High Performance Implementation of MPI Derived Datatype Communication over InfiniBand [*]

Jiesheng Wu[†] Pete Wyckoff[‡] Dhabaleswar Panda[†]

[†]Computer and Information Science
The Ohio State University
Columbus, OH 43210
{wuj, panda}@cis.ohio-state.edu

[‡]Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212
pw@osc.edu

Abstract

*In this paper, a systematic study of two main types of approach for MPI datatype communication (*Pack/Unpack-based approaches *and* Copy-Reduced approaches*) is carried out on the InfiniBand network. We focus on overlapping packing, network communication, and unpacking in the Pack/Unpack-based approaches. We use RDMA operations to avoid packing and/or unpacking in the Copy-Reduced approaches. Four schemes (*Buffer-Centric Segment Pack/Unpack, RDMA Write Gather With Unpack, Pack with RDMA Read Scatter, *and* Multiple RDMA Writes *have been proposed. Three of them have been implemented and evaluated based on one MPI implementation over InfiniBand. Performance results of a vector micro-benchmark demonstrate that latency is improved by a factor of up to 3.4 and bandwidth by a factor of up to 3.6 compared to the current datatype communication implementation. Collective operations like MPI_Alltoall are demonstrated to benefit. A factor of up to 2.0 improvement has been seen in our measurements of those collective operations on an 8-node system.*

1. Introduction

The MPI (Message Passing Interface) Standard [17] has evolved as a *de facto* parallel programming model for distributed memory systems. MPI has a number of features that provide both convenience and high performance. One of the important features is MPI *derived datatype*. Derived datatype provides a powerful and general way to describe arbitrary collections of noncontiguous data in memory in a compact fashion. The MPI standard provides run time support to create MPI derived datatypes and use them in other functions, such as regular message passing functions, performing remote memory access (RMA), and I/O operations.

Typically, MPI derived datatypes allow users to have concise representations of many commonly used data layouts [8, 21]. An example is given in Section 2.2. It can be expected that MPI derived datatype could become a key aid in application development. In practice, however, the poor performance of many MPI implementations with derived datatypes [4, 8, 22, 25] becomes a barrier to using derived datatypes. A programmer often prefers packing and unpacking noncontiguous data manually even with considerable effort. Therefore, it would not be surprising that there is no datatype communication in either the NAS benchmarks [3] or the ASCI benchmarks [15]. On the other hand, noncontiguous communication occurs commonly in many applications, such as (de)composition of multi-dimensional data volumes [2, 6], fast Fourier transform, and finite-element codes [4]. Thus, it is very important to provide efficient MPI datatype communication in MPI implementations.

MPI datatype communication involves datatype processing, and noncontiguous data communication (in this paper, unless stated otherwise, we refer datatype to noncontiguous datatype). In many networks which only support transfer of contiguous data blocks, packing data into and unpacking data out of a contiguous buffer are usually used for noncontiguous data communication. There are several potential ways to improve MPI datatype communication accordingly: *Improve datatype processing system* [8, 11, 21]; *Optimize packing and unpacking procedures* [4, 8]; *Take advantage of network features to improve noncontiguous data communication* [25, 28]. In this paper, we focus on the last two areas based on the InfiniBand network.

We systematically study two main types of approach for MPI datatype communication: *Pack/Unpack-based approaches* and *Copy-Reduced approaches* on the InfiniBand network. In the first type of approach, to reduce the impact of pack and unpack costs on the performance of datatype communication, we propose a new technique called *Buffer-Centric Segment Pack/Unpack (BC-SPUP)* to pipeline the three steps in a datatype communication: packing, network communication, and unpacking. This technique offers potential overlaps between packing, network communication, and unpacking. Particularly, InfiniBand provides bandwidth comparable to system memory copy band-

[*]This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #EIA-9986052, #CCR-0204429, and #CCR-0311542.

width, which makes more sense to overlap these three steps. Therefore, the pack/unpack costs visible to applications are reduced effectively. In addition, this technique also reduces dynamic memory allocation and deallocation, memory registration and deregistration by using pre-registered pack/unpack buffers as much as possible.

In Copy-Reduced approaches, the main idea is to use InfiniBand remote direct memory access (RDMA) operations to transfer noncontiguous data in a datatype message directly without packing and/or unpacking. We design three schemes: *RDMA Write Gather with Unpack (RWG-UP)*, which avoids packing on the sender side; *Pack with RDMA Read Scatter (P-RRS)* , which avoids unpacking on the receiver side; and *Multiple RDMA Writes (Multi-W)*, which avoids both packing and unpacking and achieves zero-copy datatype communication.

We design the aforementioned four schemes. We identify their design issues and provide solutions to these issues. Three of them: BC-SPUP, RWG-UP, and Multi-W, have been implemented and evaluated based on MVAPICH [19, 14], an MPI implementation over InfiniBand. In this paper, we make the following contributions:

1. Memory copies in datatype communication have significant impact on the InfiniBand network which offers high bandwidth comparable to memory bandwidth. The proposed Buffer-Centric Segment Pack/Unpack scheme effectively overlaps packing, network communication, and unpacking; and reduces pack/unpack costs visible to applications.

2. RDMA Gather/Scatter functionality can be used to transfer datatype messages efficiently by reducing memory copies. It allows multiple blocks to be transferred in one single operation. This not only reduces the total startup costs, but also increases network utilization.

3. Using multiple RDMA writes to transfer a datatype message is very efficient due to zero-copy with condition that each block size is large enough. Otherwise, the total startup costs and the low network utilization of small messages offset the benefit of zero-copy.

4. Memory registration and deregistration on networks with RDMA capabilities add a new dimension to datatype communication. Our scheme to register and deregister datatype message buffers permits efficient use of RDMA operations for datatype communication.

5. The BC-SPUP, RWG-UP and Multi-W schemes have been implemented and evaluated based on MVAPICH. Significant improvement has been achieved in both point-to-point and collective datatype communication. These schemes perform differently in different cases. A combination of these schemes can be deployed in an MPI implementation. An appropriate scheme can be chosen with respect to the datatype characteristics. We provide a method to do such selection dynamically and intelligently.

The rest of the paper is organized as follows. Section 2 presents an overview of MVAPICH, its datatype communication, and a motivating example that illustrates problems and potential improvements in the current implementation. Section 3 describes the pack/unpack approaches. Section 4 describes three copy-reduced approaches. We describe how to choose an appropriate approach dynamically with respect to the datatype characteristics in Section 5. The performance results are presented in Section 6. We examine some related work in Section 7 and draw our conclusions and discuss possible future work in Section 8.

2. Datatype Communication in MVAPICH

In this section, we first describe MVAPICH, an MPI implementation over InfiniBand [19, 14], including its basic communication protocols and datatype communication. Then we present a motivating example to demonstrate performance problems of datatype communication in MVAPICH and discuss possible ways to improve datatype communication performance.

2.1. Overview of MVAPICH

MVAPICH is a high performance MPI implementation on InfiniBand. Its design is based on MPICH [24] and MVICH [12]. There are two basic protocols in MVAPICH: *Eager* and *Rendezvous*. In Eager protocol, a message is transferred eagerly to a receiver's internal buffer regardless of whether a receive operation had been issued or not. In this protocol, data is first copied into an internal buffer on the sender side. Then it is transferred to an internal buffer on the receiver side. Later, data is copied from the receiver internal buffer into the application buffer. The Eager protocol is used to transfer small and control messages.

In the Rendezvous protocol, the sender and the receiver first perform handshake to synchronize with each other. This synchronization ensures that a matched receive operation has been issued before data transfer. User buffers on both sides are registered and related information is exchanged in the handshake procedure. A zero-copy implementation of the Rendezvous protocol is implemented using RDMA Write operations. The Rendezvous protocol is used to transfer large messages.

In the current MVAPICH, we have not exploited the design space for MPI derived datatype communication over InfiniBand. The MVAPICH datatype communication path is derived from MPICH and MVICH without change. Figure 1 shows the communication paths for both small and large datatype messages in the current MVAPICH. For small datatype messages, there are two memory copies on both send and receive sides. One is between user buffers and pack/unpack buffers. Another is between Eager Protocol internal buffers (pre-registered) and pack/unpack buffers. To transfer a large datatype message, both sides allocate pack and unpack buffers dynamically. The sender packs data into a pack buffer and then RDMA writes data into the receiver's unpack buffer. The receiver unpacks data into the user buffer. Zero-copy messaging happens only between the pack and unpack buffers. In both protocols, pack and unpack buffers are allocated and freed dynami-

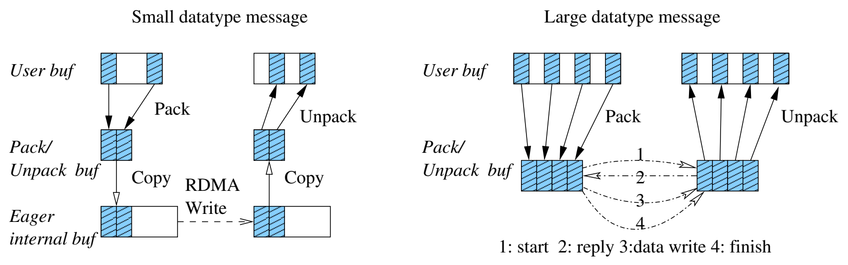


Figure 1. Datatype Communication in MVAPICH

cally. This pack/unpack scheme is a generic datatype communication method in many MPI implementations.

2.2. A Motivating Example

Many MPI implementations perform poorly with derived datatypes [4, 8, 22, 25, 20]. We use a vector datatype ping-pong latency test to show performance problems in the generic scheme and particular issues arisen in MVAPICH over InfiniBand.

Suppose we want to send one or more columns in a two-dimensional 128×4096 integer array from one process to another process. There are several potential schemes. The first scheme builds a derived datatype using `MPI_Type_vector(128, x, 4096, MPI_INT, &newtype)`, where x is the number of columns, and then to use this *newtype* in `MPI_Send()` and `MPI_Recv()`. The second scheme performs manual pack and unpack and only transfers contiguous messages. The third scheme transfers each contiguous block one by one using individual MPI calls. We refer these three schemes as *Datatype*, *Manual*, and *Multiple* schemes, respectively.

One particular issue with the generic scheme on InfiniBand network is that dynamic pack and unpack buffers may incur on-the-fly memory registration and deregistration in each datatype operation. It is possible that different pack and unpack buffers are used in different datatype operations. We call this case as *Datatype plus registration and deregistration* (*DT + reg* for short in Figure 2).

Figure 2 shows a log-log plot of the ping-pong latencies of the aforementioned cases with variable numbers of columns in MVAPICH. As a reference, the latency results for transferring the same size of contiguous data, termed as *Contig*, are also shown.

Several observations can be made. First, no more than *one quarter* of contiguous communication performance is achieved in any scheme. All schemes except *Multiple* have at least two memory copies on top of contiguous communication, one on each side. Second, *Manual* performs a little better than *Datatype*. This is because of datatype processing overhead. Note that the *vector* datatype used is very simple. For some complicated datatypes, this overhead may be significant [11]. Third, *Datatype plus registration and deregistration (DT+reg)* is much slower than *Datatype*. Fourth, *Multiple* performs a little better when the block size is large enough. In this case, Rendzvous protocol is used. However, each individual MPI call needs to pay the handshake overhead. The total cost of all calls degrades the zero-copy benefit. In other cases where Eager protocol is used, performance suffers due to the total protocol overhead and low network utilization.

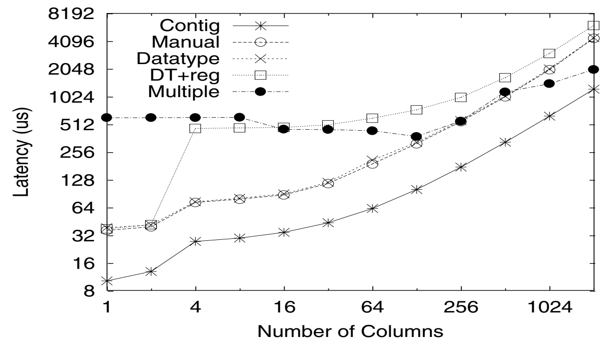


Figure 2. Vector Datatype Transfer Latency Comparison Across Different Schemes.

Therefore, the poor performance of datatype communication comes from: (1) memory copy; (2) memory registration and deregistration; (3) datatype processing; (4) total startup costs of operations; and (5) low network utilization due to small messages. This example motivates us to redesign the datatype communication path in MVAPICH.

In the following Section 3 and Section 4, we present four schemes to improve performance of datatype communication.

3. Pack/Unpack-Based Approaches

In this section, we propose a Buffer-Centric Segment Pack/Unpack scheme (*BC-SPUP*) and discuss its potential benefits and design issues.

3.1. Buffer-centric Segment Pack/Unpack

As discussed in Section 2, the generic pack/unpack scheme has been deployed in many MPI implementations due to its simplicity. However, the performance of this scheme is poor. First, it incurs intermediate copies. Usually two additional copies are required. Second, dynamic pack and unpack buffers incur many memory allocation and deallocation operations. Third, for networks which require that buffers be registered before communication, dynamic pack and unpack buffers may incur on-the-fly memory registration and deregistration. Fourth, packing, communication, and unpacking are serialized because packing and unpacking are performed on a whole datatype message.

To overcome these performance problems, we propose a scheme, called *Buffer-Centric Segment Pack/Unpack (BC-SPUP)*. The BC-SPUP scheme uses two techniques to achieve high performance. First, it uses pre-allocated buffers for pack and unpack operations. These buffers are also optimized and ready for communication, such as aligned on page boundary and registered for RDMA operations. Second, it breaks a datatype message into several segments and applies the basic pack/unpack processing to each segment.

The BC-SPUP scheme has the following potential advantages. First, it avoids dynamic memory allocation and deallocation, reducing system overheads [5]. Second, it eliminates memory registration and deregistration on pack/unpack buffers. Third, these buffers can be well aligned for better communication performance. Fourth, this scheme has the potential to overlap packing, network communication, and unpacking.

Note that this scheme still needs two copies, one for each side. However, most of the copy costs are not visible to end users due to overlap between packing, communication, and unpacking.

3.2. Design Issues in BC-SPUP

There are three issues in the BC-SPUP schemes: how to pipeline the sender's processing; how to pipeline the receiver's processing; and how to deal with the limited pre-registered pack/unpack buffers.

3.2.1 Pipelining Sender Processing

One of the main objectives in the BC-SPUP scheme is to overlap host processing (including datatype processing and packing/unpacking) and communication. This overlap is achieved by breaking a large datatype message into several segments and pipelining the host processing and communication of each segment. This pipelining at the sender has a pronounced effect when operating in the context of a Rendezvous send. In this case it is not possible to start the next message from the application point of view, thus pipelining within this message shows substantial improvement. In the case where the application submits multiple small messages instead of large ones, these will be pipelined independently of our choice of datatype processing. However, to enable a sender to perform packing on any part of a datatype message, partial datatype processing is required. Partial datatype processing allows us to start and stop the processing of a datatype at nearly arbitrary points. Several techniques [21, 11] have been proposed to provide partial processing on MPI datatypes. Another issue is to choose the segment size. Given a datatype message, the segment size should be chosen to provide good overlap and to utilize high network bandwidth as well.

3.2.2 Pipelining Receiver Processing

A receiver must know when each segment arrives to pipeline. This requires a notification per segment for a datatype message. A sender can use RDMA Write with Immediate data to send each segment. Then a receive descriptor is consumed and a completion entry is generated into a CQ. However, this approach requires that multiple receive descriptors have been posted for the coming segments. In another way, the send can write a flag to a specified location in the receiver memory. The receiver then can check this flag to detect the arrival of a segment. This approach can avoid pre-posting receive descriptors and possible flow control. However, it requires one more RDMA operation per each segment or techniques discussed in [14]. In our design, we choose RDMA Write with Immediate data.

3.2.3 Handling Buffer Limit

The size of pre-registered pack/unpack buffer pool should be limited to an appropriate size. We could have a per-connection buffer pool to simplify buffer management; however, it will occupy a significant amount of physical memory and limit scalability. It is desirable that the pack/unpack buffer pool in each process be used to communicate with any remote process. In case of burst communication, the pack/unpack buffer pool may be used up. If all pack buffers are used up, a sender can stop sending messages and wait for completion of previous operation for pack buffers. If all unpack buffers are used up, a receiver can delay response to the sender and then stall the communication until unpack buffers are available. Another solution to this issue is to allocate extra pack/unpack buffers when they are used up. These buffers can be added into the pack/unpack buffer pool. When the total size exceeds some threshold, some of these extra pack/unpack buffers may be deregistered. We choose the second solution. When pack/unpack buffers are used up, we fall back to the dynamic pack/unpack allocation and registration as in the basic pack/unpack scheme.

4. Copy-Reduced Approaches

A common feature in the Pack/Unpack-based approaches described above is that data is copied between user buffers and pack/unpack buffers. Copy-reduced approaches are proposed to reduce or avoid these copies. In this section, we describe three design schemes: *RDMA Write Gather with Unpack*, which avoids packing on the sender side; *Pack with RDMA Read Scatter*, which avoids unpacking on the receiver side; and *Multiple RDMA Writes*, which avoids both packing and unpacking.

4.1. RDMA Write Gather with Unpack

In RDMA Write Gather with Unpack (*RWG-UP*) scheme, only unpack buffers are assigned on the receiver side. A sender uses RDMA Write with Gather operations to write multiple contiguous blocks of a datatype message from its user buffers directly into the receiver's unpack buffer. Then, the receiver unpacks data into its user buffers. Figure 3 shows this scheme. Thus, packing (i.e. one copy) is eliminated on the sender side. Since a relatively large number of blocks can be gathered in one RDMA Write operation (for example, the current Mellanox SDK supports 64 blocks), the total number of RDMA Write operations needed to transfer the whole datatype message is reduced significantly. Therefore, the total startup overhead of all

RDMA operations is reasonably low. In addition, information about a receiver's unpack buffer for performing RDMA Write is simple.

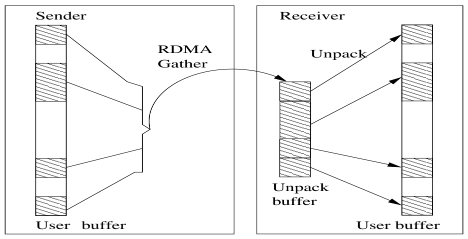


Figure 3. RDMA Write Gather with Unpack Scheme.

This scheme can easily achieve segment unpack as discussed in the BC-SPUP scheme in Section 3.1 to mask the memory copy cost on the receiver side. The sender can break a large message into several segments. Each time it uses RDMA Write with Gather to send a segment and drives the receiver to unpack the incoming segment. To enable RDMA Write operations on a datatype message, the sender needs to register all contiguous pieces of the datatype message. Unlike registering and deregistering a contiguous buffer, memory registration and deregistration on datatype message buffers are more complicated. We discuss how to achieve efficient memory registration and deregistration on datatype message buffers in details in Section 4.4.1.

4.2. Pack with RDMA Read Scatter

As shown in Figure 4, the Pack with RDMA Read Scatter (*P-RRS*) scheme follows the exactly opposite procedure of the above RDMA Write Gather with Unpack scheme. The sender packs a datatype message into a pack buffer, then it asks the receiver to read it directly using RDMA Read operations. The receiver can scatter what it reads into multiple blocks of its datatype message buffer in one single operation.

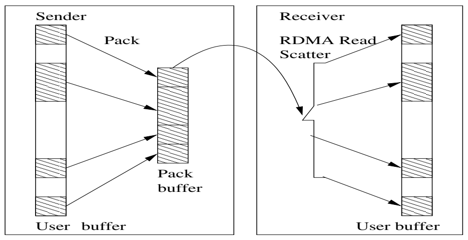


Figure 4. RDMA Read Scatter with Pack Scheme.

This scheme has almost the same requirements as mentioned in the RWG-UP scheme. However, this scheme is a little more costly to pipeline packing and communication. The sender can have segment pack, however, when a segment is available, a control message must be sent to the receiver to trigger the receiver's RDMA Read operation. Another difference is that RDMA Read performance is always lower than that of RDMA Write. This scheme may be useful for asymmetric datatype communication: the sender's datatype is contiguous and the receiver's datatype is noncontiguous. For communication with noncontiguous datatypes on both sides, this scheme will be less efficient than the above RWG-UP scheme. Due to these observations, this scheme is not implemented in our implementation.

4.3. Multiple RDMA Writes

The Multiple RDMA Writes scheme (*Multi-W*) writes each contiguous piece of a datatype message directly into the receiver's buffer, as shown in Figure 5. This scheme can achieve zero-copy datatype messaging. There are two requirements. First, all contiguous blocks of user datatype message buffers must be registered. Second, the sender should be aware of the layouts of contiguous blocks in the receiver user buffer. That means the receiver must send the sender not only its buffer information, but also its datatype information. Then, the sender can decide the source and destination buffers for each RDMA Write operation according to these information.

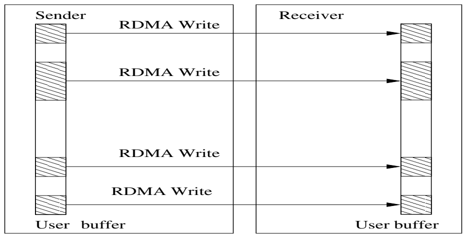


Figure 5. Multiple RDMA Writes Scheme.

One of the disadvantages in this scheme is the number of RDMA operations may be large. The total costs of posting each RDMA write descriptor one by one may be very high. This problem can be alleviated by an extended IBA Verbs interface which supports posting a list descriptors in one call [16]. Another disadvantage is that network utilization may be low because the message sizes in RDMA writes are limited to the sizes of contiguous blocks. However, it can be expected that when these sizes are reasonably large, this scheme can achieve good performance because of zero-copy messaging. The third disadvantage is that some datatype information may be complicated. We discuss how a receiver can send its datatype information to a sender efficiently in Section 4.4.2.

4.4. Design Issues in Copy-Reduced Approaches

Copy-Reduced approaches show very attractive potential benefits due to reduced memory copies. However, several issues need to be addressed for these schemes to be used in MPI implementations to achieve high performance of datatype communication. Note that in heterogeneous systems, the byte-order issue in the copy-reduced apporaches is out of the scope of this paper. We limit our discussion to a system in which RDMA operations can be performed directly on user buffers.

4.4.1 Reducing Memory Registration Overhead

Reducing overheads of memory registration and deregistration on datatype message buffers is a common issue in all three Copy-Reduced schemes. Techniques such as Pin-down cache [9] and FMRD [27] mainly deal with registration and deregistration of contiguous buffers. There is another complication in registering datatype message buffers due to data noncontiguity in these buffers.

In [28], we proposed an efficient approach, *Optimistic Group Registration (OGR)* to register a list of arbitrary buffers. OGR also deals with the situation in which gaps between two buffers may not really have been allocated by the application. Thus, it is a more general case than the case of MPI datatype message buffers.

The Optimistic Group Registration scheme first groups buffers into several memory regions. A cost model is used to achieve the tradeoff between the number of operations and the buffer size to be registered and deregistered. Details of the model can be found in [28]. Large gaps which nulls any benefit over individual registration are filtered out. Then, it registers each region independently. The effectiveness of this scheme has been demonstrated in [28] as well.

4.4.2 Handling Receiver's Datatypes

A unique issue arises in the Multiple RDMA Writes scheme: handling the receiver's datatype on the sender side. The sender needs to know the layout of data in the receiver Datatype message buffer.

An MPI datatype can be represented by a linear list of `<offset, length>` tuples. Other light-weight representation formats such as type tree [11] and dataloop [21] can be used here to reduce the size of datatype representation messages. To avoid sending datatype representation for each operation, a datatype cache mechanism [10] can be used. This cache mechanism was proposed by Träff *et al* in the context of performing MPI-2 [18] one-sided communication. We extend this cache mechanism to handle datatype free and datatype index reuse. In case the receiver frees a datatype, and reuses the type index for a new datatype, the receiver is responsible for sending the new datatype representation. To achieve this, we associate a version number with each type index. When a type index is reused, its version number increases by one. The receiver can detect the version number change and then send the new datatype presentation to the sender. The sender simply replaces the obsolete datatype in its cache with the new one.

5. Choosing an Appropriate Approach

We discussed four main schemes to transfer datatype messages in the last two sections. These schemes have their own advantages and disadvantages. Accordingly, they offer different performance in different situations. An interesting question is: *Given a datatype communication, can we choose the best approach to perform data transfer?* In this section, we discuss a method to choose an appropriate approach for a given datatype message communication dynamically and intelligently.

5.1. Small Datatype Messaging

Small datatype messages are transferred using the Eager protocol, as mentioned in Section 2.1. Since the amount of data copied is relatively small, the BC-SPUP scheme with only one segment can be used to achieve both high performance and simplicity. Different from the generic pack/unpack scheme (shown in Figure 1), the Eager protocol internal buffers are used as the pack/unpack buffers directly. One memory copy on each side is eliminated.

5.2. Large Datatype Messaging

Large datatype messages are transferred using the Rendezvous protocol. The handshake in this protocol (shown in Figure 1) can help the sender choose an appropriate approach by taking both its and the receiver's datatype characteristics into account. The basic procedure is as follows.

First, both the sender and the receiver get the statistic information of its datatype, including the total size, the average contiguous block size, the median contiguous block size, and the deviation of contiguous block sizes. These information can be obtained statically or dynamically.

Second, the sender sends a Rendezvous start control message to the receiver. While waiting for the Rendezvous reply message, the sender uses its datatype statistic information to decide whether pack is used or not. Some simple rules can be applied. Given a datatype, if its average contiguous block size is very small and the deviation among all contiguous block sizes is also small, it is highly possible that layout of data in this datatype is very sparse. Packing these sparse data into a contiguous buffer may be better. When the sender has its initial decision, it can begin to perform either packing or memory registration.

Third, when the Rendezvous start arrives and a matched receive request is issued, the receiver decides whether the segment unpack is beneficial or not. The same rules as on the sender side can be applied. If the layout of data is sparse, the receiver can assign an unpack buffer. Otherwise, user buffers are used directly. In the first case, only the unpack buffer information is sent in the Rendezvous reply message, indicating that the unpack scheme is chosen. In the second case, the datatype information and the datatype message buffer information are sent in the Rendezvous reply message, indicating that RDMA operations on the user buffers are expected.

Lastly, when it receives the Rendezvous reply message, the sender knows the receiver's decision. Combining the receiver's decision and its initial decision in the second step, the sender can finalize its decision and start communication with the appropriate scheme. Table 1 shows how the sender makes its final decision.

Table 1. Choosing an appropriate approach

Sender's Initial Decision	Receiver's Decision	Sender's Final Decision
Pack	Unpack	BC-SPUP
Pack	Copy-Reduced	Multi-W
Copy-Reduced	Unpack	RWG-UP
Copy-Reduced	Copy-Reduced	Multi-W

The proposed method takes full advantage of the handshake in the Rendezvous protocol and incurs little overhead. Note that it is possible that different schemes can be used in different parts of a single datatype message. For simplicity, we choose only one scheme for an entire datatype message.

6 Performance Results

We individually implemented the Buffer-centric Segment Pack/Unpack (BC-SPUP), RDMA Write Gather with UnPack (RWG-UP), and Multiple RDMA Write (Multi-W) schemes into MVAPICH [14, 19]. Due to space limitation, the implementation details are not covered in this paper. Interested readers can refer to [26]. This section presents performance results from a range of benchmarks on our implementations of three schemes in MVAPICH. We attempted to find some standard benchmarks such as the NAS benchmarks [3] and the ASCI blue benchmarks [15], in which we wished datatype communications were used. Unfortunately no noncontiguous datatype communication is used in these benchmarks yet, perhaps due to the limited performance achieved in most applications. SKaMPI [20] provides benchmark for MPI derived datatypes. The test datatypes are synthetic and most parameters are defined by users. For simplicity, we developed our own benchmarks, with intention to capture typical usage of derived datatypes. Unless stated otherwise, the unit megabytes (MB) in this paper is an abbreviation for 2^{20} bytes, or 1024×1024 bytes.

In this section, we first compare point-to-point latency and bandwidth, and performance of collective operations in different implementations. Then, we quantify effects of several design choices on the performance of datatype communication, including segment unpack, list descriptor post and buffer usage.

6.1. Experimental setup

Our experimental testbed consists of a cluster system consisting of 8 nodes built around SuperMicro SUPER P4DL6 motherboards and GC chipsets which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 2.4 GHz processors with a 512 kB L2 cache and a 400 MHz front side bus. The machines are connected with Mellanox InfiniHost MT23108 DualPort 4x HCA adapter through an InfiniScale MT43132 Eight 4x Port InfiniBand Switch. The Mellanox InfiniHost HCA SDK version is thca-x86-0.2.0-build-001. The adapter firmware version is fw-23108-rel-1_17_0000-rc12-build-001. We used the Linux RedHat 7.2 operating system.

6.2. Vector Micro-benchmark

We evaluated the same example described in Section 2.2 in three new implementations: BC-SPUP, RWG-UP, and Multi-W. Unless stated otherwise, in our tests, segment unpack was enabled in the RWG-UP scheme and list descriptor post was enabled in the Multi-W scheme. In this benchmark, a certain number of columns in a two-dimensional 128×4096 integer array are transferred between two processes. These columns can be represented by a vector datatype shown in Section 2.2. The number of columns varies from 1 to 2048.

Figure 6 compares ping-pong latencies in different implementations, including the current MVAPICH datatype implementation ("Generic"). BC-SPUP performs better than the Generic scheme consistently. It gives a factor of 1.5 improvement over the Generic scheme for large datatype messages. RWG-UP performs better than the Generic scheme in most cases, except that the size of contiguous block is too small for RDMA operations to achieve good performance. It gives a factor of up to 1.8 improvement over the Generic scheme. Multi-W offers a factor of 3.4 improvement when the number of columns is large. When the size of contiguous blocks is small, Multi-W performance degrades significantly.

Figure 7 compares their bandwidth. In the bandwidth test, the same vector datatype is used. The sender pushes 100 consecutive datatype messages and then waits for a reply from the receiver when all messages have been received. Both BC-SPUP and RWG-UP give a factor of 1.2–2.0 improvement over the Generic scheme. Multi-W gives a factor of 1.4–3.6 improvement over the Generic scheme when the number of columns is larger than 64. Similarly, when the number of columns varies from 4 to 64, Multi-W performance degrades a lot because of the large number of RDMA Write operations and the small message size in each operation.

When the number of columns is 1 or 2, the datatype message follows the Eager protocol and has the same communication path in all BC-SPUP, RWG-UP and Multi-W schemes. Thus, the performance is identical. Compared to the Generic scheme, two copies are saved as mentioned in Section 5.1. Thus, there is perceivable improvement over the Generic scheme.

6.3. Performance of MPI_Alltoall

Collective datatype communication can benefit from high performance point-to-point datatype communication provided in our implementations. We noticed that some collective operations such as `MPI_Bcast` perform explicit pack and unpack operations in their implementation when noncontiguous datatype communication occurs [23], these collective operations will not benefit from the performance improvement of point-to-point datatype communication achieved in our implementations. Many of others, which still use point-to-point noncontiguous datatype communication in their implementation [23], can benefit from our implementations.

We designed a test to evaluate `MPI_Alltoall` performance with derived datatypes. In the previous tests, all block sizes are same. In this test, we designed a structure datatype in which the size of its contiguous blocks is different. The datatype is designed as shown in Figure 8: the block size varies from one integer to x integers. The gap (empty blocks in the plot) between two blocks equals to the size of the first block.

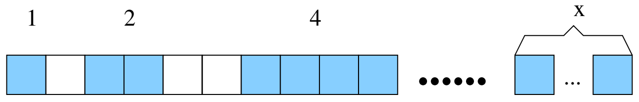
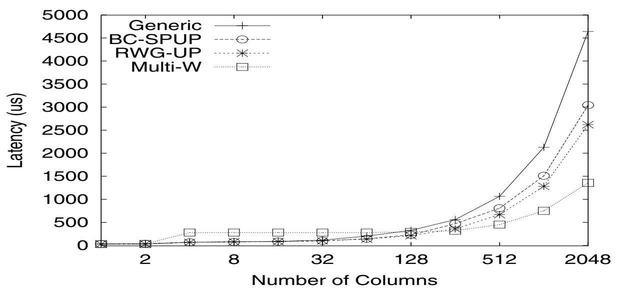


Figure 8. A Struct Datatype.

Figure 9 compares MPI_Alltoall performance of four datatype communication implementations. We use 8 pro-



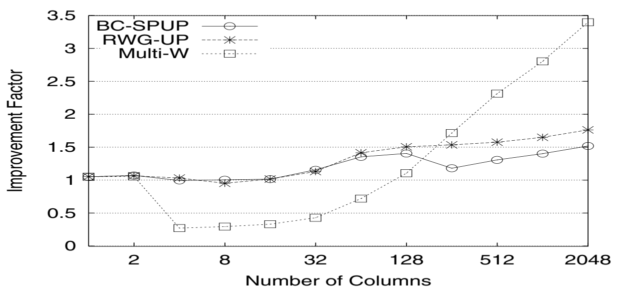
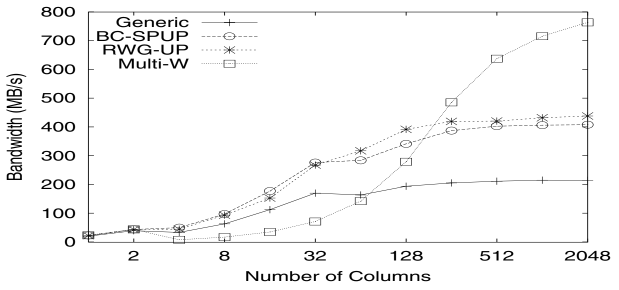


Figure 6. Latency Comparison.



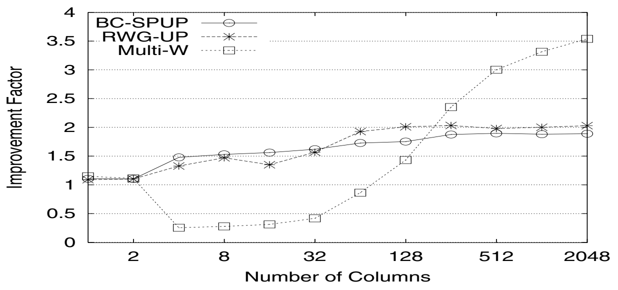


Figure 7. Bandwidth Comparison.

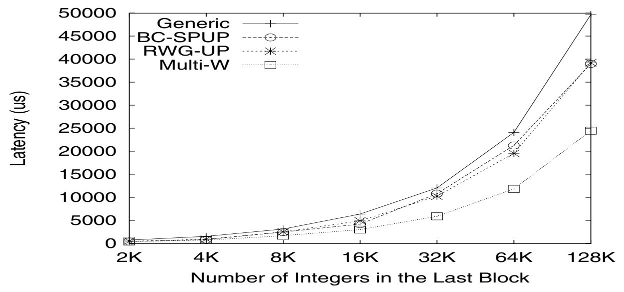


Figure 9. MPI_Alltoall Performance.

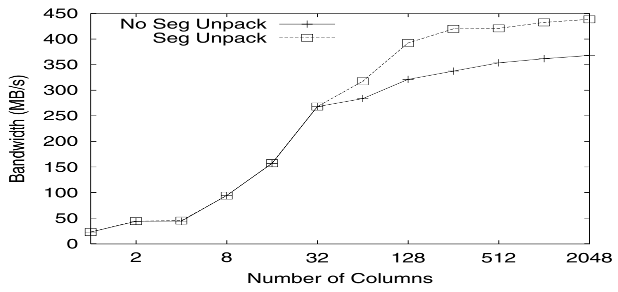


Figure 10. Effects of Segment Unpack.

cesses. The number of integers in the last block varies from 2048 to 131072, as shown in the x-axis. The block sizes increase exponentially from 4 bytes to the largest block size from the first block to the last block. For example, when the number of integers in the last block is 8192, the block sizes vary from 4 bytes to 32768 bytes. We can see that all BC-SPUP, RWG-UP and Multi-W schemes outperform the Generic scheme. BC-SPUP gives an improvement factor of minimum 1.2, maximum 1.5, and average 1.3. RWG-UP gives an improvement factor of minimum 1.2, maximum 1.4, and average 1.3. Multi-W gives an improvement factor of minimum 1.8, maximum 2.1, and average 2.0. For this datatype, it can be observed that Multi-W is a good choice.

Measurements for other collective operations, which could not be presented in this paper due to space limitation, have shown similar results as we observed in the test of MPI_Alltoall [26].

6.4. Effects of Segment Unpack

To quantify the effects of segment unpack in the RWG-UP scheme, we disabled the unpack trigger in each segment. Then, the receiver begins to unpack until the whole datatype message arrives. It can be expected that the segment unpack gives us better performance due to the overlap between communication and unpacking. We used the aforementioned vector bandwidth test to quantity the effects of segment unpack. Figure 10 shows that a factor of 1.3 improvement in bandwidth can be achieved using the segment unpack.

6.5. Effects of List Descriptor Post

As mentioned in Section 4.3, we have two methods to post a list of RDMA write descriptors: single post many times and list post once. We evaluated the vector bandwidth test on these two methods of the Multi-W scheme. Figure 11 shows that the list post offers improvement with a maximum factor of 2.0 and a minimum factor of 1.2 over the single post. The average improvement factor is 1.6. This has two implications. First, posting descriptor is costly and we expect InfiniBand vendors to further optimize it. Second, list descriptor post is a good extension and should be supported.

6.6. Effects of Buffer Usage

Buffer usage has a great impact on MPI communication performance [13]. Performance achieved in schemes

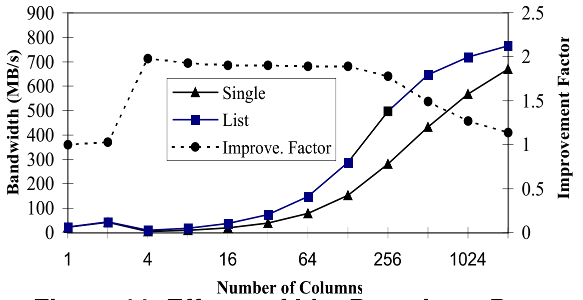


Figure 11. Effects of List Descriptor Post.

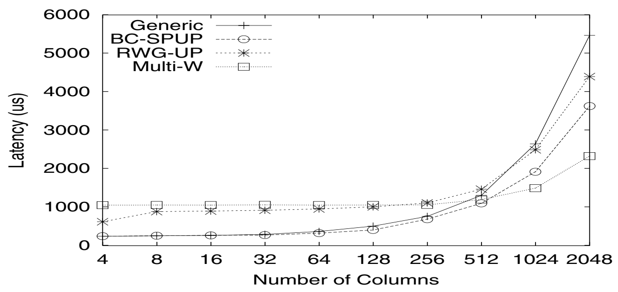


Figure 12. Latency Comparison in the Worst Case of Buffer Usage.

we discussed has different dependency with buffer usage, for either application buffers or MPI internal buffers. Particularly, as shown in Section 2.2, the pack/unpack buffers used in the Generic scheme are dynamically allocated and potentially change in datatype communication operations. It is highly probable that memory registration is necessary for each datatype communication. The BC-SPUP scheme uses a pre-registered buffer pool to reduce the impact of dynamic memory allocation and registration to some extent. In the RWG-UP scheme, the unpack buffer is also pre-registered. However, both schemes will need to stall communication when the buffer pool is used up or register more buffers. Both RWG-UP and Multi-W schemes register user buffers. Their performance heavily depends on user buffer usage patterns. For example, if an application keeps using different buffer in each operation, the registration becomes necessary. To show the buffer usage effect, we conducted the vector latency test in a worst scenario. That is, if a scheme needs an internal buffer, the internal buffer is allocated, registered, and deregistered on-the-fly; if a scheme uses user buffers directly, the user buffers are different, dynamic registration and deregistration are included.

Figure 12 shows the worst latencies for each scheme. When the number of columns is less than 512, both RWG-UP and Multi-W schemes perform very poor. This is because they need to register and deregister the whole user array (registering each block is even more costly in this case [28]), while Generic and BC-SPUP only register and deregister buffers with the real data size. The memory registration and deregistration costs dominate their performance. When the number of columns increases, the difference in the costs of registration and deregistration between these schemes decreases. While the memory copy costs catch up, both RWG-UP and Multi-W perform better than Generic due to reduced memory copies. In this test, BC-SPUP always performs better than Generic. Since they both have same registration and deregistration costs, the benefits completely come from the overlap between packing, communication, and unpacking due to the segment pack/unpack technique.

7. Related Work

Our work is related to previous studies in the following areas.

Improving datatype processing system: Gropp *et al.* [8] have provided a taxonomy of MPI derived datatypes according to their memory access patterns and described how to efficiently implement these patterns. Träff *et al.* have described a technique, called flattening on the fly, for improving the datatype processing system [11]. Ross *et al.* [21] have designed a reusable datatype-processing component for the MPICH2 implementation [1]. This component can be used in our implementations to further improve performance as our future work.

Optimizing packing and unpacking procedures: Byna *et al.* [4] have presented a technique which selects an appropriate packing algorithms with respect to the architecture-specific parameters and the datatype memory access patterns. Recently, MPICH2 [1] has begun to deploy segment pack and unpack in its implementation. The Los Alamos Message Passing Interface (LA-MPI) system [7] has used shared memory regions as pack and unpack buffers in its datatype communication path.

Taking advantage of network features to improve noncontiguous data communication: In [25], Worringen *et al.* have presented a direct copy technique to improve performance of datatype communication using shared memory region provided by the SCI network. In [28], we have demonstrated the benefits of using RDMA Gather/Scatter operations to support noncontiguous file access in PVFS over InfiniBand. Only the case in which buffers on the I/O server side are always contiguous is discussed.

None of these previous work discusses and analyzes the benefits of segment pack and unpack. Issues to register and deregister pack/unpack buffers on RDMA-capable networks are not addressed yet. Furthermore, using RDMA operations to reduce memory copies in datatype communication and its associated design issues are also not addressed in these previous work.

8. Conclusions and Future Work

In this paper, we systematically study two main types of approach for MPI datatype communication: *Pack/Unpack-based approaches* and *Copy-Reduced Approaches* on the InfiniBand network. Along the first type of approach, a *Buffer-Centric Segment Pack/Unpack (SPUP)* scheme is proposed to overlap packing, communication and unpacking in datatype communication and reduce memory registration and deregistration costs. Along the second type of approach, we propose three schemes: RDMA Write Gather with Unpack (RWG-UP), Pack with RDMA Read Scatter (P-RRS), and Multiple RDMA Writes (Multi-W). The main idea behind these schemes is to use RDMA operations to reduce and/or eliminate packing and unpacking in datatype communication.

Three of these four schemes, BC-SPUP, RWG-UP, and Multi-W, have been implemented and evaluated in MVAPICH over InfiniBand. Performance results with both point-to-point and collective benchmarks show significant performance improvement can be attained compared to the generic pack/unpack scheme.

We also notice that these schemes perform differently in different cases. We propose a method which dynamically chooses an appropriate scheme to fit a given datatype communication and to achieve the best performance. We are working on the implementation of this selection method. We plan to integrate the implementation into MVAPICH and release it publicly after we have more application level tests and performance tuning.

Acknowledgments

We would like to thank David Ashton, Darius Buntinas, Rob Ross and Neill Miller at Argonne National Laboratory for MPI derived datatype discussion with us. We are also thankful to nowlab fellows, Dr. Hyun-Wook Jin, Jiuxing Liu, Weikuan Yu and Amith Mamidala or their helpful comments on the paper draft. We also thank the anonymous reviewers for many valuable comments.

References

[1] Argonne National Laboratory. MPICH2 Release 0.94. `http://www-unix.mcs.anl.gov/mpi/mpich2/`, August 2003.

[2] M. Ashworth. A Report on Further Progress in the Development of Codes for the CS2. In *Deliverable D.4.1.b F. Carbonnell (Eds), GPMIMD2 ESPRIT Project, EU DGIII, Brussels*, 1996.

[3] D. H. Bailey, E. Barszcz, L. Dagum, and H. Simon. NAS Parallel Benchmark Results. Technical Report 94-006, RNR, 1994.

[4] S. Byna, X.-H. Sun, W. Gropp, and R. Thakur. Improving the Performance of MPI Derived Datatypes by Optimizing Memory-Access Cost. In *Proceedings of the IEEE International Conference on Cluster Computing*, 2003.

[5] P. Ezolt. A Study in Malloc: A Case of Excessive Minor Faults. In *Proceedings of the 5th Annual Linux Showcase and Conference*. USENIX Association, 2001.

[6] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modelling Astrophysical Thermonuclear Flashes. *Astrophysical Journal Suppliment*, 131:273, 2000.

[7] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. Minnich, C. E. Rasmussen, L. Dean Risinger, and M. W. Sukalski. A Network-Failure-tolerant Message-Passing system for Terascale Clusters. In *Proceedings of the 2002 International Conference on Supercomputing*, June 2002.

[8] W. Gropp, E. Lusk, and D. Swider. Improving the Performance of MPI Derived Datatypes. In *MPIDC*, 1999.

[9] H. Tezuka and F. O'Carroll and A. Hori and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *12th Int. Parallel Processing Symposium*, March 1998.

[10] J. L. Träff, H. Ritzdorf and R. Hempel. The Implementation of MPI–2 One-sided Communication for the NEC SX. In *Proceedings of Supercomputing*, 2000.

[11] J. L. Träff, R. Hempel, H. Ritzdorf and F. Zimmermann. Flattening on the Fly: Efficient Handling of MPI Derived Datatypes. In *PVM/MPI 1999*, pages 109–116, 1999.

[12] Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture, August 2001.

[13] J. Liu, B. Chandrasekaran, J. W. W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda. Performance Comparison of MPI Implementations over InfiniBand Myrinet and Quadrics. In *Supercomputing 2003: The International Conference for High Performance Computing and Communications*, Nov. 2003.

[14] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *17th Annual ACM International Conference on Supercomputing*, June 2003.

[15] Los Alamos National Laboratory. The ASCI Blue Benchmarks. `http://www.llnl.gov/asci-benchmarks/`.

[16] Mellanox Technologies. Mellanox InifniBand Technologies. http://www.mellanox.com.

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

[18] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.

[19] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. `http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html`, January 2003.

[20] R. Reussner, J. L. Träff, and G. Hunzelmann. A Benchmark for MPI Derived Datatypes. *Lecture Notes in Computer Science*, 1908:10+, 2000.

[21] R. Ross, N. Miller, and W. Gropp. Implementing Fast and Reusable Datatype Processing. In *EuroPVM/MPI*, Oct. 2003.

[22] R. Ross, D. Nurmi, A. Cheng, and M. Zingale. A Case Study in Application I/O on Linux Clusters. In *SC2001*, Nov. 2001.

[23] R. Thakur and W. Gropp. Improving the Performance of Collective Operations in MPICH. In *EuroPVM/MPI*, Oct. 2003.

[24] W. Gropp and E. Lusk and N. Doss and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard.

[25] J. Worringen, A. Gaer, F. Reker, and T. Bemmerl. Exploiting Transparent Remote Memory Access for Non-Contiguous- and One-Sided-Communication. In *Workshop on Communication Architecture for Clusters 2002 (in conjunction with IPDPS)*, April 2002.

[26] J. Wu, P. Wyckoff, and D. K. Panda. High Performance Implementation of MPI Derived Datatype Communication over InfiniBand . Technical Report, OSU-CISRC-10/03-TR58. http://nowlab.cis.o hio-state.edu/projects/mpi-iba/, Oct. 2003.

[27] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *the 2003 International Conference on Parallel Processing (ICPP 03)*, Oct. 2003.

[28] J. Wu, P. Wyckoff, and D. K. Panda. Supporting Efficient Noncontiguous Access in PVFS over InfiniBand. In *Proceedings of the IEEE International Conference on Cluster Computing*, 2003.