2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC) | 978-1-6654-1016-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/HIPC53243.2021.00044

Layout-aware Hardware-assisted Designs for Derived Data Types in MPI

Kaushik Kandadi Suresh, Bharath Ramesh, Chen Chun Chen, Seyedeh Mahdieh Ghazimirsaeed, Mohammadreza Bayatpour, Aamir Shafi, Hari Subramoni, Dhabaleswar K.Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, USA
{kandadisuresh.1,ramesh.113, chen.10252, ghazimirsaeed.3, bayatpour.1, shafi.16, subramoni.1, panda.2}@osu.edu

***Abstract*—Modern MPI-based scientific applications frequently use derived datatypes (DDT) for inter-process communication. Designing scalable solutions capable of dynamically adapting themselves to the complex communication requirements posed by DDT-based applications bring forth several new challenges. In this work, we address these challenges and propose solutions to efficiently improve the performance of hardware-assisted datatype transfers. Further, we design a layout-aware DDT scheme that dynamically adapts the datatype processing to the communication requirements of the datatype layouts used by the application. The proposed layout-aware adaptive scheme is able to dynamically switch between different host-based and the proposed hardware-assisted schemes to deliver the best performance and scalability while hiding the communication overheads. The experimental evaluations on multiple HPC systems including Frontera at TACC and Expanse at SDSC show that our proposed designs achieve up to 22% improvement in performance over state-of-the-art MPI libraries at the micro-benchmark level. We also evaluate our designs with various scientific application kernels such as MILC, WRF, and applications such as miniGhost and demonstrate up to 9% improvement in performance at 128 nodes for the miniGhost application.**

I. Introduction

Modern High-Performance Computing (HPC) systems are enabling domain scientists to solve "grand challenge" problems in their respective fields. With the end of Moore's law in sight, these application scientists are relying on parallelism—by running larger jobs—offered by such HPC systems to push the performance envelope. The Message Passing Interface (MPI) [9] standard is a popular defacto parallel programming model that aids the development of such high-performance scientific applications on large-scale HPC systems.

Many scientific codes define and perform computations on complex, application-specific data structures. For instance, while solving systems of non-linear equations, an application might need to communicate specific columns of a matrix or lower/upper triangular matrix. The challenge here is that these structures are not contiguous in memory. A naive solution is that the application developers pack the data into a temporary contiguous buffer, send it over, and then unpack it on the receiver side. This is not always an efficient approach since it creates multiple copies of data, which is time-consuming and also increases the memory footprint of the application. Moreover, this approach has poor productivity since it requires application developers to handle packing/unpacking and managing the temporary buffers. Therefore, to improve performance and productivity, the MPI standard provides a mechanism called Derived Datatype (DDT) for expressing custom data types—these include contiguous, vector, indexed, and struct. The MPI DDT provides the opportunity to communicate non-contiguous data in a portable manner. The DDT communication schemes currently used in the state-of-the-art MPI libraries (such as Open MPI [2], IntelMPI [1], and MVAPICH [11]) or those proposed in literature [6], [8], [15], [20] can be broadly classified into two categories: "host-based" schemes, and "hardware-assisted" schemes.

In host-based schemes, the host CPU is responsible for packing/unpacking the non-contiguous data by copying all the memory segments to/from a contiguous buffer. State-of-the-art MPI libraries pre-allocate and pre-register a pool of contiguous buffers and then they use this set of internal buffers for packing the data and transferring them through the network. To tackle the overheads of the pack/unpack operations, modern Host Channel Adapters (HCA) such as NVIDIA/Mellanox Infiniband adapters have included the ability to drive non-contiguous datatype transfers on behalf of the host CPU. These supports are generally in form of Scatter and Gather Lists (SGL), where the underlying interconnect hardware provides mechanisms to scatter/gather the non-contiguous data directly to/from host memory.

While such hardware-assisted features in InfiniBand enable the MPI library to hide the copy operations overhead included in the host-based schemes, they cannot be used as a universal solution due to various performance bottlenecks. For instance, in hardware-assisted schemes, the application buffers are directly registered with the HCA at run-time, imposing an overhead every time an application uses a DDT with a new address and layout. If the application reuses the same set of buffers, then there is an overhead regarding maintenance and usage of InfiniBand registration cache in the critical path as well as memory layout transfer between sender and receiver. As we characterize the hardware-assisted

*This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, #2112606, and XRAC grant #NCR-130002

TABLE I
COMPARISON OF DIFFERENT DDT SCHEMES PROPOSED IN LITERATURE

Design Features	Host-based Schemes	Hardware-assisted Schemes	Zero-copy Communication	Adaptive Scheme Selection	Scalability	Layout-aware Registration Cache
Gopal et al	✗	✓	✓	✗	✗	✗
Schneider et al	✓	✗	✗	✗	✓	✗
Traff et al	✓	✗	✗	✗	✓	✗
Girolamo et al	✗	✓	✗	✗	✗	✗
Prabhu et al	✓	✗	✗	✗	✓	✗
LAH (proposed)	✓	✓	✓	✓	✓	✓

schemes, we also realize that there are performance limitations in the hardware for certain datatypes and memory layouts. Furthermore, the current state-of-the-art SGL-based designs suffer from scalability issues due to the usage of an extra connection resource for every peer process [15].

These observations show that while hardware-assisted DDT schemes are useful in certain scenarios, MPI libraries need to address several challenges to mitigate their performance bottlenecks to provide an integrated and high-performance solution.

Table I provides an overview of different state-of-the-art DDT schemes and compares them with the proposed Layout-aware scheme in this paper. As we can see here, there is no solution that enhances the performance bottlenecks regarding the hardware-assisted SGL-based DDT transfers and covers various pillars of sophisticated DDT transfer designs. For instance, Santhanaraman et al [15] propose zero-copy hardware-assisted schemes with scatter-gather lists (SGL). However, their scheme involves expensive layout exchange operations and suffers from scalability issues. Furthermore, it does not hide the performance bottlenecks within hardware, making the performance worse than host-based designs for certain communication scenarios. We address these issues by proposing layout-aware communication transfers whilst minimizing the amount of information exchanged between processes and guarantee scalability. Schneider et al. [17] use run-time compilation techniques to generate efficient and optimized pack code for MPI datatypes at commit time. Prabhu et al [14] propose a declarative language and optimized data movement routines using Just-in-time compilers. The focus of our work is on optimizing network transfers for datatypes, and optimized pack/unpack routines complement our design by accelerating the packing of data in host-based schemes.

II. BACKGROUND

This section provides the background material for our work.

Registration Cache : Infiniband 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", which is a local key used as an identifier for a memory region. Posting sends using Infiniband verbs requires this lkey along with other information such as the memory address and buffer size. To amortize the cost of expensive memory registrations, MPI libraries employ the use of a registration cache. When an MPI_Isend is invoked by a process for a new buffer, the MPI library registers the send buffer with the HCA and adds the generated lkey, buffer address, and buffer size to the registration cache. If a process invokes MPI_Isend again with the same buffer, the MPI library queries the registration cache using the address as a key and uses the value obtained for posting sends, thereby avoiding repeated registration costs. The registration cache resides in the host memory. Typically, MPI libraries use a Binary Search Tree (BST) to cache the entries (indexed by memory addresses). More details are explained in [10].

SGL-based Transfers : Scatter Gather List or SGL refers to the hardware-assisted feature to exchange non-contiguous data. It is defined in the InfiniBand (IB) Architecture's channel semantics. The HCA at the sender's side gathers data from source memory locations and receiver HCA scatters the data to the destination memory locations. The combination of the lkey, buffer address and buffer size are referred to as a Scatter Gather Entry (SGE). Multiple such entries are grouped together to form Scatter-Gather List which is used to post non-contiguous sends/receives to the HCA. In this paper, we use SGL as the hardware-assisted scheme. However, the observations and the proposed designs in SGL can be applied to other hardware-assisted schemes such as User mode Memory Registration (UMR) [1].

Dynamic Connected (DC) Transport : Dynamic Connected transport is a reliable transport protocol in which a single queue pair (called an Initiator) can dynamically establish a connection with any end-point (called a DC target). It drastically reduces the number of queue pairs required for communication when compared to the Reliable Connected (RC) transport protocol, as one DC initiator can connect to multiple DC targets.

III. CHARACTERIZATION AND MOTIVATION

Motivation 1: HPC applications can exhibit diverse memory layouts. Figure 1 shows the approximate representation of memory layouts in some HPC applications obtained by profiling the DDT layouts they use. The blue boxes represent *blocks*, which contain data relevant to the application. The *stride* by which we access these layouts is essentially the size of the current block plus the gap between the current block and the next block. We use the term "DDT layout" to denote the physical descriptions of the DDT, including starting addresses and lengths for all blocks/segments that are part of the DDT, whereas "Memory layout" refers to the structure/organization

[1] In the rest of the paper, we refer to SGL and the hardware-assisted scheme, interchangeably.

of the DDT in memory. In other words, a DDT layout is a specific instance of a memory layout. As shown in this figure, block lengths, the number of segments and the stride/gap between blocks tend to show huge variability. These factors, along with the number of times a DDT layout is used at run-time (the *frequency*), make it difficult to have a one size fits all communication transfer approach. This leads us to the following motivation: **Can we propose a comprehensive DDT design that achieves the peak performance for various communication scenarios and applications?**

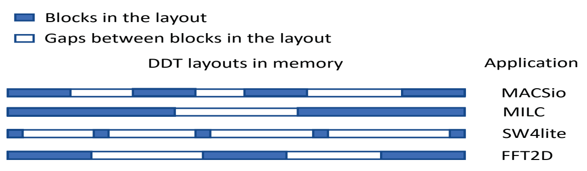


Fig. 1. Application memory layouts. All blocks shown in the figure represent relative sizes and are not drawn to scale.

Motivation 2: Memory registration is a major run-time overhead for hardware-assisted schemes, with a Binary Search Tree (BST) based cache used to amortize repeated registration costs. The number of elements in this BST (N) is proportional to the number of distinct memory addresses which can range from hundreds to several thousand. To register a non-contiguous layout, we need to register all the contiguous segments that are part of the layout. Therefore, the cost of lookup for each segment, and the insertion of entries into a BST-based registration cache would be $\mathcal{O}(log(N))$. First, we study the impact of using a BST-based registration cache for registering all the blocks/segments of different non-contiguous layouts. We wrote a simple MPI-based ping-pong benchmark that exchanges layouts found in HPC applications. Figure 2 shows the breakdown of the total registration time for 100 iterations for three different representative application layouts—which includes the cost of registration using ibv_reg_mr in the first iteration (shown in dark blue), and the cost of cache accesses in subsequent iterations. We observe that the cache access itself represents up to 20% of the total registration time. This problem is exacerbated by increasing the number of times a given DDT layout is used. Thus, we motivate the need to amortize this cost even further, with a goal to provide constant lookup time after registration is complete for memory regions in a given datatype layout. This leads us to the next motivation: **Can we reduce the registration cache overheads in hardware-assisted schemes to get efficient performance?**

Motivation 3: Figure 3 shows the values of performance of hardware-assisted scheme normalized to host-based scheme for varying block length and frequency in a verbs level DDT layout based ping-pong benchmark. Values less than one are red where the hardware-assisted scheme performs better and values greater than one are blue where the host-based scheme performs better. As shown in the figure there is no clear boundary between the two schemes, showing a non-trivial dependence on the latency, frequency, and block size.

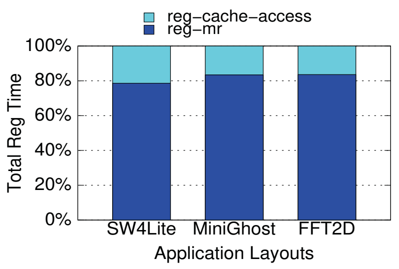


Fig. 2. Registration cache overhead for selected application layouts for 100 accesses

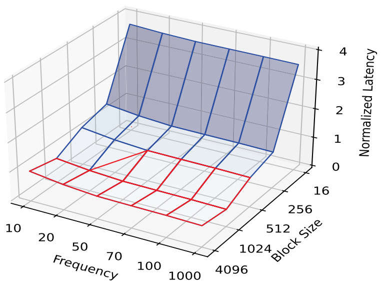
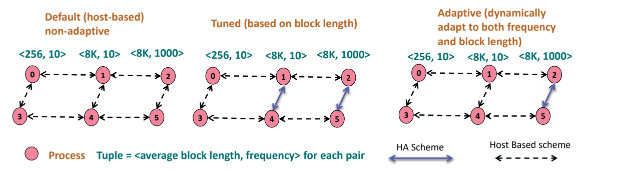


Fig. 3. Impact of block length and layout frequency on the relative performance of Hardware-Assisted scheme with Host-based scheme.

Figure 4 shows an example communication pattern for six processes. The host-based scheme being used is represented using black edges and the hardware-assisted scheme (HA) is represented using orange edges. The tuple between arrow brackets represent the average block length and frequency of the DDT layout being communicated for three pairs of communicating processes : 0-3, 1-4, 2-5. The leftmost figure shows the default case, where all processes use the host-based scheme for all layouts. The figure in the middle shows a tuned case based on the block length threshold, where we select a host-based scheme for any block length less than the threshold (say, 4 KB in this case) and the HA scheme for all average block lengths greater than the threshold. This is an issue for the communicating pair 1-4 in this example as it will not hide the registration cost and end up using the HA scheme for low-frequency communication (which will turn out to be expensive). The figure on the right represents the best selection of algorithms that dynamically adapt to the given communication pattern. Thus, a simple scheme that tunes and selects either a host-based or hardware-assisted mechanism depending on the layout may not always work. There is a need for a scheme that can dynamically give the best possible latency for any DDT layout type and any frequency. This leads us to the last motivation: **Is it possible to design a layout-aware scheme that can dynamically select between**



Default (host-based) non-adaptive
Tuned (based on block length)
Adaptive (dynamically adapt to both frequency and block length)
<256, 10>
<8K, 10>
<8K, 1000>
0
1
2
3
4
5
Process
Tuple = <average block length, frequency> for each pair
HA Scheme
Host Based scheme

Fig. 4. Selection of data transfer schemes for a given communication pattern

host-based and hardware-assisted schemes depending on the frequency of communication and the datatype layout used?

A. Contributions

In this paper, we tackle these questions and demonstrate that neither hardware nor host-based DDT schemes can provide the best performance for various communication scenarios. Thus, an integrated and hybrid design that can take advantage of both these approaches and adapt to the application's communication requirements is essential. This can guarantee the best performance for any communication scenario. We propose multiple solutions to efficiently improve the various phases in the state-of-the-art hardware-assisted schemes while efficiently augmenting the hardware-schemes with host-based schemes.

To summarize, we make the following contributions in this paper:

1) Thorough characterization of state-of-the-art hardware-assisted DDT transfers and identifying the major bottlenecks of current non-contiguous data transfer schemes and the impact of different data layouts on their performance.
2) Evaluating the shortcomings of registration cache mechanisms in current hardware-assisted schemes and propose an efficient hardware-assisted design that significantly reduces the cost of registration.
3) Proposing a mechanism to minimize exchanging the layout information required for every send/receive operation in zero-copy hardware-assisted schemes.
4) Extending the hardware-assisted schemes to take advantage of many benefits of the DC transport protocol, including memory efficiency, scalability, and reliability.
5) Proposing a layout-aware derived datatype design that considers various parameters (e.g. message size, data layout, the frequency of the layouts, etc.) to dynamically select the best scheme for each layout during the application's run time.
6) Demonstrate the benefits of the proposed designs on various applications and microbenchmarks.

The experimental evaluations on multiple HPC systems including Frontera at TACC and Expanse at SDSC show that our proposed designs achieve up to 22% improvement in performance over state-of-the-art MPI libraries at the microbenchmark level. We also evaluate our designs with various scientific applications such as miniGhost and application kernels such as MILC, and WRF and we demonstrate up to 9% improvement in performance at 128 nodes.

IV. Design and Implementation

Figure 5 shows an overview of the proposed designs in the MPI software stack. The proposed hardware-assisted cache (HA-cache) scheme is built on top of hardware-assisted SGL-based transfers and provides multiple enhancements and optimizations to improve its performance. These optimizations include layout-based registration cache, optimized layout exchange design, and DC-based non-contiguous data transfers. We discuss these designs in detail in this section. As

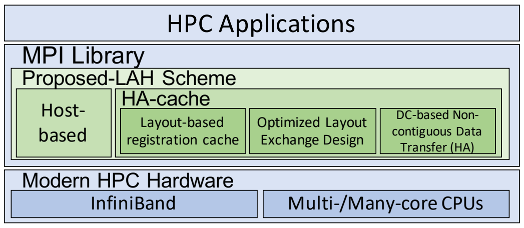


Fig. 5. An overview of the proposed designs

seen in figure 5, the HA-cache and host-based schemes are components of the proposed layout-aware hardware-assisted (Proposed-LAH) scheme which is discussed in detail in Section IV-C.

A. DC-based Hardware-assisted Non-Contiguous Data Transfers (HA)

In this section, we propose a mechanism to take advantage of the DC transport protocol [18] to improve the scalability of the SGL scheme. For non-contiguous datatypes, designing an RC-based solution would require an additional set of endpoints for each source [15], which would be detrimental for large-scale runs as systems can run out of memory. This motivates the need for a DC-based non-contiguous data transfer scheme.

In this scheme, the sender first sends an Request-to-Send (RTS) to the receiver. The receiver maintains a pool of available DC targets. Once the receiver receives RTS, it selects an available DC target from the pool and marks it as unavailable. Then, the receiver sends the target number to the sender through the Clear-to-Send (CTS) packet. The sender receives the DC target and posts all the sends to it. When all the receive operations are done, the receiver makes the associated DC target available. If the receiver does not have any available DC target when it receives a RTS, it queues the RTS request. Once a DC target becomes available, the receiver sends the target number to the sender and removes the associated RTS request from the queue. In the above design, the receiver avoids the case where two different senders post requests to the same DC target. With this scheme, we can provide a scalable DC-based non-contiguous data transfer while ensuring data validity.

B. Optimized DC-based Hardware-assisted Non-Contiguous Data Transfers (HA-cache)

In this section, we describe the design and features of HA-cache—An optimized version of the HA scheme explained in Section IV-A.

1) Layout-based Registration Cache: A DDT layout can be registered block by block (at a page-level granularity) or the entire memory region (including the gaps) can be registered. Since the second approach may fail in certain scenarios [7], we have considered the first approach in this paper. MPI implementations amortize the registration cost through the use of a registration cache, indexed by memory addresses.

Since blocks in a DDT layout are registered one by one and then added to the BST-based cache, the number of elements in the cache (typically implemented as Balanced Binary Search Tree (BST)) is bounded by $\mathcal{O}(l \times count)$, where l is the number of distinct data layouts and $count$ is the maximum number of blocks amongst all layouts. As a result, the cost for accessing a single block is $\mathcal{O}(log(l \times count))$ and the total cost of querying an entire DDT layout is $\mathcal{O}(count \times log(l \times count))$. To reduce query costs with large tree sizes, we propose a layout-based registration cache, which is another layer of cache that augments the existing BST-based registration cache. Implemented as a hash table, this design provides a constant time lookup for each block in a DDT layout. In the following paragraph, we discuss the details of the layout-based registration cache design.

The Layout Based Registration cache works as follows: When a sender/receiver process encounters a datatype handle in MPI_Isend/MPI_Irecv, it combines the datatype handle, base address of the buffer, and message size to generate a composite key, which is used to then query the layout cache. If a miss occurs, the sender flattens the datatype layout (using the handle) to get a list of a structure containing the starting address and the length of each block. Then, the sender/receiver uses the BST-based registration cache to register each block and obtains a list of registration cache entries. This list is added to the layout cache as an entry identified by the generated composite key. On a layout cache hit, the sender/receiver queries the layout cache to get the list of registration cache entries, instead of flattening the layout and querying the registration cache for every block. This reduces the query cost on a cache hit to $\mathcal{O}(count)$ as compared to $\mathcal{O}(count \times log(l \times count))$ in the BST-based registration cache design.

Eviction occurs when 1) a datatype layout is freed (by using MPI_Type_free) or 2) a memory region is freed. All the associated BST-based registration cache entries are evicted first, followed by the eviction of layout-based registration cache entries.

2) Optimized Layout Exchange Design: In the past, researchers [15] have used full layout information exchange to decide on the size and the number of work requests for DDT layout exchange. The problem with this approach is that the amount of layout information exchange and memory usage increase dramatically as we increase the number of blocks in the layout. To deal with these issues, we propose an optimized layout exchange design.

Figure 6 shows the details of a hardware-assisted design using the optimized layout exchange design. The layout-based registration cache behavior works exactly as explained in Section IV-B1. The Optimized Layout Exchange Design works by exchanging the minimum of all layout segment/block lengths in RTS and CTS messages. The sender first sends its minimum segment length ($IOVsmin$) in the RTS message. The receiver uses this value, and its own minimum segment length ($IOVrmin$) to compute $minWQE = m * MIN(IOVsmin, IOVrmin)$. This value is then sent back to the sender in the CTS message. This forms a global agreement between the processes. Thus, the optimized layout exchange design performs a minimal amount of layout information exchange and is complemented by the layout-based registration cache described in Section IV-B1. After receiving the CTS message, the sender registers memory regions and initiates SGL based transfers (based on the agreed chunk size) directly to the receive buffers. The last send is posted as an immediate send to indicate the completion of the transfer to the receiver.

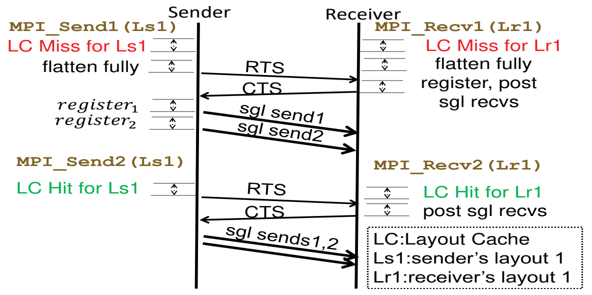
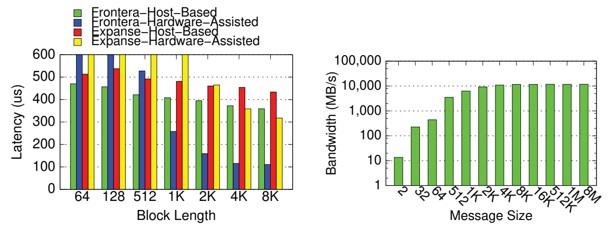


Fig. 6. Hardware-assisted optimized scheme using SGL (HA-cache)

C. Layout-aware Hardware-assisted Design (Proposed-LAH)

In this section, we propose a layout-aware scheme that tries to address the challenges mentioned in Section III to provide efficient data transfers for any DDT layout.



(a) Comparison of Host vs Hardware Assisted on Frontera and Expanse systems for different block lengths (b) One-way bandwidth using verbs level benchmark on Expanse

Fig. 7. Experiments to understand differences between host and HA schemes.

1) Comparison of the Host-Based and HA Scheme: Figure 7(b) shows the one-way bandwidth of sending data for different message sizes. Since small messages do not saturate the bandwidth (as shown in Figure 7(b)), it is beneficial to pack the message (as done in the host-based scheme) to a large enough size in order to saturate the network bandwidth. However, for large block lengths, packing leads to sub-optimal utilization of bandwidth since the peak network bandwidth can already be achieved through direct transfers (without packing). To understand this further, we show the performance of host-based transfers and Hardware Assisted transfer on an MPI level ping-pong benchmark in Figure 7(a) for two different architectures (for details refer section VI) . We observe that after a threshold, the HA scheme starts performing better. The threshold after which the Hardware Assisted scheme performs better depends on the processor, memory, and the HCA. The correct block length threshold for a given system is tuned for different architectures and stored in a table. Standard MPI libraries use such an approach for architecture-dependent parameters like the eager threshold, collectives algorithm selection etc. We use this predefined threshold T in our adaptive design.

2) Adaptive Switching between transfer schemes:

Algorithm 1: Layout-aware Scheme (Main Thread)

```
Input  : DDT layout $L1$, block length threshold $T$, layout
         cache $LC$
Output: Selected scheme $S$
// Scheme selection and initiate RTS
    (sender)
1  Function Send($L1$, $dst\_rank$):
2      $S$ = SelectLocalScheme($L1$, $LC$, $T$)
       SendRTS($dst\_rank$, $S$)
// RTS handler function called by the
    receiver.
3  Function RTSHandler($L1$, $src\_rank$, $src\_S$):
4      $S$ = SelectLocalScheme($L1$, $LC$, $T$);
5      if $src\_S$ == HA-scheme and $S$ == HA-scheme then
6          $Final\_Scheme$=HA-scheme;
7      else
8          $Final\_Scheme$=host-based;
9      SendCTS($src\_rank$, $Final\_Scheme$)
// CTS handler function (sender)
10 Function CTSHandler($L1$, $Final\_Scheme$, $dst\_rank$):
11     if $Final\_Scheme$ == HA-scheme then
12         doHATransfer($dst\_rank$, $L1$);
13     else
14         doHostTransfer($dst\_rank$, $L1$);
15 Function SelectLocalScheme($L1$, $T$, $LC$):
16     $is\_hit \leftarrow$ query $LC$ for $L1$;
17     l $\leftarrow$ average segment length of $L1$;
18     if $is\_hit$ then
19         if $l > T$ then
20             Lock();
               // $f$ depicts if $L1$ was registered
                   by the registration thread
21             $f$= read registration flag of $L1$ in $LC$;
22             Unlock();
23             if $f$==1 then
24                 $S$=HA-scheme;
25             else
26                 $S$= host-based;
27         else
28             $S$= host-based;
29     else
30         $S$ = host-based;
31         flatten $L1$;
32         Add $L1$ to $LC$;
33         if $l > T$ then
34             Wake up registration thread;
35     return $S$
```

Using the observations made on the impact of the block length threshold from IV-C1, we design a communication scheme that dynamically adapts to account for factors described in Section III. In our design, the main thread takes care of using this threshold to decide how to switch between communication schemes. However, hardware-assisted schemes require memory regions to be registered with the HCA before data transfer can happen, which is an expensive operation. This motivates the need for an additional asynchronous thread, that takes care of registering memory regions in the background and in parallel, up to which the main thread can only use host-based schemes. We refer to this asynchronous thread as the "registration" thread.

Algorithm 1 and 2 show the proposed design for these threads. In Algorithm 1, L1 represents the input layout that is used to describe the data type used in MPI_Send or MPI_Recv and the is_hit variable indicates if the layout cache is a hit for layout L1 or not. When MPI_Send/MPI_Recv is called, we first query the layout cache. If a cache hit occurs, then we retrieve the average block/segment length (l) from the layout cache, and based on its value we decide whether to use the HA scheme or not. Since popularly used applications that support derived datatypes such as MILC, miniGhost, etc. do not have a huge variation in segment length for a particular layout, we consider the average segment length as the representative segment length. If l is greater than a predefined threshold T, we check the registration flag (f) for the entry corresponding to $L1$ in the cache. The registration flag indicates if all the blocks for layout $L1$ are registered or not. If the flag is set to 1, then we use the HA scheme, otherwise, we use the host-based scheme. If a layout cache miss occurs, then we use a host-based scheme, where we flatten the layout, pack it, and perform the send in a pipelined manner. After flattening, the main thread populates the layout-based registration cache and wakes up the registration thread.

Algorithm 2 shows the logic of the registration thread. The thread goes to sleep after entering the while loop. When the registration thread is woken up by the main thread, it checks if there are any unregistered entries in the cache. If such an entry is found, it registers each and every block in that cache entry, sets the registration flag to 1, and continues the loop. Once it has checked all the entries, it goes back to sleep. Since the registration happens asynchronously and is overlapped with the main thread, the frequency of use of a layout is implicitly factored into 1. This is because layouts that are used repeatedly will eventually have all their blocks registered during the course of the application, after which the best scheme can be chosen based on the other factors to be considered.

Algorithm 2: Proposed Layout-aware Scheme (Registration Thread)

```
Input  : layout cache $LC$
Output: layout cache with updated registration flags
1  while true do
2      Sleep();
3      foreach $L_i$ in $LC$ do
4          Lock();
5          $f$= read registration flag of $L_i$ in $LC$;
6          Unlock();
7          if $f == 0$ then
8              Register $L_i$;
9              Lock();
10             Set $f$ to 1;
11             Unlock();
```

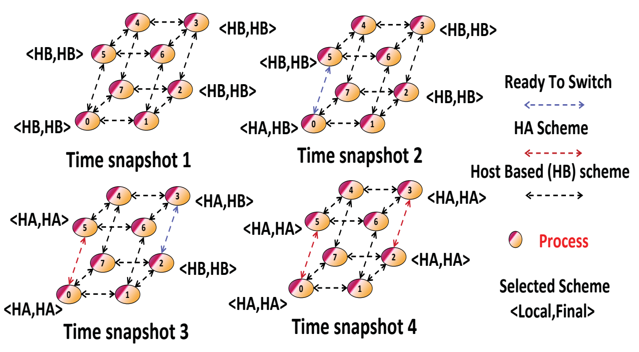


Fig. 8. Selection of schemes for an example communication pattern over four time snapshots representing 0%, 25%, 75%, 100% of total run time.

V. Design Discussion

Figure 8 shows an example communication pattern. The tuple representing "Local" and "Final" selected schemes are the same as represented in Algorithm 1. At time snapshot 1, all processes use the host-based scheme (represented as black edges). Registration begins to happen asynchronously at this point and the threshold (T) is used to determine whether the switch to the HA scheme (represented as red edges) is necessary or not. At time snapshot 2, processes that frequently communicate are ready to switch (edges are represented in blue). This demonstrates the "adaptive" portion, where only processes that communicate frequently at the application level become candidates for switching. Time snapshot 3 shows processes that switch to the HA scheme (others use the host-based scheme) as well as processes that just completed registration (ready to switch). Time snapshot 4 shows the final schemes selected by every process for communication. A manually "tuned" algorithm that picks the algorithm based on the block-length threshold will not hide the registration cost and end up using the HA scheme for low-frequency communication (which will turn out to be expensive).

Even though an additional thread is used for reducing the impact of registration costs, registration is a one-time operation for any new layout due to which the thread will not be active throughout the application's run. Therefore, the impact of such a thread on any other existing threads (say, application compute threads using OpenMP) will be minimal and amortized in the long run.

Thus, the layout-aware scheme provides an efficient solution to varying layout patterns in applications by taking the frequency, block length, and the behavior of both host-based and hardware-assisted schemes into account.

VI. Experimental Evaluation

We implemented the proposed schemes in the MVAPICH2 MPI library. In this section, we compare the performance of the proposed layout-aware hardware-assisted scheme (proposed-LAH) and the enhanced hardware-assisted scheme (HA-cache) with state-of-the-art communication runtimes: MVAPICH2-2.3.5 (MVAPICH2), Intel MPI 2019 (Intel-MPI) and OpenMPI-4.1 (OpenMPI), MPICH-3.4.1 (MPICH) with datatype support. MVAPICH2 is our unmodified baseline.

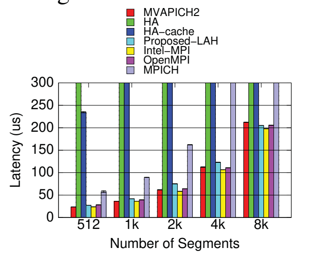
We ran experiments on the TACC Frontera and SDSC Expanse systems. The Frontera system has 8,008 compute nodes with dual-socket Intel Cascade Lake processors running at 2.7 GHz with 28 cores per socket, 192 GB of RAM, and a Mellanox IB-HDR100 (100G) interconnect. The Expanse system has 728 compute nodes with dual-socket AMD EPYC 7742 processors running at 2.25GHz with 64 cores per socket, 256GB of DDR4 RAM, and a Mellanox IB-HDR100 (100G) interconnect.

For the micro-benchmark evaluations, we used a modified version of the osu_latency benchmark. The osu_latency benchmark of OSU Microbenchmark Suite (OMB) [12] is a simple ping-pong test that measures one-way latency averaged over multiple iterations. We modified osu_latency to add MPI_Type_Vector derived datatype support (referred to as the "vector benchmark" in Section VI-A). In the modified version, each process sends and receives a regular layout with a configurable block length, stride, and count. Each test was run for 100 iterations and an average of 5 runs is reported. In addition, we have added standard error as error bars in the graphs which was calculated by dividing the standard deviation by square root of the number of runs. Since the variation is small, they are not visible in most of the graphs. To evaluate the layouts used in real applications, we ran DDT bench [16] which contains the kernel of many HPC applications using MPI Derived Data Types. For large-scale evaluations, we used FFT communication benchmarks and the miniGhost application. For all evaluations of the Proposed-LAH scheme on Expanse, the registration thread is bound to a different core than the main process that spawned the thread. On Frontera (CPUs with hyper-threading), the registration thread is bound to the same core as the main process. The switching threshold of the Proposed-LAH scheme is set to 2K.

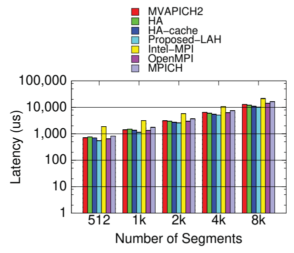
A. Microbenchmark Results

All experiments in this section were conducted on the TACC Frontera system. First, we use the vector benchmark to study the performance of different block lengths. Figures 9(a) and 9(b) show the performance of proposed-LAH, hardware-assisted schemes, and state-of-the-art implementations for representative layouts with small (64 bytes) and large (4KB) block lengths for a varying number of blocks/segments and compare it against state-of-the-art implementations. We observe that Hardware-Assisted schemes (HA, HA-cache) perform poorly for small block lengths when compared to host-based schemes implemented in MPI libraries. The trend reverses for relatively large block length layouts (4KB), with hardware-assisted schemes performing better than host-based schemes. The Proposed-LAH scheme would select the host-based scheme for a block-length of 64 bytes. Therefore, the performance of the Proposed-LAH is almost 10X better than hardware-assisted schemes. For large block length (4KB) layouts, we observe that the Proposed-LAH scheme performs 23% better than the Hardware-Assisted (HA) scheme and up to 16% better than Hardware-Assisted with Layout cache (HA-cache).

The Proposed-LAH scheme performs up to 30 % better than OpenMPI, 3X better than Intel-MPI and 22 % better our baseline (MVAPICH2). The reason for the improvement of the proposed-LAH scheme comes from 1) selection of hardware-assisted scheme for 4KB layout, 2) Reduction of the impact of registration by switching to the hardware-assisted scheme after the registration thread is done with the registration operation.



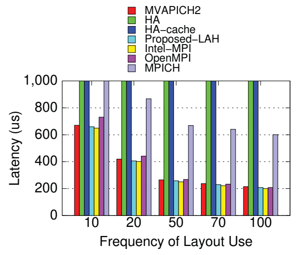
(a) Comparison of proposed schemes with state-of-the-art MPI libraries for 64 byte block size.



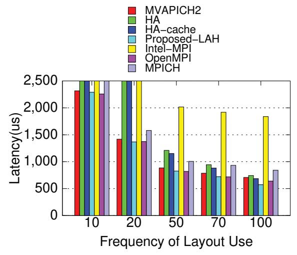
(b) Comparison of proposed schemes with state-of-the-art MPI libraries for 4KB block size in log scale.

Fig. 9. Performance comparison of schemes for representative layouts with small and large block length against state-of-the-art MPI libraries

Next, we study the impact of varying layout frequency on the Proposed-LAH scheme. Figure 10(a) shows the performance of a layout with a block length of 64 bytes for a frequency ranging from 10 to 100. As discussed earlier, the Proposed-LAH scheme selects the host-based scheme, and as expected it performs better than hardware-assisted schemes. Figure 10(b) shows the performance for the layout of the 4k block lengths. We observe that the performance of the hardware-assisted scheme for low frequencies is poor because of the high registration cost. As the frequency increases, the hardware-assisted scheme starts improving as the registration cost is amortized due to the enhanced layout cache designs. Here, we observe that the Proposed-LAH scheme always performs better than the Hardware-Assisted scheme. The reason for this improvement is that the Proposed-LAH scheme starts with the Host-Based scheme for the first few transfers then switches to the Hardware-Assisted scheme for data transfer after the registration thread completes registering the layout, thereby picking the most effective usable scheme for any given layout.



(a) Impact of varying the frequency of layouts for block of 64 and count of 8192



(b) Impact of varying the frequency of layouts for block of 4k and count of 512

Fig. 10. Performance comparison of proposed schemes for representative layouts with different communication frequencies against state-of-the-art MPI libraries

Finally, we study the impact of layout cache on the hardware-assisted scheme. Figure 11 shows the performance of the Hardware-Assisted scheme with and without layout cache for a block length of 4K and frequency of 100. For each block count, the figure shows the split-up registration time (which includes one-time registration cost and the overhead of registration/layout cache) and time to communicate. We observe up to 10 % improvement in total latency which directly comes from the reduction in the cache overhead. The improvement is more prominent as the number of blocks increase in the layout.

Figure 12 shows the OSU Latency benchmark comparison of the proposed scheme with that of our baseline (which is MVAPICH2). This shows that the scheme does not add any overhead for basic datatypes.

We note that our scheme does similar/worse compared to OpenMPI for those cases where the host-based scheme is selected by our design (small block length layouts). The reason for degradation is OpenMPI tends to have better datatype processing engines and copy-based kernels than our library. Our framework is complementary to an existing library's optimized host-based schemes. If applied to other libraries it will still perform similar to or better than their current host-based implementation.

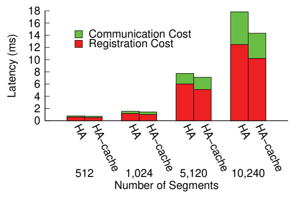


Fig. 11. Impact of proposed layout cache for block size of 4 KB with varying counts

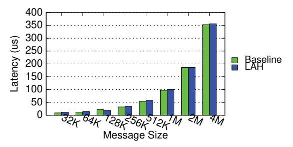


Fig. 12. OSU Latency benchmark using contiguous data types for baseline and Proposed scheme.

B. Application Layout Results using DDT bench

In this section, we evaluate the efficiency of the proposed layout-aware adaptive design on the performance of various applications layouts using DDT bench. The experiments in this section were evaluated on the SDSC Expanse system.

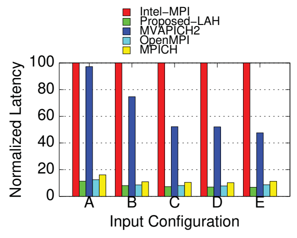
MILC: It studies the integration of quarks and gluons using Quantum Chromodynamics (QCD). The *MILC_su3_zd* kernel in DDTbench models the z-direction of the *su3_rmd* application from the MILC code. It uses nested vector datatype for 4D face exchanges. Figure 13(a) shows that the Proposed-LAH scheme does more than 10× better compared to MVAPICH2 and Intel-MPI. The block lengths of the layouts used range from 700 bytes to 12KB, and the count goes up to 128. The benefit comes from both the adaptive switching of communication schemes (thereby hiding the registration cost) and the layout cache. For this kernel, layout cache contributes significantly because it makes sure that flattening and registration happen only once. Due to the nesting of the layouts, the flattening cost is high for this kernel.

NAS_MG: It is a fluid dynamics application that does 3d face exchanges in x,y,z directions with vector and nested vector datatypes. For inputs selected as shown in figure 13(b), the layout block length is 8 bytes for x-direction, and ranges from

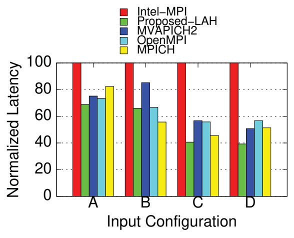
256 bytes, and goes up to 5KB in the y-direction. Since this kernel has both small and very large block lengths, the layout aware scheme selects the appropriate approach for each class of layouts. This, coupled with the adaptive scheme for large block lengths, is the reason why we observe benefits up to 28% compared to MVAPICH2 and up to 2.5× improvements over Intel-MPI.

WRF: It is a weather prediction system that models the atmosphere as a 3-dimensional cartesian grid. The datatypes used are struct of vectors or struct of subarrays, for both x and y directions. We have used the struct of vectors in the y-direction for our experimental evaluation. Figure 13(c) shows the results for three different inputs whose block sizes vary from 2KB to 6KB. We see improvements up to 1.75× compared to MVAPICH2 and up to 2.5× improvements over Intel-MPI.

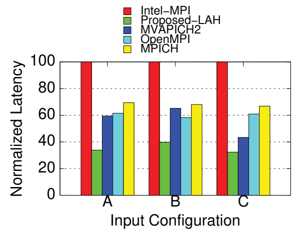
SPECFEM3D_GLOBE: It is a spectral-element application that can simulate global seismic wave propagation through the earth model. We used the *SPECFEM3D_mt* kernel, which uses vector and contiguous data types for data exchange. Figure 13(d) shows the comparison of Proposed-LAH with Intel-MPI and MVAPICH2 for three input configurations. The block lengths vary from 4KB to 12KB. The Proposed-LAH scheme shows improvements up to 33% compare to MVAPICH2 and up to 1.5× over Intel-MPI.



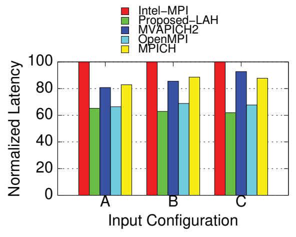
(a) **MILC**. Grid dimensions are A = (16,16,32,32), B = (32,32,32,32), C = (64,32,32,32)), D = (64,64,32,32), E = (64,64,32,64).



(b) **NASMG**. Grid dimensions are A = (256,32,32), B = (512,66,66), C = (2048,66,120), D = (5120,92,120).



(c) **WRF**. Input parameters (ims, ime, is, ie) are A = (4,4014,8,4010), B = (4,2060,8,2056), C = (4,6012,8,6008).



(d) **SPECFEM3D_mt**. Grid dimensions are A = (1024,2,512), B=(2048,2,256), C=(3072,2,200)

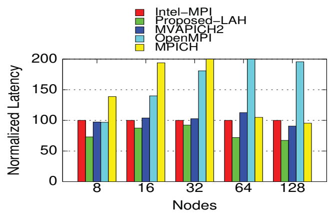
Fig. 13. Normalized performance comparison of Proposed-LAH with state-of-the-art solutions using application kernels for different input sizes. Latencies are normalized with Intel-MPI value set to 100. Lower is better.

C. Large Scale Results

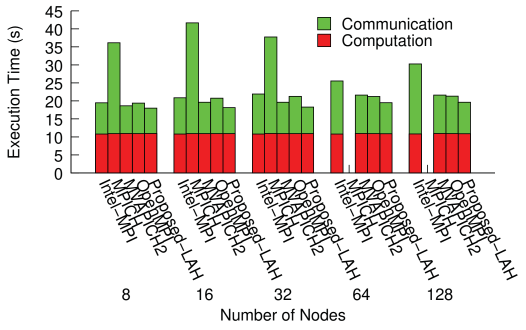
In this section, we provide large-scale results evaluating the efficiency of the proposed design on the FFT benchmark (evaluated on the Frontera cluster) from DDT-Bench, and the MiniGhost application (evaluated on the Expanse cluster).

1) Benchmark-level evaluations: **FFT:** We used the *fft* kernel of DDT bench [16], which runs MPI_Alltoall with vector and contiguous DDTs. We performed a weak scaling experiment by starting with a problem size of 2048 and doubling for every other node count. Figure 14(a) shows the scaling numbers for this kernel up to 128 nodes. We observe that the Proposed-LAH scheme performs up to 32% better than Intel-MPI and 36% better than MVAPICH2.

2) Application-level evaluations : **miniGhost:** [3] performs a 3-D nearest neighbor halo exchange that is present in popular HPC codes. We used a modified version of the MiniGhost application that uses MPI DDTs. Figure 14(b) shows the total execution time of miniGhost application at scale for 1 process per node. The experiments are run with a problem size (nx,ny,nz,nvars) = 30,30,30,3500. We observe that the execution time of the Proposed-LAH scheme is up to 35% better than Intel-MPI, 7.8% better than OpenMPI and 9% better than MVAPICH2 at a scale of 128 nodes. As the scale increases, the benefits of the Proposed-LAH scheme are more prominent. The Figure also shows that the additional threads used by the Proposed-LAH scheme have minimal impact on the compute time of the application. Compute time is close to the same across node counts because they are measured in weak scaling experiments. We have omitted the MPICH results for 64 and 128 nodes as we found that their performance was worse compared to our proposed design which is also evident from the microbenchmark results for large block length layouts (see figure 9(b))



(a) Weak Scaling of FFT2D-Alltoall on Frontera



(b) Weak Scaling of miniGhost upto 128 nodes on Expanse

Fig. 14. Execution time comparison of Proposed-LAH against state-of-the-art MPI libraries at scale

VII. Related Work

Researchers in the past have optimized various aspects of MPI derived datatype-based transfers. Schneider et al. [17] use runtime compilation techniques to generate efficient and optimized pack code for MPI datatypes at commit time. Traff et al. [21] propose hybrid approaches that flatten sub-trees on the fly during packing, thereby efficiently flattening the DDT. Byna et al. [19] optimize the memory access time for packing by providing mechanisms to efficiently re-use cache. Perry et al. [13] proposed a compile-time transformation algorithm to reduce the non-contiguous datatype entities inside the application and avoid the copy overhead associated with pack/unpack operation. Gropp et al. [5] proposed concise datatype patterns and showed efficient implementation techniques for designing them. All the above designs are aimed at optimizing DDT costs such as packing, flattening which will optimize host-based and/or HA scheme. Our design is complementary to the above schemes as we choose the best scheme for a layout.

Researchers have also explored using network features to improve the performance of derived data types. Santhanaraman et al. [15] propose a new scheme called Send Gather Receive Scatter (SGRS), to perform zero-copy datatype communication over InfiniBand. However, they have addressed the schemes for specific layouts and have not taken a holistic look at the performance of schemes based on layout frequency and type. Li et al. [8] leverage both UMR feature and RDMA functionality on the IB network and propose designs to improve MPI-derived datatype communications. However, the paper doesn't consider issues with registration and variability of layouts. All designs in our paper can be extended to use UMR instead of SGL, but we focus on SGL due to its widespread availability on production HPC clusters. Some recent works [4], [6] propose mechanisms to take advantage of zero-copy MPI datatype processing for intra-node transfers.

VIII. Conclusion and Future Work

In this paper, we performed a thorough characterization of state-of-the-art hardware-assisted schemes in driving the DDT transfers. We highlighted the shortcomings of these schemes with respect to registration caches, layout exchanges, and scalability. We addressed these challenges and proposed solutions to efficiently improve the performance of hardware-assisted datatype processing. Furthermore, we designed an adaptive, and layout-aware DDT scheme that dynamically adapts the datatype processing to the communication requirements of the datatype layouts used by the application. The proposed layout-aware adaptive scheme can dynamically switch between different host-based and the proposed hardware-assisted schemes to deliver the best performance and scalability while hiding the communication overheads for any communication scenario. The experimental evaluations on multiple HPC systems including Frontera at TACC and Expanse at SDSC show that our proposed designs achieve up to 22% improvement in performance over state-of-the-art MPI libraries at the micro-benchmark level. We also evaluated our designs with various scientific application kernels such as MILC, WRF, and the miniGhost application and demonstrated 9% improvement in performance at 128 nodes. As part of future work, we plan to evaluate the performance of the proposed layout-aware schemes at larger scales and for more applications.

References

[1] Intel MPI Library. http://software.intel.com/en-us/intel-mpi-library/.

[2] Open MPI: Open Source High Performance Computing. http://www.open-mpi.org.

[3] R. F. Barrett, C. T. Vaughan, and M. A. Heroux. Minighost: a miniapp for exploring boundary exchange strategies using stencil computations in scientific parallel computing. *Sandia National Laboratories, Tech. Rep. SAND*, 5294832:2011, 2011.

[4] S. Di Girolamo, K. Taranov, A. Kurth, M. Schaffner, T. Schneider, J. Beránek, M. Besta, L. Benini, D. Roweth, and T. Hoefler. Network-Accelerated Non-Contiguous Memory Transfers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2019.

[5] W. Gropp, E. Lusk, and D. Swider. Improving the Performance of MPI Derived Datatypes. In *Proceedings of the Third MPI Developer's and User's Conference*, pages 25–30. MPI Software Technology Press, 1999.

[6] J. M. Hashmi, S. Chakraborty, M. Bayatpour, H. Subramoni, and D. K. Panda. FALCON: Efficient Designs for Zero-Copy MPI Datatype Processing on Emerging Architectures. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 355–364. IEEE, 2019.

[7] Jiseheng Wu, Wyckoff, and Panda. Supporting efficient noncontiguous access in PVFS over InfiniBand. In *2003 Proceedings IEEE International Conference on Cluster Computing*, pages 344–351, 2003.

[8] M. Li, H. Subramoni, K. Hamidouche, X. Lu, and D. K. Panda. High Performance MPI Datatype Support with User-Mode Memory Registration: Challenges, Designs, and Benefits. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, pages 226–235. IEEE, 2015.

[9] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[10] F. Mietke, R. Rex, T. Hoefler, T. Mehlan, and W. Rehm. Reducing the Impact of Memory Registration in InfiniBand. In *Proceedings of 2005 KiCC Workshop, Chemnitzer Informatik Berichte*, Nov. 2005.

[11] Network-Based Computing Laboratory. MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE. http://mvapich.cse.ohio-state.edu/.

[12] OSU Micro-benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/.

[13] B. Perry and M. Swany. Improving MPI Communication via Data Type Fission. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 352–355, New York, NY, USA, 2010. ACM.

[14] T. Prabhu and W. Gropp. Dame: Runtime-compilation for data movement. *The International Journal of High Performance Computing Applications*, 32(5):760–774, 2018.

[15] G. Santhanaraman, J. Wu, and D. K. Panda. Zero-Copy MPI Derived Datatype Communication over InfiniBand. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 47–56. Springer, 2004.

[16] T. Schneider, R. Gerstenberger, and T. Hoefler. Micro-applications for Communication Data access Patterns and MPI datatypes. In *European MPI Users' Group Meeting*, pages 121–131. Springer, 2012.

[17] T. Schneider, F. Kjolstad, and T. Hoefler. MPI datatype processing using runtime compilation. In *Proceedings of the 20th European MPI Users' Group Meeting*, pages 19–24, 2013.

[18] H. Subramoni, K. Hamidouche, A. Venkatesh, S. Chakraborty, and D. K. Panda. Designing MPI Library with Dynamic Connected Transport (DCT) of InfiniBand: Early Experiences. In J. M. Kunkel, T. Ludwig, and H. W. Meuer, editors, *Supercomputing*, pages 278–295, Cham, 2014. Springer International Publishing.

[19] X.-H. Sun et al. Improving the performance of MPI derived datatypes by optimizing memory-access cost. In *2003 Proceedings IEEE International Conference on Cluster Computing*, pages 412–419. IEEE, 2003.

[20] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and D. Panda. Host-assisted Zero-copy Remote Memory Access Communication on InfiniBand. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 31. IEEE, 2004.

[21] J. L. Träff, R. Hempel, H. Ritzdorf, and F. Zimmermann. Flattening on the Fly: efficient handling of MPI derived datatypes. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 109–116. Springer, 1999.