BluesMPI: Efficient MPI Non-blocking Alltoall Offloading Designs on Modern BlueField Smart NICs

Mohammadreza Bayatpour(✉), Nick Sarkauskas, Hari Subramoni, Jahanzeb Maqbool Hashmi, and Dhabaleswar K. Panda

The Ohio State University, Columbus, USA
{bayatpour.1,sarkauskas.1,subramoni.1,hashmi.29,panda.2}@osu.edu

Abstract. In the state-of-the-art production quality MPI (Message Passing Interface) libraries, communication progress is either performed by the main thread or a separate communication progress thread. Taking advantage of separate communication threads can lead to a higher overlap of communication and computation as well as reduced total application execution time. However, such an approach can also lead to contention for CPU resources leading to sub-par application performance as the application itself has less number of available cores for computation. Recently, Mellanox has introduced the BlueField series of adapters which combine the advanced capabilities of traditional ASIC based network adapters with an array of ARM processors. In this paper, we propose BluesMPI, a high performance MPI non-blocking Alltoall design that can be used to offload MPI_Ialltoall collective operations from the host CPU to the Smart NIC. BluesMPI guarantees the full overlap of communication and computation for Alltoall collective operations while providing on-par pure communication latency to CPU based on-loading designs. We explore several designs to achieve the best pure communication latency for MPI_Ialltoall. Our experiments show that BluesMPI can improve the total execution time of the OSU Micro Benchmark for MPI_Ialltoall and P3DFFT application up to 44% and 30%, respectively. To the best of our knowledge, this is the first design that efficiently takes advantage of modern BlueField Smart NICs in deriving the MPI Alltoall collective operation to get peak overlap of communication and computation.

Keywords: BlueField · SmartNIC · MPI · Alltoall · Offload

1 Introduction

The rapid growth in the scale of supercomputing systems over the last decade has been driven by the multi-/many-core architectures, and RDMA-enabled,

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

© Springer Nature Switzerland AG 2021
B. L. Chamberlain et al. (Eds.): ISC High Performance 2021, LNCS 12728, pp. 18–37, 2021.
https://doi.org/10.1007/978-3-030-78713-4_2

high-performance interconnects such as InfiniBand [8] (IB). The Message Passing Interface (MPI) [4] has been extensively used for implementing high-performance parallel applications and it offers various primitives such as point-to-point, collective, and Remote Memory Access operations. An MPI library that supports highly efficient communication primitives will be essential to the performance of HPC and parallel deep learning applications.

Overlap of communication and computation is critical for increasing resource utilization and performance. MPI provides non-blocking point-to-point and collective primitives that are used to achieve communication and computation overlap. In MPI, communication must be progressed, either by the main thread by calling MPI_Test or an extra offload entity such as a separate thread, or a hardware feature inside the network. If none of these exist, the amount of overlap will be limited as the main process thread must context switch from the application computation to progress the communication inside the MPI library. This also greatly depends on the application developer on how frequently they explicitly call MPI_Test. The application developer can either call MPI_Test or there may be an asynchronous communication thread in MPI. Both scenarios, however, can lead to sub-par performance as the main application has less CPU resources for useful application-level computation. Therefore, network offload mechanisms are gaining attraction as they have the potential to completely offload the communication of MPI primitives into the network, maximizing the overlap of communication and computation. However, the area of network offloading of MPI primitives is still nascent and cannot be used as a universal solution.

Table 1. Designs and features to support efficient non-blocking collectives in representative MPI libraries. C#1: computation and communication overlap, C#2: communication latency, challenge #3: network scalability, C#4: availability of cores for compute, C#5: hardware contexts for multiple communicators

	Features of representative MPI libraries							
	No offload	Core [16]-Direct	SHARP [15]	HW tag [6] matching	RDMA-aware [17]	MPICH [7] Async Thrd	MVAPICH2 Async [11]	Proposed
C#1	Poor	Good	Fair	Fair	Fair	Good	Fair	Good
C#2	Good	Good	Good	Good	Poor	Fair	Good	Good
C#3	Good	Fair	Good	Fair	Fair	Good	Good	Good
C#4	Poor	Good	Good	Fair	Good	Poor	Fair	Good
C#5	Good	Poor	Fair	Fair	Good	Good	Good	Good

Table 1 summarizes the different hardware offloading approaches. Among the most recent schemes in networking technologies, SHARP collective offload mechanism [15] only supports Barrier and Allreduce operations and it supports a few number of application level communicators as the Switch contexts are limited. Due to the limitation of SHARP contexts inside each switch, MPI libraries have to allow only one process per node (also known as the leader process) to use the

SHARP feature. Therefore, all the processes inside the same node must use host CPU resources to conduct the intra-node operations before using SHARP. This can limit the overlap opportunities of SHARP. Hardware Tag Matching for MPI point-to-point operations [5,6] is another state-of-the-art network offloading feature for MPI. Even though this mechanism can improve the overlap of communication and computation of large Rendezvous messages, when this point-to-point mechanism is used in dense collectives such as Alltoall, its overlap potential hugely degrades as the scale goes higher. This is due to a limited number of outstanding tags in this architecture [6]. On the other hand, in recent years, Smart NICs are able to bring more compute resources into the network and a high performance middleware such as MPI must take advantage of these additional resources to fill in the limitations of other in-network technologies. Smart NICs can act as a brand new host on the network by setting them to "separated host" mode. Therefore, instead of using them as a packet processing engine where all packets go through the processors inside the Smart NIC, these Smart NICs have the potential for any in-network offloading purpose.

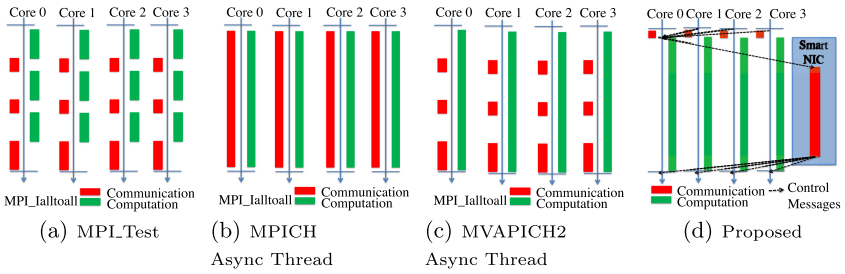


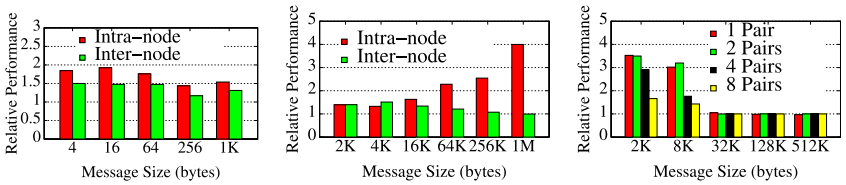
Fig. 1. Timeline of various designs for MPI non-blocking collectives

1.1 Challenges

In this paper, our goal is to efficiently take advantage of modern Smart NICs in separated host mode to propose novel MPI non-blocking Alltoall designs for large messages that 1) Achieves maximum overlap of communication and computation without requiring any changes inside the upper-level application, 2) Leaves the entire host processor for the useful application computation with minimal context-switching, and 3) Minimizes the overhead involved in offloading to the Smart NIC and provides good communication latency. In other words, we are envisioning communication offload, as outlined in Fig. 1. To achieve our goal, we are considering additional compute capabilities that are available in modern high performance interconnects, such as Smart NICs. One of the latest developments of such interconnects is the BlueField adapter that is based on the ConnectX-6 series of Infiniband Mellanox models. This adapter is equipped with an array of cache-coherent 1999 MHz ARM cores and they can be used as a general-purpose system [3]. It also provides support for dual-port Remote Direct

Memory Access (RDMA). These developments lead to the following broad challenge: **How can existing production-quality HPC middleware such as MPI be enhanced to take advantage of emerging networking technologies to deliver the best performance for HPC and DL applications on emerging dense many-core CPU/GPU systems?**

We break down this broad challenge into the following questions: **1)** What shortcomings regarding MPI exist in current state-of-the-art in-network technologies? **2)** Can we use additional compute resources provided by modern Smart NICs to accelerate MPI collective primitives? **3)** What are the challenges regarding exploiting these modern Smart NICs for offloading non-blocking Alltoall operations? **4)** Can we propose efficient designs to take advantage of Smart NICs capabilities without requiring the upper-level application changes? **5)** How to minimize pure communication latency of non-blocking Alltoall collective operations designed using BlueField Smart NICs?, and **6)** What are the performance overheads of each component of the framework and what is the impact of the proposed design at the microbenchmark level as well as the application-level?



(a) Small Msg Point-to-point Latency (b) Large Msg Point-to-point Latency (c) Inter-node Multi-Pair Bandwidth

Fig. 2. Comparison of point-to-point latency and bandwidth of the processes on BlueField smart NIC versus processes on the host. Latency Relative Performance (speedup) is calculated by BF-latency/Host-latency, and bandwidth speedup is calculated by Host-bw/BF-bw. We can observe that as message size increases, the inter-node performance of ARM cores of BlueField (BF) smart NIC converges to the performance of XEON cores.

1.2 Motivation and Characterization

As an initial step to answering our broad challenge, we need to identify opportunities provided by Smart NICs and thoroughly characterize a system enabled with BlueField adapters. Based on this characterization, we conclude which MPI operations have the potential for offloading to the ARM cores of the BlueField and provide insights for our proposed Smart NIC-aware MPI library. To do so, we compare the latency and bandwidth of communication between MPI processes on the host cores versus MPI processes running on the ARM cores of the BlueField adapters using OSU Micro Benchmarks [1]. Please refer to Sect. 1.5 for detailed experimental setup information. For each test, we launch all the processes on the XEON cores of hosts and measure the latency and bandwidth.

Then, we perform a similar test by launching all the processes on the ARM cores of the BlueField Smart NIC and calculate the speedup of host tests versus Smart NIC tests. The speedup is calculated by ARM-latency/host-latency.

Figures 2(a) and (b) shows that for intra-node operations, as the message size increases, the performance of intra-node operations diverges from the host processes. This is in line with our expectations as for the intra-node operations, CPU is in charge of the copy operations, and having a faster CPU has a significant impact on point-to-point performance. Therefore, in our BluesMPI framework, we avoid going through the CPU based intra-node operations for BluesMPI worker processes on the Smart NIC. On the other hand, Fig. 2(b) shows an opposite trend. Here **as the message size increases, inter-node latency of Smart NIC worker processes and host processes converge.** This is because the HCA is in charge of operations and for medium and large messages (large than 16 KB) where the rendezvous protocol is used for point-to-point operations. In this protocol, there are no copy operations involved. Therefore, as message size increases, the network overheads will have more share of the total latency.

Figure 2(c) illustrates the bandwidth comparison between the process running on ARM core of Smart NIC and the processes running on the host XEON cores. The speedup is calculated by host-BW/ARM-BW. For multiple pair bandwidth tests, we used osu_mbw_mr [1] that calculates the aggregate bandwidth for multiple pairs of processes. Here experimental results of the inter-node operation are shown as intra-node operations are not interesting for us anymore. These results show a similar trend as for large message inter-node latency operations. Here also as the message size increases, the performance of processes on ARM cores of Smart NIC converges to the performance of XEON cores of host. This trend is consistent as the number of pairs increases as well. **This shows that processes on Smart NIC have the potential to handle dense communication for large messages using RDMA.**

Based on this characterization, our proposed BluesMPI framework is purely based on RDMA operations and the focus is to provide maximum overlap of communication and computation with low communication latency for dense non-blocking Alltoall collectives with medium and large messages.

1.3 Contributions

In this paper, we characterize various MPI point-to-point operations and identify the aspects of the MPI library that can be efficiently driven by the additional compute resources on the modern Smart NICs. Then based on our characterization, we propose BluesMPI, an adaptive MPI non-blocking Alltoall collective offload design on modern Smart NICs. We propose various designs on the top of BlueField for MPI_Ialltoall operations. Our experimental results show that BluesMPI can successfully take advantage of the available Smart NICs SoC on the network and lower the execution time of OSU Micro Benchmark by 44% and P3DFFT application by 30% on 1024 processes. To the best of our knowledge, this is the first design that efficiently takes advantage of modern BlueField

Smart NICs in deriving the MPI collective operations to get the peak overlap of communication and computation.

To summarize, this paper makes the following contributions:

- In-depth analysis and characterization of MPI operations running on the available compute resources of Smart NICs.
- Proposing novel designs for non-blocking Alltoall operations that provide full overlap of communication and computation and low pure communication latency.
- Performing a thorough characterization of different components of the proposed BluesMPI framework.
- Performance evaluations of the proposed designs at the micro benchmark level and application level.

1.4 Overview of BlueField Smart NICs

Within each of the products in the BlueField family is the BlueField Data Processing Unit (DPU). This is a system-on-chip containing 64-bit ARMv8 A72 cores connected in a mesh, DDR4 memory controllers, a ConnectX network controller, and an integrated PCIe switch. The DPU is sold as part of products in different lines of the BlueField family. These include BlueField Smart NICs, BlueField storage controllers, and the BlueField Reference Platform. Figure 3 depicts a schematic overview of the BlueField Smart NIC architecture. The BlueField Smart NIC has two modes of operation: Separated Host mode (default) and Embedded CPU Function Ownership (Smart NIC) mode. Each physical port on the Smart NIC can be independently configured to either mode [3]. In separated host mode, the ARM cores can appear on the network as any other host and the main CPU (i.e. ×86) is exposed through a PCIe function for direct connectivity to the ConnectX. The ARM cores are exposed through a symmetric (to the host) PCIe function for their own connectivity to the ConnectX network adapter. Bandwidth is shared between the two functions. In our experiments, we use this mode. Embedded CPU Function Ownership (Smart NIC) mode places several restrictions on the host. In Smart NIC mode, all network controller resources

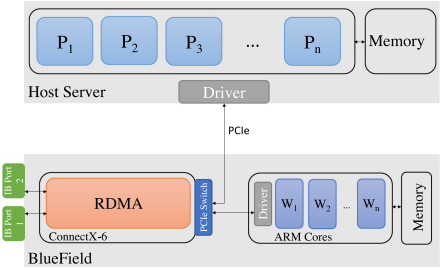


Fig. 3. BlueField smart NIC architecture

are controlled by the ARM cores via the Embedded CPU Physical Function (ECPF). The ECPF in this mode will own the embedded switch (e-switch) as well. In order to pass traffic to the host, either the e-switch must be set up with forwarding rules, or kernel netdev representors (Open vSwitch virtual ports) must be configured on the ARM cores [3].

1.5 Experimental Setup

We used the HPC Advisory Council High-Performance Center (HPCAC) [2] cluster for our evaluation. HPCAC has 32 nodes that contain the BlueField-2 network adapters. These adapters have an array of 8 ARM cores operating at 1999 MHz with 16 GB RAM. Each BlueField adapter is equipped with Mellanox MT41686 HDR ConnectX-6 HCAs (100 Gbps data rate) with PCI-Ex Gen3 interfaces [3]. The host is equipped with the Broadwell series of Xeon dual-socket, 16-core processors operating at 2.60 GHz with 128 GB RAM.

2 BluesMPI Designs

In this section, we provide the details of various components of the proposed BluesMPI framework. In Sect. 2.1, we discuss the overall design of the framework and explain each step that is required for non-blocking Alltoall collective operations to be offloaded onto the Smart NIC. In Sect. 2.2, we describe the details of various novel designs for non-blocking Alltoall operations.

2.1 BluesMPI Non-blocking Alltoall Collective Offload Framework

In BluesMPI, non-blocking Alltoall collective operations are offloaded to a set of the Worker processes which have been spawned in the MPI_Init to the Smart NICs that are in the separated host mode. Therefore, all that application's host processes have to do is to prepare a set of metadata and provide it to the Worker processes. Once the collective operations are completed, Worker processes notify the host processes. BluesMPI framework goes over a set of steps in order to prepare the non-blocking Alltoall collective operations to be offloaded to the Worker processes on the Smart NIC. Although these steps are described for nonblocking Alltoall, a similar framework can be used for any other dense collective communication, with a few modifications. For instance, for Allgather, some of the steps can be done in the host shared memory to avoid excessive IB link utilization.

Step 0) *Buffers Registration with HCA:* In the first step, all the processes inside the host communicator need to register the send buffer and receive buffer of the MPI collective call with HCA, so that remote processes are able to perform RDMA Read and Write on these buffers, asynchronously. Memory registration is a costly operation, therefore, in our designs, we take advantage of a registration cache to avoid re-registering the same set of buffers more than once.

Step 1) *Metadata Aggregation to the Host Communicator Leader Process:* Once a process in the host communicator registers its send and receive buffers, it creates a collective info object that includes RDMA buffer addresses and keys. It also includes this process's rank in MPI_COMM_WORLD as well as the count and datatype of this collective call. This information is the Metadata for the collective call from this host process. The host communicator leader (which is rank 0 in our design) gathers the Metadata from all the processes in the communicator.

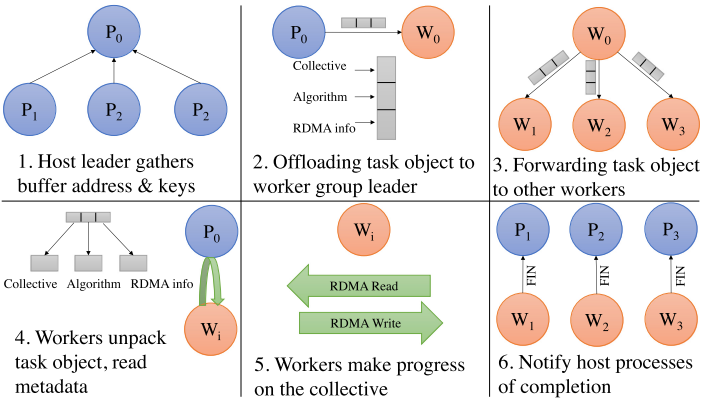


Fig. 4. BluesMPI procedure to offload non-blocking Alltoall collective operation to the worker processes on the Smart NIC. Step 0 is not included in this figure.

Step 2) *Metadata Registration with HCA and Offloading the Task Object to Leader of the Workers Group:* Once the host communicator leader generates the array of Metadata, it has to register this array with HCA so that all the Worker process on the Smart NIC can read whatever information that they require at any time during progressing the collective. Once the registration is done, the host communicator leader creates a new task object and sends it to the Workers group leader. This task object has the information about the type of the collective and the algorithm which must be performed by the Worker processes on the Smart NIC. It also has the RDMA information of the Metadata array and the host communicator size.

From now, the host processes are free to perform useful application computation. In the meantime, the leader of the Worker group on Smart NIC waits for the incoming task objects from the leaders of the host communicators. Since the application could have several sub-communicators, the leader of the Workers group on Smart NIC can receive several task objects at the same time. It is also possible that even for a single host communicator, several back-to-back nonblocking collective calls are issued before going into the MPI_Wait. In order

to handle all these scenarios, the leader of the Worker group on the Smart NIC creates a FIFO queue and pushes all the new task objects into this queue.

Step 3) *Picking up a Task from Queue of Offloaded Tasks and Forward it to the Non-leader Workers:* The leader of the Smart NIC Worker group picks a task from the head of the tasks queue and broadcasts this object task to all the processes in the Workers group.

Step 4 and 5) *Progress the Collective on Behalf of the Host Communicators:* Once every Worker process on the Smart NIC receives a task object, it unpacks the object and based on the task type, it performs the appropriate operations on it. Now every Worker process needs to read the Metadata of the collective from the host memory. In the following Sect. 2.2, we discuss the algorithm that we used for nonblocking Alltoall performed by Worker processes.

Step 6) *Collective Completion Notification:* Once each receive buffer of the host communicator processes has the correct value which is written by the Worker processes on the Smart NIC, a completion notification is sent to the host processes.

Figure 4 summarizes the required steps in the BluesMPI non-blocking Alltoall collective offload framework.

2.2 Proposed Nonblocking Alltoall Designs in BluesMPI

In this section, we discuss our proposed designs to perform the nonblocking Alltoall operations by the Worker processes on the Smart NIC. In these designs, we consider balanced Workers per node, meaning that the number of the Workers per node is the same between all the nodes. As the first step to perform the nonblocking Alltoall, Worker processes must receive a task object regarding this operation. This is done by the steps performed by the BluesMPI framework discussed in Sect. 2.1. Once each Worker process has access to this task object and its Metadata, it has full read and write access to every buffer of every process in the host communicator.

In a perfect scenario, it is expected that the Worker processes issue RDMA read and write operations to HCA on behalf of the host communicator processes. This is because once the non-blocking collective is issued by the host process, this process starts working on the application computations and it is not inside MPI, progressing the communication. Therefore, in order to have a complete overlap of communication and computation for the non-blocking collective operation, and assuming that there is no extra communication progress thread running on the host CPU, Worker processes should be able to progress the HCA on behalf of the host processes. However, modern interconnects do not have this support. This means that even if a remote Worker process on the Smart NIC has the RDMA address and key of a local memory of host processes, it cannot directly issue RDMA read or write from the host local memory to the destination memory of another host process. Therefore, in our proposed non-blocking Alltoall designs, data is staged in the main memory of the Smart NIC, and then it is forwarded to the destination. Figure 5(a) depicts a single transfer in our proposed designs. Scatter destination algorithm works best for medium and large messages [11],

thus our proposed designs are based on this algorithm. In the scatter destination algorithm for Alltoall, there is a loop with communicator-size iterations and in each iteration, an exclusive piece of send buffer is sent to the destination receive buffer of the remote process.

Once the Worker processes running on the ARM cores of the Smart NIC have the collective Metadata, they share the collective progression among themselves in a balanced manner. Therefore, if there are PPN number of the processes of the host communicator in the same node and there are WPN number of Worker processes per node, each Worker process is responsible for the PPN/WPN number of the host processes. Depending on how Worker processes on the Smart NIC take advantage of the staging based message transfer mechanism depicted in Fig. 5(a), we explore three designs: 1) Direct Design, 2) Message Chunking Design, and 3) Message Pipelining Design. All of these designs in nature are scatter destination Alltoall designs.

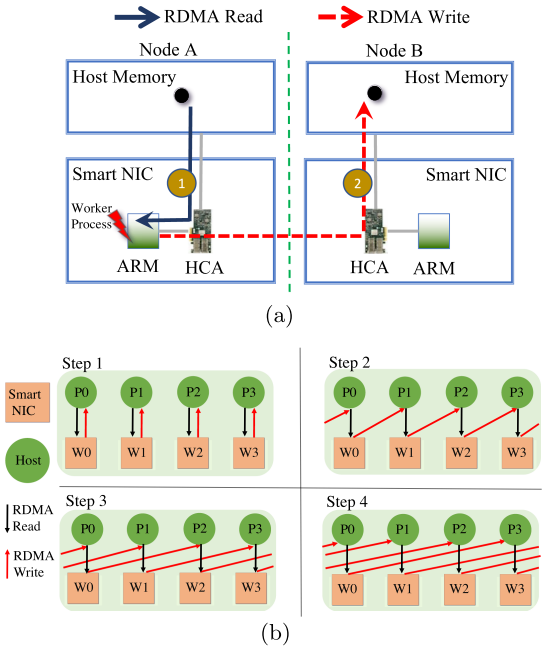


Fig. 5. (a) A single message transfer from a host communicator process in node A to another host process on node B in our proposed Alltoall designs. (b) The Proposed Direct Design in BluesMPI for Ialltoall for 1 PPN, 1 WPN, and 4 nodes scenario.

Direct Design. In this design, each Worker process starts from the first host process which is assigned to it and then delivers each exclusive piece of data from the local memory of the host process to all other host processes in the host

communicator. If there are N processes in the host communicator, since we are performing an Alltoall operation, the send buffer of each host process will have N exclusive data each with a size that depends on the count and datatype inputs of MPI_Ialltoall. Each of these N elements is sent to the appropriate index of the receive buffer of another process in the host communicator. Therefore, if a Worker process is responsible for the H number of host processes, it has to perform N × H number of message transfers on behalf of those H number of host processes that offloaded their collective communication on this Worker process. Each of these individual back-to-back staging based transfers uses the mechanism illustrated in Fig. 5(a) in an asynchronous manner while the host process is performing the application compute and it is outside of the MPI library. Figure 6 shows the Direct Design for the first four message transfers of a Worker process. To further optimize this algorithm, we propose a link efficient load-balanced staging technique. To achieve load balancing, in this design, we need to make sure that at any point during the Direct Design, only one Worker process is writing to receive buffer of a host process. Therefore, instead of allowing each Worker process to start writing to the destination processes with rank 0, each Worker process sets its initial destination process to the same host process that is assigned to it. An example is provided in Fig. 5(b). Figure 7(a) shows that we can achieve 38% in pure communication latency by taking advantage of this link load-balancing mechanism.

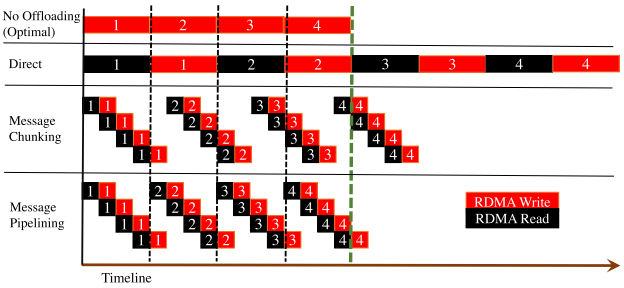
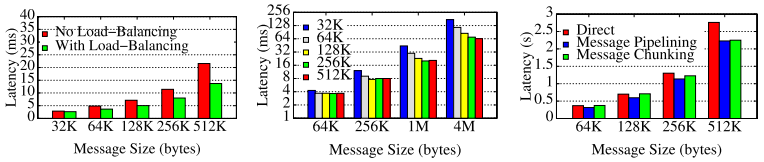


Fig. 6. Timeline of the proposed staging based Alltoall designs for large messages. For No Offloading scenario, scatter destination algorithm used in blocking Alltoall is considered.

Message Chunking Design. One of the major bottlenecks in the Direct Design is that it suffers from the overheads of the message staging in the Smart NIC. This is because due to the staging operation, the number of the RDMA operations doubles compared to the No Offloading CPU driven scatter destination scenario. Although in the Direct Design, there is a full overlap of communication and computation, still, in order to get noticeable benefit in total application time, we need to further reduce the pure communication time of Direct Design. In order to do so, in Message Chunking Design, we break down



(a) 8 Nodes, 4 PPN, 1 WPN (b) 8 Nodes, 4 PPN, 1 WPN (c) 32 Nodes, 32 PPN, 4 WPN

Fig. 7. (a) Impact of load-balancing, (b) Impact of chunking, (c) Performance comparison of different proposed designs. In these figures, pure communication latency of MPI_Ialltoall is reported.

a single message size of *msg_size* to multiple chunks. Then in each iteration, we try to overlap the RDMA write of the current chunk with the RDMA read of the next chunk. Figure 6 illustrates the Message Chunking Design. Infiniband links are bi-directional, therefore, RDMA Write and Read can happen at the same time without any extra cost. The base of this algorithm is indeed the Direct Design, however, in the Message Chunking Design, we replace each staging based transfer of size *msg_size* with an another primitive that chunks the message to *chunk_size* equivalent pieces and overlaps the RDMA read and writes of back to back chunks for this specific message. In this design, chunk size plays a major role in the pure communication performance. Figure 7(b) shows the impact of the chunk size compared to Direct Design. All of our experiments are conducted on the HPCAC cluster which is introduced in Sect. 1.5.

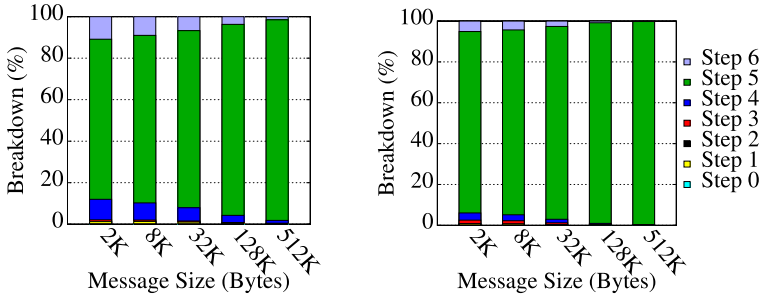
Message Pipelining Design. Message Chunking Design is able to further reduce the impact of the staging to Smart NIC. However, due to the nature of this design that it considers each message transfer in an isolated manner, there are still multiple chunks of the messages that are not taking advantage of the overlapping between RDMA read and write. This is due to the fact that for each message transfer, the first RDMA read and last RDMA write are not getting overlapped with any other operations. This is also depicted in Fig. 6. Although by increasing the number of the chunks, we can reduce this impact, but on the other hand, choosing too small chunks can have a negative impact on IB links as they are able to fill up the bandwidth and get the best performance. In order to reduce the number of chunks which have not been overlapped, in Message Pipelining Design, we take advantage of pipelining the back to back transfers. In this design, RDMA write from Smart NIC to host memory of the current message transfer of size *msg_size* is overlapped with RDMA read of the next message from host memory to Smart NIC. In this design, there will be only two messages which have not been overlapped: the RDMA read of the first message transfer and the RDMA write of the last message transfer. As the communicator size N increases, the negative impact of staging to Smart NIC also decreases, as the total number of transfers increases by a factor of N while the number of messages which have not been overlapped remains 2. On a small scale and

Algorithm 1: Message Pipelining Design (Design-3)

```
   Input  : rdma_info — Array of Send/Recv/FIN Buffers RDMA Info
   Input  : world_ranks — Array of Host processes ranks in MPI_COMM_WORLD
   Input  : host_comm_size — Host communicator size
   Input  : worker_comm_size — Workers communicator size
   Input  : worker_rank — Rank of this worker process in Workers communicator
   Input  : count — Count
   Input  : datatype_size — Datatype size of a single element
   Input  : chunk_size — Chunk size to be used for data staging procedure
   Output: mpi_errno
 1 begin
 2     customers_list_size =
 3     host_comm_size / worker_comm_size
 4     for i ← 0 to customers_list_size do
 5         Find the host processes that this worker is responsible for
 6         customers_list[i] =
 7         customers_list_size × worker_rank + i
 8     end
 9     chunk_num = msg_size / chunk_size
10     total_msgs = host_comm_size × customers_list_size × chunk_num
11     for msg ← −1 to total_msgs do
12         Prepare for RDMA Read for a single chunk
13         i = (msg +1) / chunk_num
14         if msg = total_msgs − 1 then
15             Skip RDMA Read, set read completion flag for this chunk and jump to
               skip_read
16         end
17         src_rank = customers_list[i / host_comm_size ]
18         src_world_rank = world_ranks[src_rank]
19         sendbuf = rdma_info[src_rank].sendbuf.buf_addr
20         src_key = rdma_info[src_rank].sendbuf.rkey
21         dst_rank = ((i % host_comm_size) + src_rank) % host_comm_size
22         src_buf = sendbuf + dst_rank × msg_size + chunk_size × ((msg + 1) %
           chunk_num)
23         Initiate an RDMA Read for a single chunk
24         staging_tmp_buf_read = staging_tmp_buf + msg_size × ((msg + 1) %2)
25         NonBlockingRdmaRead( staging_tmp_buf_read,
26         src_key, src_buf, src_world_rank, chunk_size )
27         Prepare for RDMA Write for a single chunk
28         skip_read:
29         i = msg / chunk_num
30         if msg = −1 then
31             Skip RDMA Write, set write completion flag for this chunk and jump to
               skip_write
32         end
33         src_rank = customers_list[i / host_comm_size ]
34         dst_rank = ((i % host_comm_size) + src_rank) % host_comm_size
35         dst_world_rank = world_ranks[dst_rank]
36         recvbuf = rdma_info[dst_rank].recvbuf.buf_addr
37         dst_key = rdma_info[dst_rank].sendbuf.rkey
38         dst_buf = recvbuf + src_rank × msg_size + chunk_size × (msg %
           chunk_num)
39         staging_tmp_buf_write = staging_tmp_buf + msg_size × (msg %2)
40         NonBlockingRdmaWrite( staging_tmp_buf_write,
41         dst_key, dst_buf, dst_world_rank, chunk_size )
42         skip_write:
43         BlockingWaitRdmaReadWrite()
44     end
45     Barrier(worker_comm)
46     Notify all the host processes in the customers_list
47 end
```

especially for large messages, this design is combined with Message Chunking Design. Therefore, each message is chunked into multiple pieces, and the RDMA write of the last chunk of each message is overlapped with the RDMA read of the first chunk of the next message. Algorithm 1 provides further details about the procedure that worker processes perform to implement this algorithm. Figure 6 compare the pipelining opportunities of all the Direct, Message Chunking, and Message Pipelining designs. Figure 7(c) compares their performance against each other.

Once each Worker process is done with the task assigned to it, it goes into a barrier, and it waits for all other Worker processes in the same group to finish their task. Once every Worker process is done, they notify thehost processes which are assigned to them. They do so by issuing an RDMA write to the FIN flag on the local memory of each host process which was provided to Worker processes. After this step, each Worker process goes into a broadcast operation, and they wait for the leader of the Workers group to assign them a new collective offloading task.



(a) 4 PPN, 1 WPN (b) 16 PPN, 4 WPN

Fig. 8. Performance breakdown of pure communication latency of MPI_Ialltoall directly followed by MPI_Wait for different steps of the BluesMPI framework discussed in Sect. 2.1. These tests run on 8 nodes using Message Pipelining design.

3 Results

In this section, we discuss the experimental analysis of MPI collective primitives using OSU Micro Benchmarks [1] and a modified P3DFFT [12] application with nonblocking Alltoall support that is proposed by Kandalla et al. [9]. We provide a performance breakdown of different steps of BluesMPI framework. BluesMPI is designed on the top of the MVAPICH2 v2.3 MPI library. Comparisons with HPCX 2.7.0 with HCOLL NBC flag enabled, MVAPICH2-X v2.3 with MPICH asynchronous thread enabled, as well as optimized asynchronous thread enabled are also provided. All the reported numbers are an average of three runs and micro-benchmark evaluations ran for 1,000 iterations for each message size and

an average of three experiments is reported. The standard deviation between these iterations is kept under 2%.

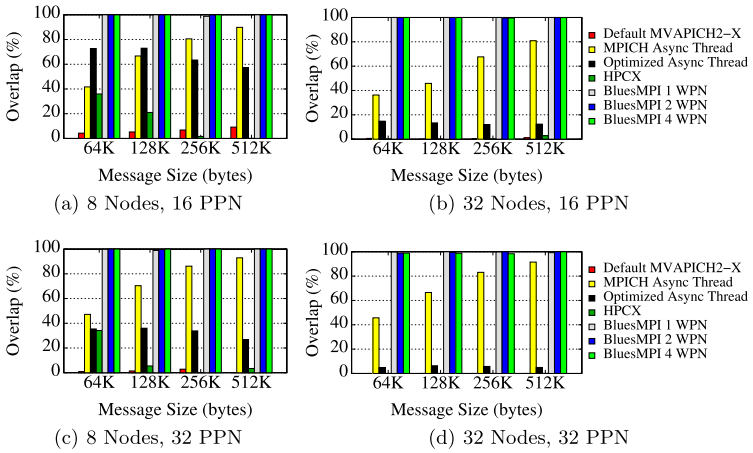


Fig. 9. Overlap of communication and computation reported by osu_ialltoall benchmark for various designs.

3.1 Performance Characterization of BluesMPI Framework

In this section, we conduct a performance characterization of different steps of the BluesMPI framework which are introduced in Sect. 2.1. To do so, light-weight timers are added inside the BluesMPI framework and the time taken for each of the six steps of the framework is measured. Figure 8 shows this performance breakdown of pure communication of MPI_alltoall for two tests with 8 nodes. As we can see here, for smaller message sizes, the overheads of BluesMPI are more visible compared to larger messages. This is because the overheads of BluesMPI, which are the steps of 1 to 4 and step 6 (considering step 5 as the useful collective time) are not dependant on the message size and they only depend on the Workers group size and host communicator size. This means that if the Workers group size and host communicator size do not change, the overhead remains constant, regardless of the message size. Therefore, only step 5 is dependant on the job size and message size of MPI_Ialltoall. Figure 8 shows the same trend. For a single job size, as the message size increases, step 5 latency increases, and since other steps remain constant, the percentage overhead compared to step 5 decrease. After step 5, steps 4 and 6 have the highest overhead compared to other steps. This is because these two steps run on the slower ARM cores of the BlueField and therefore, compared to host-related overhead (steps 0, 1, and 2), they are more signified.

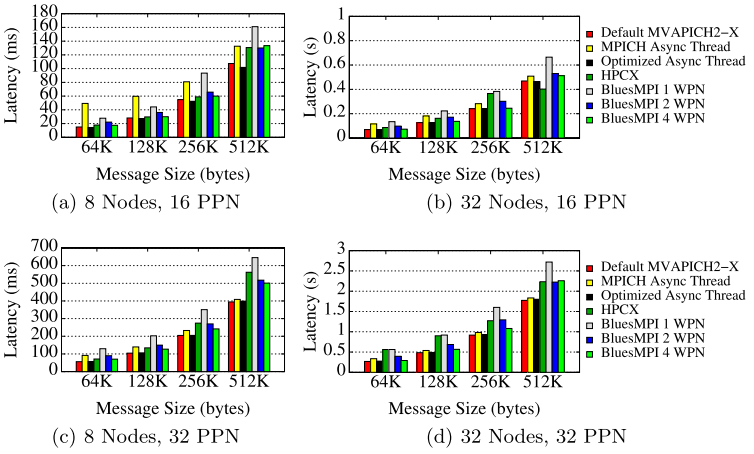


Fig. 10. Pure communication time of MPI_Ialltoall (time of MPI_Ialltoall followed by MPI_Wait) for various designs.

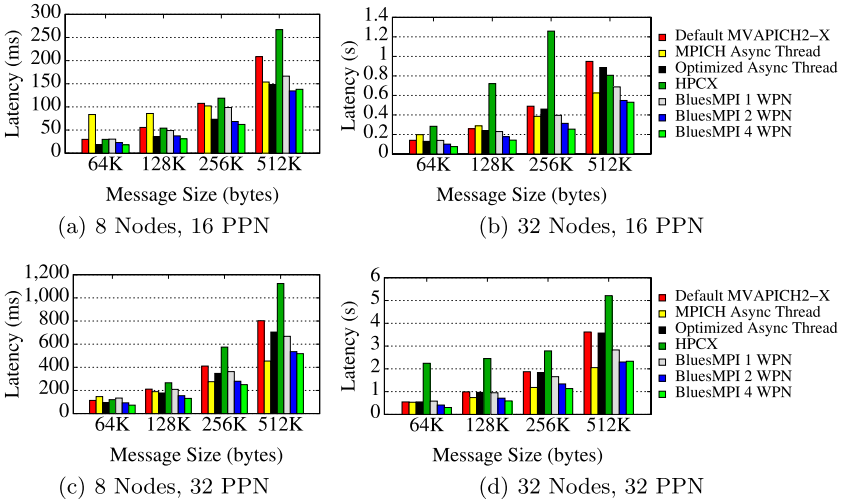
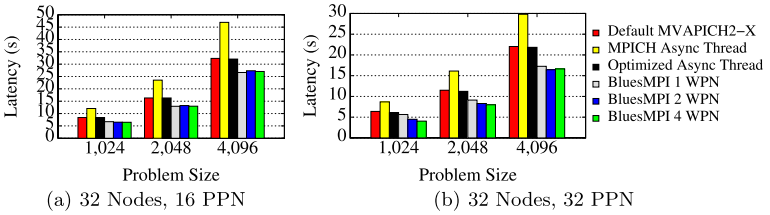


Fig. 11. Total execution time of osu_ialltoall benchmark for various designs.

3.2 Performance of MPI Collective Operations

In this section, we compare the performance of MPI_Ialltoall using the osu_ialltoall benchmark from the OSU Micro Benchmark suite. Figures 9, 10, and 11 show the impact of our proposed BluesMPI collective offloading framework on the InfiniBand based BlueField Smart NICs. For these tests, we used our most optimized algorithm which is Message Pipelining Design discussed in Sect. 2.2. As we can see here, our proposed design can guarantee the peak

communication and computation overlap, as indicated in Fig. 9. On the other hand, BluesMPI high-performance staging based nonblocking alltoall design, with the proper number of Workers per node, it can gain on-par pure communication performance with tuned non-offloaded designs for large messages. By providing the peak communication and computation overlap and achieving low pure communication latency, BluesMPI can gain up to 2X speedup in the total osu_ialltoall execution time compared to default MVAPICH2-X. Comparing to the HPCX 2.7.0 with HCOLL NBC flag enabled, we can see that the proposed design's pure communication performance is on-par with this library. However, as the proposed design can provide full overlap of communication and computation, the total execution time improves up to 2X. The closest in performance of osu_ialltoall is MVAPICH2-X with MPICH asynchronous thread enabled. However, as we will see in the next section, having a separate thread for each process running constantly can severely degrade the performance. On the other hand, our proposed design does not interfere with the main application's compute, and therefore, can provide full overlap of communication and computation in a transparent manner, showing its benefits at the application level.



(a) 32 Nodes, 16 PPN (b) 32 Nodes, 32 PPN

Fig. 12. CPU time per loop of P3DFFT application for various designs.

3.3 Application Evaluations

In this section, we evaluate the impact of the BluesMPI framework on performance of Parallel Three-Dimensional Fast Fourier Transforms (P3DFFT) application. This library uses a 2D, or pencil, decomposition and increases the degree of parallelism and scalability of FFT libraries. The data grid during each iteration is transformed using nonblocking Alltoall collectives [9]. Figure 12 shows the impact of the proposed BluesMPI designs with various number of Workers per node and various scales. For these tests, we used the Message Pipelining Design discussed in Sect. 2.2. The program that we used is test_sine.x and we set x and y grids to 2048. On x-axis, we run the tests for different values of z. As we can see here, as the scale of the application increases, the benefits of the BluesMPI also become more visible, gaining up to 30% improvement in the execution time of this application at 32 PPN 32 Nodes of the BlueField-enabled thor nodes of HPCAC cluster. It can be seen from this figure that even having a single Worker

on each Smart NIC is having benefit. This is because even with a single Worker per node BluesMPI can achieve close to full overlap and communication and computation and if an application can provide enough computation to be overlapped with the communication time of the collective, it can see benefit with a single Worker per Smart NIC as well. On the other hand, MVAPICH2-X with MPICH asynchronous thread is showing the worst performance. This is because this thread is constantly running and it interferes with the main application's compute resources.

4 Related Work

There have been some recent research efforts that offload networking functions onto FPGA-based SmartNICs. There are also studies on offloading tasks to SmartNICs in distributed applications. Floem [13] proposed a dataflow programming system aimed at easing the programming effort. Liu et al. [10] built an "actor" based prototype (called ipipe) and developed several applications using it. The evaluation showed that by offloading computation to a SmartNIC, considerable host CPU and latency savings is achievable. Researchers have also explored various ways of offloading the progression of communication to NICs for MPI point-to-point and collective operations. Sur et al. [18] discuss different mechanisms for better computation/communication overlap on InfiniBand clusters. These mechanisms exploit RDMA Read and selective interrupt-based asynchronous progress and achieves nearly complete computation/communication overlap. Potluri et al. [14] studied novel proxy-based designs to optimize the internode point to-point and collective MPI primitives for Intel Xeon Phi based cluster systems connected using InfiniBand network.

5 Conclusion and Future Work

In this paper, we characterized the performance impact of the smart NICs on MPI and we found out the potential MPI primitives that can be offloaded into the Smart NICs. Based on our observations, we proposed BluesMPI, an adaptive non-blocking Alltoall collective offload framework that can be used on modern Smart NICs. Furthermore, we proposed efficient offloading designs for non-blocking Alltoall operations on the top of the BlueField Smart NIC. Our experimental evaluations showed that using the proposed methods, we are able to efficiently take advantage of the additional compute resource of Smart NICs in the network and accelerate the performance of OSU Micro Benchmarks and P3DFFT by a factor of 44% and 30%, respectively. To the best of our knowledge, this is the first design that efficiently takes advantage of modern BlueField Smart NICs in deriving the MPI collective operations to get the peak overlap of communication and computation. Our future work is to provide similar designs for other dense collective operations as well.

References

1. http://mvapich.cse.ohio-state.edu/benchmarks
2. High-Performance Center Overview. https://www.hpcadvisorycouncil.com/cluster_center.php
3. Mellanox BlueField. https://docs.mellanox.com/x/iQO3
4. Panda, D.K., Subramoni, H., Chu, C.H., Bayatpour, M.: The MVAPICH project: Transforming research into high-performance MPI library for HPC community. J. Comput. Sci. 101208 (2020)
5. Bayatpour, M., et al.: Communication-aware hardware-assisted MPI overlap engine. In: Sadayappan, P., Chamberlain, B.L., Juckeland, G., Ltaief, H. (eds.) ISC High Performance 2020. LNCS, vol. 12151, pp. 517–535. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50743-5_26
6. Bayatpour, M., Ghazimirsaeed, S.M., Xu, S., Subramoni, H., Panda, D.K.: Design and characterization of infiniband hardware tag matching in MPI. In: 20th Annual IEEE/ACM CCGRID (2020)
7. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI, message passing interface standard. Technical report, Argonne National Laboratory and Mississippi State University
8. InfiniBand Trade Association (2017). http://www.infinibandta.com
9. Kandalla, K., Subramoni, H., Tomko, K., Pekurovsky, D., Sur, S., Panda, D.K.: High-performance and scalable non-blocking all-to-all with collective offload on infiniband clusters: a study with parallel 3D FFT. Comput. Sci. **26**, 237–246 (2011)
10. Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S., Gupta, K.: iPipe: a framework for building distributed applications on SmartNICs. In: SIGCOMM 2019: Proceedings of the ACM Special Interest Group on Data Communication, pp. 318–333 (2019). https://doi.org/10.1145/3341302.3342079
11. Network-Based Computing Laboratory: MVAPICH2-X (Unified MPI+PGAS Communication Runtime over OpenFabrics/Gen2 for Exascale Systems). http://mvapich.cse.ohio-state.edu/overview/mvapich2x/
12. Pekurovsky, D.: P3DFFT library (2006–2009). www.sdsc.edu/us/resources/p3dfft/
13. Phothilimthana, P.M., Liu, M., Kaufmann, A., Simon Peter, R.B., Anderson, T.: Floem: a programming system for NIC-accelerated network applications. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). 13th USENIX Symposium on Operating Systems Design and Implementation (2018)
14. Potluri, S., et al.: MVAPICH-PRISM: a proxy-based communication framework using InfiniBand and SCIF for Intel MIC clusters. In: Proceedings of SC 2013, SC 2013, pp. 54:1–54:11 (2013)
15. Scalable hierarchical aggregation protocol: scalable hierarchical aggregation protocol. https://www.mellanox.com/products/sharp
16. Subramoni, H., Kandalla, K., Sur, S., Panda, D.K.: Design and evaluation of generalized collective communication primitives with overlap using ConnectX-2 offload engine. In: Internationall Symposium on Hot Interconnects (HotI), August 2010 (2010)

17. Subramoni, H., et al.: Designing non-blocking personalized collectives with near perfect overlap for RDMA-enabled clusters. In: Kunkel, J.M., Ludwig, T. (eds.) ISC High Performance 2015. LNCS, vol. 9137, pp. 434–453. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20119-1_31
18. Sur, S., Jin, H.W., Chai, L., Panda, D.K.: RDMA read based rendezvous protocol for MPI over infiniband: design alternatives and benefits. In: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2006 (2006)