GB/s bi-directional bandwidth at 1 MB message transfer. As depicted in Fig. 5(b), enabling the IPC cache designs improves the bandwidth by over 4X. We see similar trends as Fig. 4(b) in this comparison between the proposed ROCm-aware MPI library and OpenMPI + UCX, with the proposed designs performing about 3× higher than Open MPI + UCX.

**Inter-Node Point-to-Point**—Device-resident data is typically sent over the network in order to achieve higher scalability and enhanced performance for HPC applications. We evaluated the performance of MPI communication of GPU resident data across the InfiniBand network (IB HDR 200 Gbps) by using point-to-point latency, bandwidth, and bi-directional bandwidth benchmarks. Figure 6 shows the result of inter-node device-to-device communication between two MPI processes each bound to a GPU on different nodes.

We see similar trends in latency between the proposed ROCm-aware MPI and OpenMPI + UCX, achieving 3.5 µs and 4.01 µs minimum latency, respectively. This is in comparison to the minimum achievable latency of 2.8 µs for this configuration of communication. The proposed ROCm-aware MPI utilizes the PCI Bar Mapped memory for small message size communication demonstrated in Fig. 3(b) to achieve low latency between the range of 1 B to 8 KB. In terms of the figbandwidth evaluation, shown in Fig. 6(c), the peak achievable bandwidth is 11.71 GB/s. The proposed ROCm-aware MPI is able to achieve close to peak performance with 11.57 GB/s bandwidth compared to OpenMPI + UCX at 6.67



(a) Latency (Small Messages) (b) Latency (Large Messages) (c) Bandwidth (d) Bi-Directional Bandwidth

**Fig. 6.** Comparison of inter-node MPI point-to-point operations between proposed ROCm-aware MPI Library and Open MPI + UCX on the Corona system

GB/s bandwidth. This communication across the nodes is critical for scalability and important to analyze in order to understand how well the bandwidth provided by the Infiniband networks is saturated by the MPI libraries.

**MPI Collective Operations**—We evaluate the performance of the proposed ROCm-aware MPI and Open MPI + UCX on 128 GPUs (16 nodes, 8 GPUs per node) on the Corona system for MPI Collective operations including broadcast, reduce, gather, allgather, alltoall, and allreduce using ROCm-aware OMB. In Fig. 7, we evaluate small message collective operations ranging from 4 B to 4 KB. For broadcast operations, the proposed ROCm-aware MPI shows 9.03 µs compared to 18.04 µs for Open MPI + UCX. In Fig. 7(b), we see 3.13 µs compared to 4.34 µs in the lower range at 4 B for reduce operations. We see 2.07 µs compared to 4.81 us for our proposed ROCm-aware MPI and Open MPI + UCX, respectively for gather operations in Fig. 7(c). In Figs. 7(d), 7(e), and 7(f), we see a larger difference between the proposed ROCm-aware MPI and Open MPI + UCX for dense collectives, with the former having 2-5X lower latency in this message range. In Fig. 8, we evaluate large message collective operations with the message size ranging from 8 KB to 1 MB. Due to node failures with scaling Open MPI + UCX to multiple nodes with 8 processes per node where the run crashes after outputting results for 512 Bytes, Fig. 7(d) and Fig. 8(d) are missing values for the Allgather comparison.

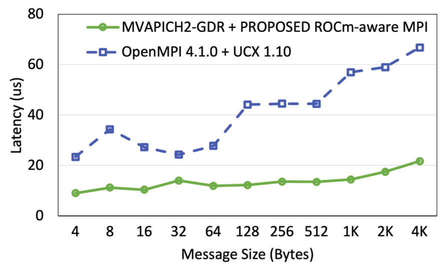
The performance gain demonstrated in collective operations is reliant on optimized point-to-point primitives detailed above when evaluating point-to-point benchmarks and optimized collective algorithms within the MPI library. The protocol level advanced designs in the UDI layer are utilized for point-to-point and collective operations. In addition to the optimized protocol level designs in the UDI layer, the library has been tuned on the system in order to adaptively select optimized GPU-based collective algorithms for different message ranges. The various collectives evaluated have been optimized for GPU-based communication to account for and utilize interconnects between GPUs, node density (number of GPUs within a node), and scalability to yield enhanced performance.

## 4.3 Application-Level Evaluation

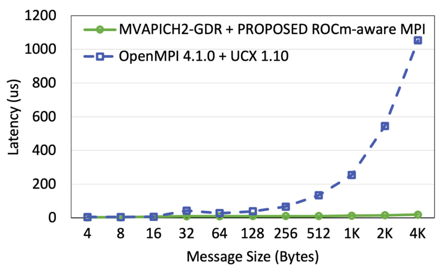
In this section, we evaluate the performance of the proposed ROCm-aware MPI implementation against Open MPI + UCX using various application kernels including ROCm-aware HPCG and 3D Stencil benchmark. Several applications are in the early stages of adding support for ROCm to run on AMD GPUs with experimental versions released to the public.

**3D Stencil**—3D Stencil is a communication kernel that mimics the communication pattern of stencils and halo-exchanges in scientific applications. The kernel creates a 3D cartesian grid of MPI processes and runs the benchmark for $n$ iterations. In each iteration, a given MPI rank performs a 7-point stencil and communicates $k$ messages with each of its peers. We demonstrate the latency of 3D stencil for 16 (Fig. 9(a)), 32 (Fig. 9(b)), and 64 (Fig. 9(c)) GPUs. We encountered runtime failures when running the 3D Stencil kernel with Open MPI. We found that Open MPI failed during `MPI_Cart_create` due to an implementation bug. Due to this failure, we are not able to present a comparison with Open MPI for this application.

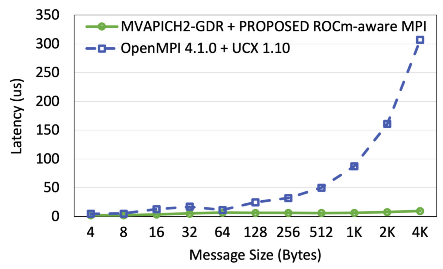
**RocHPCG**—High-Performance Conjugate Gradients (HPCG) benchmark is proposed to complement LINPACK (HPL) and used to rank modern HPC systems [10]. The computational and data-access patterns employed by the benchmark are representative of a variety of scientific codes. The numerical methods contain different communication patterns involving MPI point-to-point and collective operations. The rocHPCG is a ROCm-aware port of the HPCG benchmark intended for AMD GPUs. In Fig. 10(a) we evaluate the proposed ROCm-aware MPI against Open MPI + UCX on 16 GPUs across 16 nodes on the Corona cluster. We demonstrate the performance of each of the phases in rocHPCG: DDOT (dot products), WAXPBY (vector update phase), SpMV (sparse matrix-vector multiplication), and MG (multi-grid). Likewise, we demonstrate the performance of the proposed ROCm-aware MPI on 32 GPUs (Fig. 10(b) and 64 GPUs (Fig. 10(c)). We used per-process grid dimensions of `(nx, ny, nz) = (104, 104, 104)`. On 16 GPUs, we see a final 633.3 GFLOPs/s for all the phases combined with our proposed ROCm-aware MPI compared to 585.9 GFLOPS/s for Open MPI + UCX. On 32 GPUs, we see a vaster difference with 1056.9 GFLOPs/s compared to 565.5 GFLOPs, and on 64 GPUs we demonstrate 1673.4



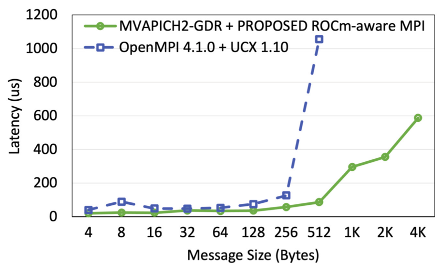
(a) BROADCAST



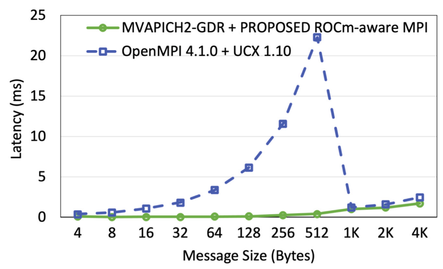
(b) REDUCE



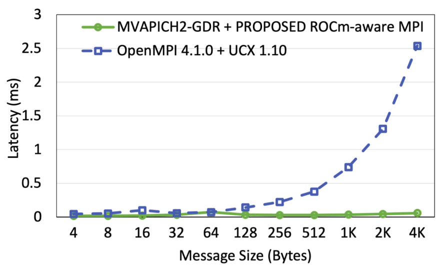
(c) GATHER



(d) ALLGATHER



(e) ALLTOALL



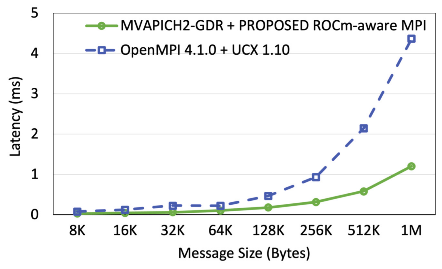
(f) ALLREDUCE

**Fig. 7.** Comparison of small message MPI collective operations between proposed ROCm-aware MPI Library and Open MPI + UCX on 128 GPUs (16 nodes, 8 GPUs per node) on the Corona system

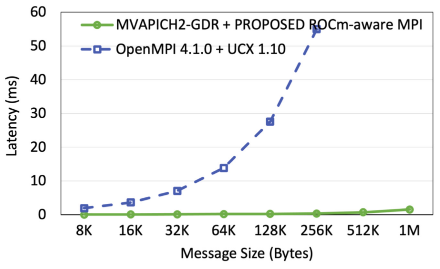
GFLOPs/s compared to 740.4 GFLOPs/s with our proposed ROCm-aware MPI and Open MPI + UCX, respectively.

## 5 Related Work

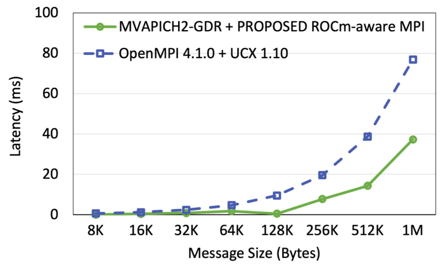
Over the last few years, GPU devices have been widely used on modern clusters to provide higher computing power. Hence, communication between GPUs has become a critical bottleneck. Wang et al. [23] proposed early research using standard MPI libraries to transfer data between GPUs in InfiniBand clusters. The communication excluded the involvement of the CPU, so it prevented the CPU/GPU buffer management and data movement issues. Potluri et



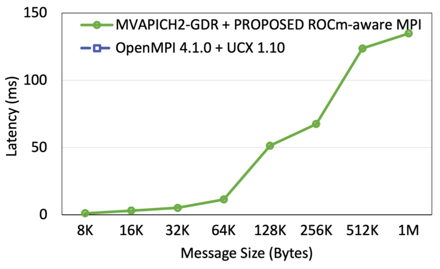
(a) BROADCAST



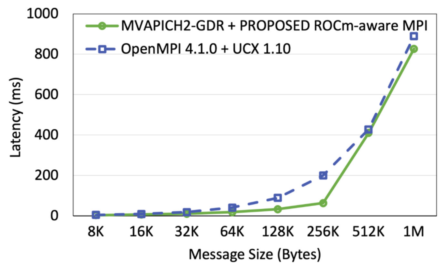
(b) REDUCE



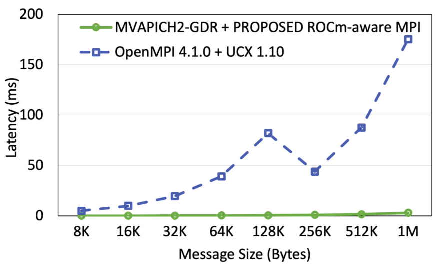
(c) GATHER



(d) ALLGATHER



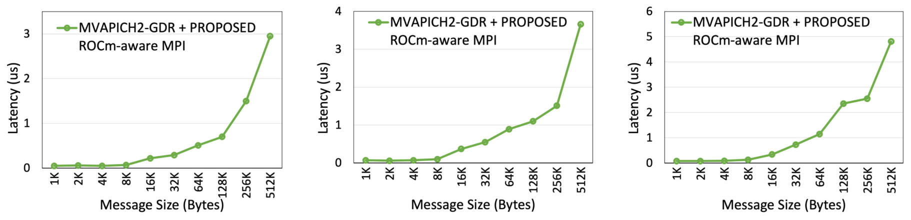
(e) ALLTOALL



(f) ALLREDUCE

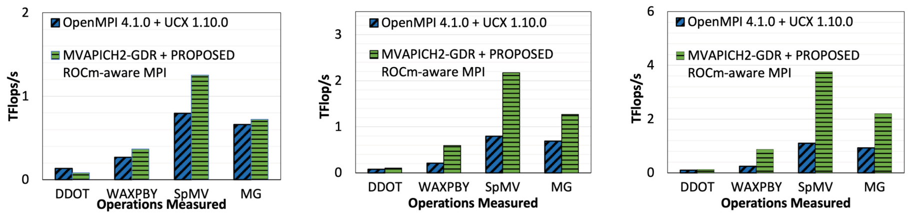
**Fig. 8.** Comparison of large message MPI Collective operations between proposed ROCm-aware MPI Library and Open MPI + UCX on 128 GPUs (16 nodes, 8 GPUs per node) on the Corona system

al. [18] aimed to deal with inter-node GPU-to-GPU MPI communication using GPUDirect RDMA. They studied the limitations of the system architectures and proposed a hybrid solution from the existing host-based pipeline and new GPUDirect-based designs. Based on the previous work on GPU-Aware MPI using GPUDirect RDMA, Shi et al. [20] further optimized the communications for small message sizes between inter-node GPUs. It supported using the eager protocol at not only sender but receiver sides as well. A new data path design was also proposed that allowed low-latency data movements between host and remote GPU memories. Recent works [9,12] identified and addressed the limitations in efficient processing on MPI derived datatypes for GPU resident data. They demonstrated significant performance improvements through novel CUDA



(a) 16 GPUs (b) 32 GPUs (c) 64 GPUs

**Fig. 9.** Evaluation of 3D Stencil Code with the proposed ROCm-aware MPI Library on the Corona system



(a) 16 GPUs (b) 32 GPUs (c) 64 GPUs

**Fig. 10.** Comparison of rocHPCG between proposed ROCm-aware MPI Library and Open MPI + UCX on the Corona system [Per-process grid size (nx, ny, nz) = (104, 104, 104)]

kernel-based packing/unpacking and kernel fusion designs for non-contiguous data transfer. Subramoni et al. [21] addressed the trade-off between communication protocols in point-to-point data transfer. They proposed designs to identify the communication characteristics of processes at runtime and dynamically adapt to them. The fully in-band design allowed the transition from one eager-threshold to another without sacrificing the throughput.

Sharkawi et al. [19] discussed the techniques used in Spectrum-MPI on modern clusters equipped with IBM POWER9 CPUs. Kawthar et al. [13] evaluated the performance of existing CUDA-aware MPI libraries on OpenPOWER GPU-enabled systems by comparing benchmark-level point-to-point performance of Spectrum MPI, Open MPI+UCX, and MVAPICH2-GDR. Much of the work done using GPU-resident data transfer and communication has been heavily focused on NVIDIA GPUs due to the heavy deployment of NVIDIA GPUs across platforms. In the context of AMD GPUs, Kuznetsov et al. [14] investigated the ROCm platform and evaluated whether it is comparable to CUDA. They also focused on the programmers' experience of porting classical molecular dynamics algorithms from CUDA to ROCm and the performance benchmarking with these modern architectures. In addition, Tsai et al. [22] discussed the experience of porting the functionality in a CUDA-focused library to the HIP ecosystem. They demonstrated the porting workflow of linear algebra kernels and some techniques from CUDA to HIP in detail. Cai et al. [8] introduced the Synthesized Collective Communication

Library (SCCL), which is a latency-optimal and bandwidth-optimal implementation of collective communication algorithms.

## 6 Conclusion

As next-generation HPC systems such as Frontier and El Capitan adopt AMD GPUs, it is important to ensure that scientific applications and the communication middleware such as MPI are supported and enhanced for these systems. In order to add support for AMD GPUs, ROCm is used as the high-performance software development platform on AMD GPUs. Over the years, the state-of-the-art GPU-aware MPI libraries evolved with enhanced support for device resident data transfer. These implementations heavily rely on the CUDA toolkit to exploit NVIDIA GPUs. With the recent trend of AMD GPU usage, it is pertinent to have a ROCm-aware MPI library with support and optimizations for AMD GPU-resident data transfer. In this work, we took up the challenge of designing a ROCm-aware MPI runtime through designing an abstract communication layer that interfaces with the CUDA and the ROCm runtimes. We utilized the various features available through ROCm such as PeerDirect, ROCm IPC, and large-BAR mapped memory to generate GPU-based communication for AMD GPUs. We evaluated the performance of MPI-level point-to-point and collective operations with our proposed ROCm-aware MPI Library built over MVAPICH2-GDR and Open MPI with UCX as the ROCm-aware communication backend on the Corona cluster. We demonstrated 3–6$\times$ higher bandwidth for intra-node communication and 2$\times$ higher bandwidth for inter-node communication, respectively. With the rocHPCG application, we demonstrate approximately 2.2$\times$ higher GFLOPs/s with MVAPICH2-GDR + our proposed ROCM-aware MPI compared to OpenMPI with UCX. To the best of our knowledge, this is the first research work that studies the tradeoffs involved in designing a ROCm-aware MPI library for AMD GPUs.

## References

1. Bandwidth test for ROCm. https://github.com/RadeonOpenCompute/
2. Corona. https://hpc.llnl.gov/hardware/platforms/corona
3. Frontier: ORNL's exascale supercomputer designed to deliver world-leading performance in 2021. https://www.olcf.ornl.gov/frontier/. Accessed 25 May 2021
4. Infiniband Verbs Performance Tests. https://github.com/linux-rdma/perftest
5. Radeon Open Compute (ROCm) Platform. https://rocmdocs.amd.com
6. RLLNL and HPE to partner with AMD on El Capitan, projected as world's fastest supercomputer. https://www.llnl.gov/news/llnl-and-hpe-partner-amd-el-capitan-projected-worlds-fastest-supercomputer. Accessed 25 May 2021
7. Unified Communication X. http://www.openucx.org/. Accessed 25 May 2021
8. Cai, Z., et al.: Synthesizing optimal collective algorithms (2020)
9. Chu, C.H., Khorassani, K.S., Zhou, Q., Subramoni, H., Panda, D.K.: Dynamic kernel fusion for bulk non-contiguous data transfer on gpu clusters. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER), pp. 130–141 (2020). https://doi.org/10.1109/CLUSTER49012.2020.00023

10. Dongarra, J., Heroux, M.A., Luszczek, P.: High-performance conjugate-gradient benchmark: a new metric for ranking high-performance computing systems. Int. J. High Perform. Comput. Appl. **30**(1), 3–10 (2016)
11. Gabriel, E., et al.: Open MPI: goals, concept, and design of a next generation MPI implementation. In: Proceedings. 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, pp. 97–104, September 2004
12. Hashmi, J.M., Chu, C.H., Chakraborty, S., Bayatpour, M., Subramoni, H., Panda, D.K.: FALCON-X: zero-copy MPI derived datatype processing on modern CPU and GPU architectures. J. Parallel Distrib. Comput. **144**, 1–13 (2020). https://doi.org/10.1016/j.jpdc.2020.05.008. http://www.sciencedirect.com/science/article/pii/S0743731520302872
13. Khorassani, K.S., Chu, C.H., Subramoni, H., Panda, D.K.: Performance evaluation of MPI libraries on GPU-enabled OpenPOWER architectures: early experiences. In: International Workshop on OpenPOWER for HPC (IWOPH 19) at the 2019 ISC High Performance Conference (2018)
14. Kuznetsov, E., Stegailov, V.: Porting CUDA-based molecular dynamics algorithms to AMD ROCm platform using HIP framework: performance analysis. In: Voevodin, V., Sobolev, S. (eds.) RuSCDays 2019. CCIS, vol. 1129, pp. 121–130. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36592-9_11
15. Leiserson, C.E., et al.: There's plenty of room at the top: what will drive computer performance after Moore's law? Science 368(6495) (2020). https://doi.org/10.1126/science.aam9744. https://science.sciencemag.org/content/368/6495/eaam9744
16. Panda, D.K., Subramoni, H., Chu, C.H., Bayatpour, M.: The MVAPICH project: transforming research into high-performance MPI library for HPC community. J. Comput. Sci. 101208 (2020). https://doi.org/10.1016/j.jocs.2020.101208. http://www.sciencedirect.com/science/article/pii/S1877750320305093
17. Potluri, S., Wang, H., Bureddy, D., Singh, A.K., Rosales, C., Panda, D.K.: Optimizing MPI communication on multi-GPU systems using CUDA inter-process communication. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum, pp. 1848–1857 (2012). https://doi.org/10.1109/IPDPSW.2012.228
18. Potluri, S., Hamidouche, K., Venkatesh, A., Bureddy, D., Panda, D.K.: Efficient inter-node MPI communication using GPUDirect RDMA for InfiniBand clusters With NVIDIA GPUs. In: 2013 42nd International Conference on Parallel Processing (ICPP), pp. 80–89. IEEE (2013)
19. Sharkawi, S.S., Chochia, G.A.: Communication protocol optimization for enhanced GPU performance. IBM J. Res. Dev. **64**(3/4), 9:1–9:9 (2020)
20. Shi, R., et al.: Designing efficient small message transfer mechanism for inter-node MPI communication on InfiniBand GPU clusters. In: 2014 21st International Conference on High Performance Computing (HiPC), pp. 1–10, December 2014
21. Subramoni, H., Chakraborty, S., Panda, D.K.: Designing dynamic and adaptive MPI point-to-point communication protocols for efficient overlap of computation and communication. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds.) ISC 2017. LNCS, vol. 10266, pp. 334–354. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58667-0_18
22. Tsai, Y.M., Cojean, T., Ribizel, T., Anzt, H.: Preparing ginkgo for AMD GPUS - a testimonial on porting CUDA code to HIP (2020)
23. Wang, H., Potluri, S., Bureddy, D., Rosales, C., Panda, D.K.: GPU-aware MPI on RDMA-enabled clusters: design, implementation and evaluation. IEEE Trans. Parallel Distrib. Syst. **25**(10), 2595–2605 (2014). https://doi.org/10.1109/TPDS.2013.222