Network-Assisted Noncontiguous Transfers for GPU-Aware MPI Libraries

Kaushik Kandadi Suresh, Kawthar Shafie Khorassani, Chen Chun Chen, Bharath Ramesh, Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda, *The Ohio State University, Columbus, OH, 43210-1277, USA*

The importance of graphics processing units (GPUs) in accelerating HPC applications is evident by the fact that a large number of supercomputing clusters are GPU enabled. Many of these HPC applications use message passing interface (MPI) as their programming model. These MPI applications frequently exchange data that is noncontiguous in GPU memory. MPI provides derived datatypes (DDTs) to represent such data. Past research on DDTs mainly focused on optimizing the pack–unpack kernels. Modern HCAs are capable of gathering/scattering data from/to noncontiguous GPU memory regions. We propose a low-overhead HCA-assisted scheme to improve the performance of GPU-based noncontiguous exchanges without the GPU-based pack–unpack kernels. We show that the proposed scheme provides up to 2$\times$ benefits compared to the existing pack-based scheme at the benchmark level. Furthermore, we show up to 17% improvement with the SW4Lite application compared to other MPI libraries, such as MVAPICH2-GDR and OpenMPI+UCX.

Graphics processing units (GPUs) have become ubiquitous in modern supercomputers due to their high compute capability and power efficiency. In these supercomputing clusters, many message passing interface (MPI)-based large-scale GPU-based applications exchange data that are noncontiguous in memory. MPI provides derived datatypes (DDTs) that allow an application to represent any noncontiguous layout in memory. Table 1 gives a summary of access patterns and possible datatypes of different HPC applications. This underscores the importance of optimizing such noncontiguous transfers.

MPI libraries typically use pack–unpack kernels for internode DDT-based exchanges. All the preceding studies on DDT-based internode optimizations have either optimized 1) pack-unpack kernels[3] or 2) overlapped kernels with transfers/other kernels for internode GPU-to-GPU transfers.[10] While pack/unpack kernels can be an effective approach to transfer DDT messages, they involve the additional steps of packing and unpacking every buffer on the sender and receiver sides, respectively. Modern host channel adapters (HCAs) provide scatter–gather feature, which allows HCAs to directly gather noncontiguous data from source buffers and scatter noncontiguous data to the destination buffers *without pack–unpack* kernels. None of the past research on GPU DDT optimization explored this possibility of doing noncontiguous transfers using HCA-assisted scatter–gather mechanisms. There are overheads associated with such transfers, such as the cost of registering the layouts with HCAs. In this work, we identify all the challenges in using the HCA-assisted mechanisms for moving GPU resident noncontiguous data and design a low-overhead HCA-assisted data-transfer scheme that performs better than pack-based schemes for certain application layouts.

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3241133
Date of publication 31 January 2023; date of current version 13 March 2023.

BACKGROUND

Memory Registration

InfiniBand (IB) requires memory regions to be registered with the HCA before they can be used for data transfers. Every memory registration with the HCA using the ibv_reg_mr function returns an lkey and rkey. For any remote direct memory access (RDMA) operation (READ or WRITE) posted local on a buffer requires an "lkey" returned from the buffer registration. For a process to perform any

TABLE 1. Summary of datatypes used in HPC applications.

Applications	MPI DDTs used	Data exchange pattern
NAS	MPI_Type_Vector	2D, 3D face exchange
MILC	MPI_Type_Vector, MPI_Type_Contiguous	4D face exchange
Specfem3D	MPI_Type_Vector, MPI_Type_Indexed	Unstructured exchange

RDMA operation on a remote buffer, an "rkey" of the remote buffer is needed. This is why MPI processes typically exchange rkeys of buffers before performing actual data transfer.

IN THIS WORK, WE IDENTIFY ALL THE CHALLENGES IN USING THE HCA-ASSISTED MECHANISMS FOR MOVING GPU RESIDENT NONCONTIGUOUS DATA AND DESIGN A LOW-OVERHEAD HCA-ASSISTED DATA-TRANSFER SCHEME THAT PERFORMS BETTER THAN PACK-BASED SCHEMES FOR CERTAIN APPLICATION LAYOUTS.

MOTIVATION

Eliminating Overheads Associated With Pack–Unpack Kernels

Prior works[3,10] have shown that pack unpack can consume 20%–40% of time in DDT-based exchanges. From our profiling, we found that the costs could be as high as 40% of the total transfer time. We have omitted this graph here considering the space. Given this information, we strive to know if we can leverage the HCA's scatter/gather mechanism to exchange non-contiguous data between MPI ranks.

Amortizing the Mkey Mapping Overhead

State-of-the-art NVIDIA HCAs support noncontiguous RDMA operations through the user mode memory registration (UMR) feature. This feature allows a program to directly exchange noncontiguous data from a set of source buffers to a set of destination buffers using a single post operation. This feature allows the source

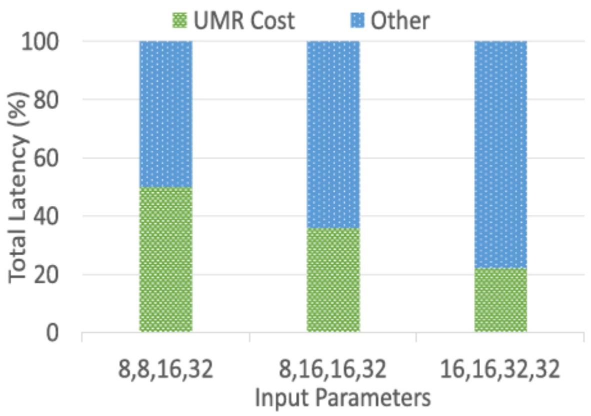

FIGURE 1. UMR overhead as a percentage of total latency in a ping-pong benchmark, which exchanges noncontiguous layouts used in the MILC application between source and destination GPU buffers. The UMR overhead refers to the cost of mapping mkeys to a particular layout. The input parameters represent the grid dimensions used by the MILC application.

HCA to gather the noncontiguous data blocks and transfer them to the destination HCA. Then, the destination HCA scatters the data to the destination memory addresses. However, this operation requires the user to create a mkey and map the set of noncontiguous buffers with the mkey and subsequently use that mkey for posting send operations. To understand the cost of these operations we wrote an IB level benchmark that exchanges noncontiguous layouts used in the MIMD lattice computation (MILC) application. We used UMR to exchange the noncontiguous data. We observed that the creation of a single mkey takes about 200 μs, which is approximately twice the latency of transfer for the layouts shown in Figure 1. In Figure 1, we show the time spent in mapping the UMR mkeys to the layout as a percentage of the total time taken to do RDMA-WRITE operation of noncontiguous data. However, applications tend to reuse a particular layout multiple times. Can we leverage this information to amortize the overhead of mapping mkeys to a layout in UMR based transfers?

Amortizing the Mkey Exchange Overhead

By mapping a single mkey to a set of buffers, we can exchange all the buffers using a single ibv_post operation. However, the number of buffers associated with a single mkey is limited. Therefore, one needs to use multiple mkeys depending on the number of blocks in a noncontiguous layout. For a process to do RDMA-WRITE to a remote process, the local process needs the list of remote process' mkeys. To understand the effect of this exchange, let us consider a layout with 4,096 blocks. Assuming the HCA can support four blocks per mkey, this would require 1,024 keys, which can amount to 4 KB of data exchange. This can have a significant impact on the performance of medium message transfers. However, the applications tend to reuse many layouts. This brings us to the next challenge where we ask: Given the temporal repetition of layouts in the application how can we amortize this mkey exchange?

CONTRIBUTIONS

In this work, we motivate the need for a hardware-assisted internode transfer mechanism for GPU resident noncontiguous memory layouts by analyzing the pack costs of application layouts. Driven by this motivation, we identify the challenges with a hardware-assisted mechanism called UMR and propose a design that addresses the abovementioned challenges.

To summarize, this article makes the following contributions.

1) Motivate the need to use HCA-assisted noncontiguous data transfers by profiling the layout of the MILC application.
2) Propose a UMR-based design and optimize it by implementing efficient caching mechanisms.
3) Demonstrate the usefulness of the proposed schemes by comparing the performance of the proposed designs on real application layouts in GPU-based HPC clusters.

DESIGN AND IMPLEMENTATION

In this section, we discuss our network-based noncontiguous transfer design (proposed UMR).

Mkey Mapping

When DDTs are used in MPI calls, such as MPI_Isend, first the sender/receiver parses the DDT handle to get a list of input/output vectors (IOVs). An IOV is a structure that contains the address and length of a contiguous memory block. The number of IOVs in a layout indicates the number of blocks in that layout. It is from this list of addresses the data are sent and received. For more details on DDT processing and IOVs refer to Suresh et al.[7]

Given this list of IOVs obtained after parsing send/receive input DDT handles, first, these memory regions are registered[7] with the HCA, which generates a list of lkeys and rkeys. Then, UMR mkey is created and then mapped to the list of addresses. This mkey object

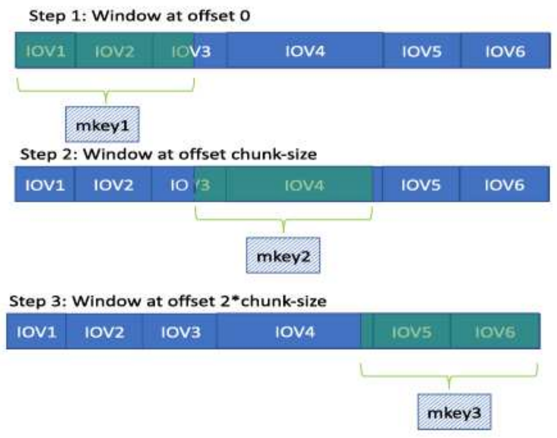
Step 1: Window at offset 0
IOV1
IOV2
IOV3
IOV4
IOV5
IOV6
mkey1
Step 2: Window at offset chunk-size
IOV1
IOV2
IOV3
IOV4
IOV5
IOV6
mkey2
Step 3: Window at offset 2*chunk-size
IOV1
IOV2
IOV3
IOV4
IOV5
IOV6
mkey3

FIGURE 2. Sliding window based approach to map mkeys to IOV list. At each step the IOVs covered by the window is mapped to a new mkey. This way the entire layout is mapped to a set of mkeys, which represent the layout.

contains one lkey and one rkey that refers to the entire memory region. We can use this mkey's lkey to post RDMA operations provided we have the corresponding mkey-based rkey of a destination buffer address. The HCA's DMA engine is responsible for gathering data referred to by the newly created mkey from the local node and scattering data to the remote node according to the mapping in the remote process's mkey.

As described in the "Motivation" section, we can only map a limited number of blocks/IOVs[7] to a single mkey. Therefore, we use a list of mkeys to represent a single layout. To simplify the data exchange process, we first fix a chunk size for a given layout. Then, we use a moving window-based approach to map mkeys with the IOVs. A window of size chunk size starts at IOV-0, and spans all the IOVs whose collective sum of size is the chunk size. The first mkey is mapped to the IOVs spanned by the window. After the first mkey is mapped, the window is moved by an offset of chunk-size. Now that the next mkey is mapped to the IOVs under the current window. This process continues until we exhaust all the IOVs. Note that the window does not span the gaps between blocks, it operates at the IOV level. This process is shown in Figure 2.

When we use the abovementioned approach to map a list of IOVs with a list of mkeys, we also add a constraint of the max number of IOVs per mkey to ensure that our RDMA operation does not fail.

UMR-Based Design

Since the sender and receiver's layouts may not be identical it is necessary to create and map mkeys on

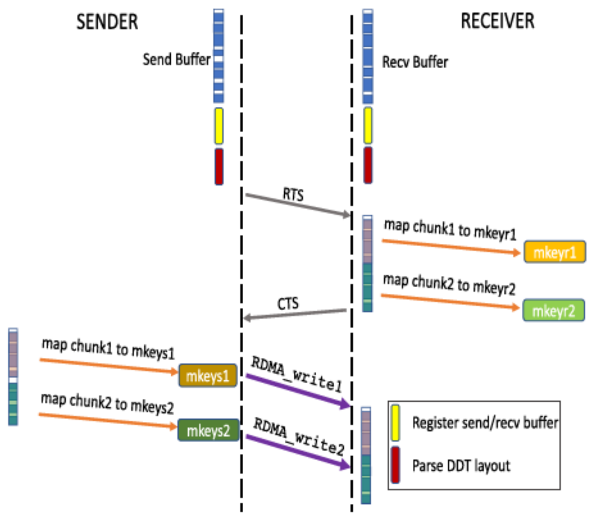

FIGURE 3. Steps involved in the UMR design between a sender and receiver. The figure shows how one send/recv buffer can have multiple mkeys representing regions in memory, and the independent transfer of each region using RDMA operations.

the sender and receiver's side in a co-operative manner to ensure data validation. Now, both sender and receiver will create mkeys based on the agreed chunk-size. This will result in the same number of mkeys on the sender and receiver's sides. A single mkey in the sender side will be used to post an RDMA-WRITE operation using a corresponding remote mkey from the receiver side. This way the responsibility of gathering and scattering data from any type of source layout to any type of destination layout is given to the source and the destination HCAs.

Figure 3 illustrates the various steps involved in our UMR design. First, the sender and receiver register the entire send and receive buffers to obtain the lkey and the rkey. Then the sender sends its chunk size to the receiver. After receiving the sender's layout information, the receiver arrives at an agreed chunk size and it creates and maps mkeys to the receiver's IOVs with this agreed chunk size. These mkeys and agreed chunk size values are sent to the sender in the clear to send (CTS) packet. Then, the sender creates and maps a set of mkeys to its layout based on the agreed chunk size value obtained in the CTS packet. Then, the sender uses the remote mkey's rkey to post RDMA-WRITE operations. The number of RDMA-WRITE operations is equal to the number of mkeys created.

The agreed chunk size is calculated as follows. The sender first sends its minimum block length ($S\text{blockmin}$) in the RTS message. The receiver uses this value, and its own minimum block length ($R\text{blockmin}$) to compute $A\text{blockmin} = \text{MIN}(S\text{blockmin}, R\text{blockmin})$. With this the chunk size is calculated as $c\text{size} = \text{MAX\_IOVS\_PER\_WQE} \times A\text{blockmin}$, where MAX_IOVS_PER_WQE is the maximum allowed IOVs that is supported by the HCA. This ensures that the chunk size satisfies the max IOV criteria for both the sender and the receiver.

Enhancing the UMR Design

As discussed in the "Motivation" section, mkey creation is an expensive operation. Therefore, we create a pool of mkeys at the time of MPI_Init. During the run of the application, a sender/receiver obtains a mkey from the pool and uses it for mapping to IOVs. If at any point the size of the free pool is reduced to 50%, an auxiliary thread is signaled, which creates and adds a fixed number of mkeys to the pool. This is done to ensure that the main thread does not get impacted if it runs out of mkeys. The mkey pool is released in MPI_Finalize at the end of the application. This is because it does not affect the application performance.

We use a layout-cache[2] to amortize the layout parsing cost. In the abovementioned design, the sender and receiver use the same set of layouts multiple times, each time the receiver performs UMR-based registrations and exchanges the list of mkeys with the sender. To amortize the UMR registration cost and the mkey exchange cost, we propose a UMR mkey caching.

We propose the implementation of two mkey caches. The first is a local cache, which is to avoid remapping of mkeys to the layouts. The second is a remote cache maintained at the sender side to cache the remote mkeys. This cache is used by the sender when the receiver's layout was already sent to it at an earlier exchange.

The first cache is implemented as a hash table that is indexed by local layout-cache-id, local base address, and chunk size. Each entry of the cache stores a list of mkeys that are uniquely identified by the abovementioned three parameters. In addition to the mkey list, the local cache stores a bitmap. This bitmap is used by the receiver to store the ranks to which the mkey list was sent. The bitmap size is determined by the total number of ranks involved in the transfer. One bit is reserved for each rank. On the sender's side, this bitmap is not used.

The remote mkey cache is implemented as an array of hash tables where the array is indexed by remote rank. Each entry of the array has a structure similar to the local cache, which is a hash table indexed by the remote layout-cache-id, remote base address and chunk size. The receiver will not send the mkey list in the CTS packet when it determines that the mkey was already sent to a particular. In such cases, when the sender receives CTS packet without the mkey list, the sender uses the remote cache to retrieve the mkey list.

Hybrid-Tuned UMR design

Although the local and remote caches help in amortizing the layout mapping and exchange costs, the HCA has limitations due to which after a certain number of IOVs the performance of the UMR is inferior to a pack-based scheme. This motivates the need for a hybrid scheme where a pack-based scheme is chosen for layouts with large IOVs counts. The optimized UMR-based scheme is chosen for layouts with small block lengths and large IOV sizes. The exact threshold for block lengths and IOV counts are determined by empirical evaluation for each architecture of GPU and HCA. On ThetaGPU systems (refer to the "Performance Evaluation" section) through experimental evaluation we found that the limit for UMR scheme is a layout of 128 IOVs and a block size of 512 bytes.

TABLE 2. Hardware specification of different test-bed clusters.

Specification	MRI	ThetaGPU
Processor family	AMD EPYC	AMD EPYC
Processor model	EPYC 7713	EPYC 7742
Clock speed	2.0 GHz	3.4 GHz
Sockets	2	2
Cores per socket	64	64
NUMA nodes	2	8
CCX per NUMA	8	4
RAM (DDR4)	256 GB	1 TB
Interconnect	IB-HDR(200 G)—1 HCA	IB-HDR(200 G)—8 HCAs
GPU processor	NVIDIA A100×4	NVIDIA A100×8
GPU memory	40 GB	40 GB
Interconnects between GPUs	PCIe	NVLink-3 and NVSwitch
NVIDIA driver version	510.39.01	470.82.01

PERFORMANCE EVALUATION

In this section, we compare the performance of the proposed scheme against other existing pack-based schemes on GPU-based clusters. We also compare the proposed scheme with the state-of-the-art MPI libraries.

Experimental Platforms and Setup

We use MRI and ThetaGPU clusters for our evaluations. The detailed hardware specifications of these clusters are shown in Table 2. MRI is an in-house cluster of eight nodes with AMD-EPYC processors and A100 NVIDIA GPU nodes. The ThetaGPU cluster, deployed at the Argonne Leadership Computing Facility (ALCF), contains 24 DGX-A100 nodes with AMD-EPYC processors. Each node has NVIDIA DGX A100 GPU with 40-GB HBM2 memory. The GPUs are connected with the third generation NVIDIA NVLink and the second generation NVIDIA NVSwitch.

We implemented the proposed scheme (UMR) in MVAPICH2-GDR[6] using MPI DDT and we compare this with other pack-based MPI-DDT schemes in our MPI library. We compare pack-gdr and pack-staged with our scheme. In pack-gdr, the noncontiguous data are packed to a contiguous GPU buffer using CUDA-pack kernels and moved across GPUs using GPU-direct-RDMA. In pack staged, the data are packed to a contiguous GPU buffer and moved to the destination GPU by staging it through host. The state-of-the-art MPI libraries, such as OpenMPI/MVAPICH2-GDR, have been tuned to use their best packing scheme by default. Moreover, libraries, such as OpenMPI, may not give us the option to select a DDT scheme. Therefore, when we compare against the state-of-the-art MPI libraries we use their default configuration.

First, to understand the performance of a simple noncontiguous layouts, we modified the osu_latency test provided by the Ohio State University (OSU) micro-benchmarks suite to support MPI_Type_Vector datatype. In this benchmark, a simple vector layout of a given block length and count is exchanged in a ping-pong manner for a given number of iterations.

Then, we evaluate application kernels with representative application layouts. Finally, to understand application level performance benefit we evaluate SW4Lite proxy application. For all our experiments, we report an average of 100 iterations, excluding the 10 warmup iterations for a total of 110 iterations.

Microbenchmark Results

In this section, we first evaluate the modified osu_latency vector benchmark for block sizes of 1, 2, and 4 KB. For each of these block sizes, we vary the number of segments from 16 to 128. These block lengths are representative of some of the layouts in halo exchange-based applications. Figure 4(a), (b), and (c)

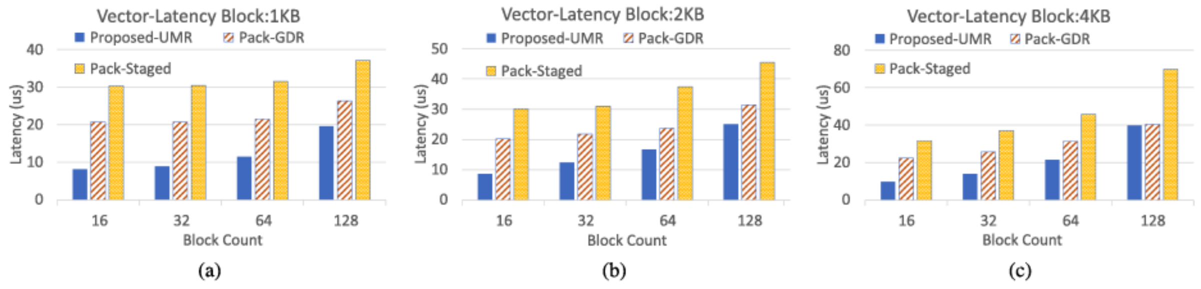

FIGURE 4. Performance comparison of schemes for representative layouts with the basic pack schemes. (a) 1-KB block size. (b) 2-KB block size. (c) 4-KB block size.

shows the results on the MRI cluster. We observe that UMR performs up to 2× better than pack-GDR and up to 3× better than pack staged. The HCA's ability to scatter/gather data directly to GPU buffers coupled with our UMR cache design, which ensures that expensive operations like UMR-registration and mkey exchange happen only once, enable the UMR scheme to outperform pack-based schemes.

FFT Alltoallw Benchmark

In this section, we used an MPI_Alltoallw benchmark with MPI_Type_subarray DDT. The benchmark is derived from an earlier work[4] on FFT that proposed the use of Alltoallw with MPI DDTs to implement multi-dimensional FFTs. Our benchmark measures the time taken to perform the communication operation to transfer the input grid from one dimension to another for a pencil-based FFT applications. We used an input grid size of 128×8×16 on two nodes and performed a weak scaling study where the grid size per node remains the same, as shown in Figure 5(a). This way the layout of DDT used for each run is the same. The layout for this input size contains 64 blocks of 1 KB. Similarly Figure 5(b) and (c) shows the same experiment for different block sizes which in-turn will have corresponding noncontiguous layouts. We observe that the proposed scheme is 2× better than the pack-staged scheme. The proposed scheme is atleast 20% better than the pack-GDR scheme. The benefits are consistent on systems of up to eight nodes.

Comparison of Application Layouts With the State-of-the-Art MPI Libraries

In this section, we evaluate the proposed designs for the performance of various application layouts using applications-level kernels and compare the results with existing state-of-the-art GPU-aware MPI libraries. For our comparisons here, we utilize the MVAPICH2-GDR library (version 2.3.7) and OpenMPI+UCX (OpenMPI version 4.1.4 and UCX version 1.13.0).

We utilize the following application layouts for our evaluation.

MILC: MILC studies the integration of quarks and gluons using quantum chromodynamics. The *MILC_-su3_zd*kernel in DDTbench models the *z*-direction of the *su3_rmd*application from the MILC code. It uses nested vector datatype for 4-D face exchanges. Figure 6(a) shows that the proposed-UMR scheme is at least 15% better than MVAPICH2-GDR and is at least 10× better than OpenMPI+UCX. The layout used

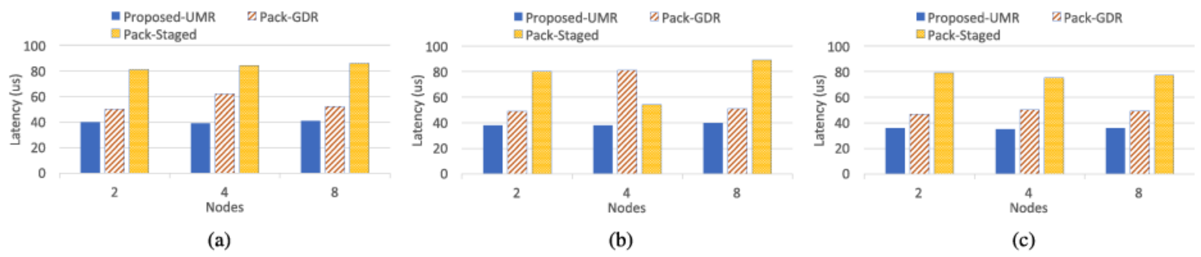

FIGURE 5. Weak scaling experiment of FFT Alltoallw benchmark for different problem sizes on ThetaGPU cluster. (a) Problem size starting at 128×8×6 on two nodes. (b) Problem size starting at 64×16×16 on two nodes. (c) Problem size starting at 64×8×16 on two nodes.

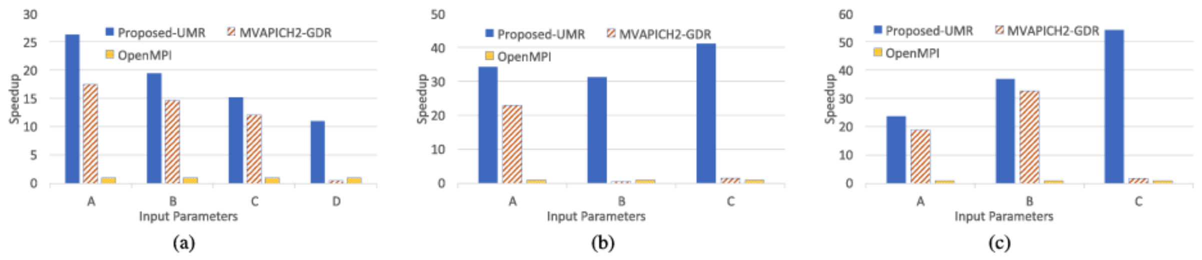

FIGURE 6. Normalized performance comparison of proposed-UMR with state-of-the-art solutions using application kernels for different input sizes. Latencies are normalized with OpenMPI+UCX. Higher is better. (a) MILC on ThetaGPU. Grid dimensions are A = (8,8,16,32), B = (8,16,16,32), C = (16,16,32,32), D = (16,32,32,32). (b) NASMGY on ThetaGPU. Grid dimensions are A = (512,66,66), B = (1024,66,66), C = (2048,66,120). (c) SPECFEM3D mt on ThetaGPU. Grid dimensions are A = (1024,2,32), B = (1024,2,64), C = (1024,2,128).

for the inputs has block lengths varying from 768 to 6,144 bytes. The strides for these layouts are several orders of magnitude larger than the block length. The 15+% benefits with GDR compared to the proposed scheme stems from the usage of optimized UMR-based design with mkey cache.

NAS_MG: NAS_MG is a fluid dynamics application that does 3-D face exchanges in x-, y-, and z-directions with vector and nested vector datatypes. For inputs shown in the graphs, the block lengths go to 6 KB and similar to MILC strides several orders of magnitude more than the block lengths. The counts used for these layouts are around 60 elements. Our proposed scheme is at least 50% better than MVAPICH2-GDR and about 30$\times$ better than OpenMPI+UCX. This again demonstrates the efficiency of HCA-assisted designs that avoid the usage of pack–unpack kernels.

SPECFEM3D_GLOBE: Specfem3d_Globe is a spectral-element application that can simulate global seismic wave propagation through the Earth model. We used the *SPECFEM3D_mt* kernel, which uses vector and contiguous datatypes for data exchange. In Figure 6(c), we compare the performance of the proposed scheme with MVAPICH2-GDR and OpenMPI. The block length used is 4 KB and the counts vary from 32 to 128. Our proposed scheme is nearly 20% better than MVAPCIH2-GDR and performs approximately 20$\times$ better than OpenMPI+UCX.

SW4Lite

SW4Lite is a proxy application, which contains some of the important kernels of the SW4 application. SW4 simulates the propagation of seismic waves in a three-dimensional heterogeneous material model. It uses a fourth order in space and time finite-difference discretization of the elastic wave equations in displacement formulation. SW4Lite exchanges data in two directions (X and Y). It uses nested vector datatype to represent the layouts to be exchanged in each of these directions. To understand the benefits of the proposed UMR design we used the pointsource test of SW4lite with a grid size of 100$\times$100$\times$50 starting on two nodes. We performed a weak scaling experiment by maintaining a constant problem size per process. The results are shown in Figure 7. For our problem size, in one direction the vector layout has a block size of 5 KB and a count of 55. In the other direction, the layout has 3,000 blocks of 48 bytes. In these scenarios with heterogeneous layouts, our hybrid-tuned UMR design will choose the optimized UMR-based scheme for one direction and the best pack-kernel-based scheme on the other direction. As a result the proposed scheme performs up to 17% better in the total application time compared to MVAPICH2-GDR on eight nodes. We also observe more than 2$\times$ improvements compared to OpenMPI+UCX.

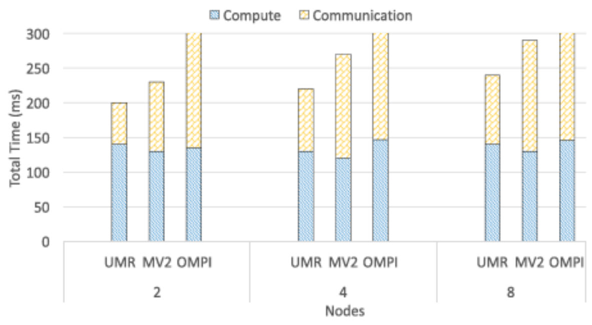

FIGURE 7. SW4Lite weak scaling result. MV2 refers to MVAPICH2-GDR, UMR refers to the proposed-UMR design and OMPI refers to OpenMPI+UCX. We have limited the Y-axis to 300 ms. The communication times for OpenMPI+UCX are 547.23, 1,980.1, and 1,887.2 ms.

RELATED WORK

Past works[5,9] optimized GPU-based DDT exchange by using efficient pack kernels to perform packing–unpacking of noncontiguous data. Chu et al.[1] and Wu et al.[10] proposed additional optimizations, such as chunking and overlapping pack kernels with transfers. However, none of these approaches use HCA hardware accelerated schemes to avoid pack–unpack kernels. Traff et al.[8] proposed "flattening on the fly" scheme to optimize the parsing of MPI DDT layout. This approach is aimed at optimizing the DDT processing cost whereas we optimize the exchange of data.

Suresh et al.[7] explored the aspect of using HCA-assisted mechanisms for GPU-based noncontiguous exchanges. However, their scheme is only beneficial for workloads with homogeneous layouts. Our design benefits applications with heterogeneous layouts by selecting the UMR-based scheme wherever it outperforms other schemes. In addition, our work shows the evaluation of the proposed scheme on the MPI_Alltoallw benchmark and SW4Lite application.

IN THIS WORK, WE PROPOSED AN EFFICIENT MECHANISM TO HANDLE NONCONTIGUOUS DATA ON GPUs BEING COMMUNICATED ACROSS THE NETWORK FOR INTERNODE COMMUNICATION.

CONCLUSION

The deployment of GPUs to accelerate many modern supercomputers has created a need for optimized communication patterns that adhere to the needs of these applications. In particular, applications that are utilizing GPU-aware MPI may require exchanging data that is noncontiguous in GPU memory. While MPI DDTs have been used in the past and extensive work has elaborated on the usage of datatypes for noncontiguous data movement, most of this work focuses on optimizing packing and unpacking schemes. In this work, we proposed an efficient mechanism to handle noncontiguous data on GPUs being communicated across the network for internode communication. Through these designs, we utilize the features provided by modern HCAs in order to gather then scatter data between noncontiguous GPU memory regions. We provide an extensive evaluation of our proposed schemes against existing approaches. At the benchmark layer, we are able to present approximately 2× improvement with our HCA-assisted schemes compared to approaches currently utilized in various state-of-the-art GPU-aware MPI libraries. We also utilize the layouts provided by MILC, NASMG, and Specfem3D to show improvement against various libraries including MVAPICH2-GDR and OpenMPI +UCX. Furthermore, at the application level we show up to 17% improvement with SW4Lite proxy application compared to other MPI libraries.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation (NSF) under Grants 1818253, 1854828, 1931537, 2007991, and 2018627; and in part by XRAC under Grant NCR-130002. The authors would like to thank ALCF for providing resources for this study.

REFERENCES

1. C.-H. Chu, K. Hamidouche, A. Venkatesh, D. S. Banerjee, H. Subramoni, and D. K. Panda, "Exploiting maximal overlap for non-contiguous data movement processing on modern GPU-Enabled systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 983–992.
2. C.-H. Chu, J. M. Hashmi, K. S. Khorassani, H. Subramoni, and D. K. Panda, "High-performance adaptive MPI derived datatype communication for modern Multi-GPU systems," in *Proc. IEEE 26th Int. Conf. High Perform. Comput., Data, Anal.*, 2019, pp. 267–276.
3. C.-H. Chu, K. S. Khorassani, Q. Zhou, H. Subramoni, and D. K. Panda, "Dynamic kernel fusion for bulk non-contiguous data transfer on GPU clusters," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2020, pp. 130–141.
4. L. Dalcin, M. Mortensen, and D. E. Keyes, "Fast parallel multidimensional FFT using advanced MPI," *J. Parallel Distrib. Comput.*, vol. 128, pp. 137–150, 2019.
5. J. Jenkins, J. Dinan, P. Balaji, N. F. Samatova, and R. Thakur, "Enabling fast, noncontiguous GPU data movement in hybrid MPI GPU environments," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2012, pp. 468–476.
6. D. K. Panda, K. Tomko, K. Schulz, and A. Majumdar, "The MVAPICH project: Evolution and sustainability of an open source production quality MPI library for HPC," in *Proc. Int. Workshop Sustain. Softw. Sci.: Pract. Experiences, Held Conjunction Int. Conf. Supercomput.*, 2013.
7. K. K. Suresh et al., "Network assisted non-contiguous transfers for GPU-aware MPI libraries," in *Proc. IEEE Symp. High- Perform. Interconnects*, 2022, pp. 13–20.

8. J. L. Träff, R. Hempel, H. Ritzdorf, and F. Zimmermann, "Flattening on the Fly: Efficient handling of MPI derived datatypes," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, J. Dongarra, E. Luque, and T. Margalef, ed. Berlin, Germany: Springer, 1999, pp. 109–116.
9. H. Wang et al., "Optimized non-contiguous MPI datatype communication for GPU clusters: Design, implementation and evaluation with MVAPICH2," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2011, pp. 308–316.
10. W. Wu, G. Bosilca, R. vandeVaart, S. Jeaugey, and J. Dongarra, "GPU-Aware non-contiguous data movement in open MPI," in *Proc. 25th ACM Int. Symp. High- Perform. Parallel Distrib. Comput.*, 2016, pp. 231–242.

KAUSHIK KANDADI SURESH is a Ph.D. student at The Ohio State University, Columbus, OH, 43210-1277, USA, advised by Dr. D. K. Panda. His research focuses on optimizing CPU and GPU-based communication runtimes, such as MVAPICH2-X and MVAPICH2-GDR. Kandadi Suresh received a bachelor's degree in electrical and electronics engineering from the National Institute of Technology Tiruchirappalli, Tiruchirappalli, India. He is a Graduate Student Member of IEEE. Contact him at kandadisuresh.1@osu.edu.

KAWTHAR SHAFIE KHORASSANI is a Ph.D. student in the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, 43210-1277, USA. Her research interests include high performance computing, and GPU communication and computation. Shafie Khorassani received a bachelor's degree in mathematics and computer science from Wayne State University, Detroit, MI, USA. Contact her at shafiekhorassani.1@osu.edu.

CHEN CHUN CHEN is a third year Ph.D. student in the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, 43210-1277, USA. His research interests include high performance computing and GPU communication. He is a Member of the Network Based Computing Laboratory. Contact him at chen.10252@osu.edu.

BHARATH RAMESH is a fifth-year Ph.D. student in the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, 43210-1277, USA. His research interests include high-performance computing, architecture-aware communication, network/hardware-based offloading, and topology-aware collective algorithms. He is a Graduate Student Member of IEEE. Contact him at ramesh.113@osu.edu.

MUSTAFA ABDULJABBAR has been a research scientist at Network-Based Computing Lab, The Ohio State University, Columbus, OH, 43210-1277, USA, since May 2022. His research interests include high-performance computing systems, hardware-aware parallel optimization, parallel programming models and their interaction with other fields, such as artificial intelligence, machine learning, and numerical algorithms. Contact him at abduljabbar.1@osu.edu.

AAMIR SHAFI is a research scientist at The Ohio State University, Columbus, OH, 43210-1277, USA, where he is involved in the high performance big data and deep learning projects. His research interests include architecting robust communication libraries and tools for HPC systems with emphasis on machine and deep learning applications. Contact him at shafi.16@osu.edu.

HARI SUBRAMONI is an assistant professor in the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, 43210-1277, USA. His research interests include high-performance interconnects and protocols, parallel computer architecture, network-based computing, exascale computing, network topology aware computing, quality-of-service (QoS), power-aware local area network (LAN)-wide area network (WAN) communication, fault tolerance, virtualization, deep learning, big data, and cloud computing. He is a Member of IEEE and ACM. Contact him at subramoni.1@osu.edu.

DHABALESWAR K. PANDA is a professor of computer science and engineering and university distinguished scholar at The Ohio State University, Columbus, OH, 43210-1277, USA. His research interests include parallel computer architecture, high-performance networking, exascale computing, big data, deep learning, programming models, accelerators, high-performance file systems and storage, virtualization, and cloud computing. He is a Fellow of IEEE. Contact him at panda@cse.ohio-state.edu.