Comput Sci Res Dev (2011) 26: 237–246
DOI 10.1007/s00450-011-0170-4

SPECIAL ISSUE PAPER

High-performance and scalable non-blocking all-to-all with collective offload on InfiniBand clusters: a study with parallel 3D FFT

Krishna Kandalla · Hari Subramoni · Karen Tomko · Dmitry Pekurovsky · Sayantan Sur · Dhabaleswar K. Panda

Published online: 13 April 2011

Abstract Three-dimensional FFT is an important component of many scientific computing applications ranging from fluid dynamics, to astrophysics and molecular dynamics. P3DFFT is a widely used three-dimensional FFT package. It uses the Message Passing Interface (MPI) programming model. The performance and scalability of parallel 3D FFT is limited by the time spent in the Alltoall Personalized exchange (MPI_Alltoall) operations. Hiding the latency of the MPI_Alltoall operation is critical towards scaling P3DFFT. The newest revision of MPI, MPI-3, is widely expected to provide support for non-blocking collective communication to enable latency-hiding. The latest InfiniBand adapter from Mellanox, ConnectX-2, enables offloading of generalized lists of communication operations to the network interface. Such an interface can be leveraged to design non-blocking collective operations. In this paper, we design a scalable, non-blocking Alltoall Personalized Exchange algorithm based on the network offload technology. To the best of our knowledge, this is the first paper to propose high performance non-blocking algorithms for dense collective operations, by leveraging InfiniBand's network offload features. We also re-design the P3DFFT library and a sample application kernel to overlap the Alltoall operations with application-level computation. We are able to scale our implementation of the non-blocking Alltoall operation to more than 512 processes and we achieve near perfect computation/communication overlap (99%). We also see an improvement of about 23% in the overall run-time of our modified P3DFFT when compared to the default-blocking version and an improvement of about 17% when compared to the host-based non-blocking Alltoall schemes.

Keywords Non-blocking collective communication · InfiniBand network offload · 3DFFT · Alltoall personalized exchange · Message passing interface (MPI)

This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CCF-0833169, #CCF-0916302, #OCI-0926691 and #CCF-0937842; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsytems.

K. Kandalla (✉) · H. Subramoni · S. Sur · D.K. Panda
The Ohio State University, Columbus, OH, USA
e-mail: kandalla@cse.ohio-state.edu

H. Subramoni
e-mail: subramon@cse.ohio-state.edu

S. Sur
e-mail: surs@cse.ohio-state.edu

D.K. Panda
e-mail: panda@cse.ohio-state.edu

K. Tomko
The Ohio Supercomputer Center, Columbus, OH, USA
e-mail: ktomko@osc.edu

D. Pekurovsky
San Diego Supercomputer Center, San Diego, MC, USA
e-mail: dmitry@sdsc.edu

1 Introduction

Current generation supercomputing systems are comprised of thousands of compute nodes based on modern multi-core architectures and high-speed interconnection networks that offer low latencies and high bandwidth. Together, these systems are allowing scientists to scale their parallel applications across tens of thousands of processes. The Message Passing Interface (MPI) [8] has been the dominant programming model for the past couple of decades. It has undergone some revisions, and the current standard version is 2.2. MPI defines a set of collective operations that are used to commu-

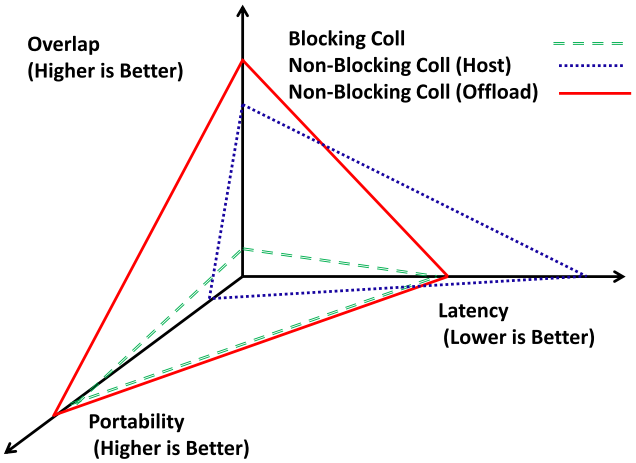


Fig. 1 Design space of collective algorithms

nicate data among a group of participating processes. The performance of collective operations is very critical to the overall scalability and performance of scientific parallel applications. Currently, MPI only defines blocking collective operations, i.e. the application has to wait until the collective call completes. This limits the overall performance and scalability of various scientific applications that extensively rely on collective communication. This has spurred interest in the design of non-blocking collective communication operations in MPI. The upcoming version of MPI, MPI-3, defines non-blocking collective communication operations to achieve communication/computation overlap.

In Fig. 1, we present the design space of collective communication algorithms. Current MPI implementations, such as MPICH2, Open-MPI and MVAPICH2 [9] rely on multi-core aware algorithms to optimize the latency of blocking collective operations across various systems [1, 10]. However, these schemes cannot offer overlap, because the collective operations are defined to be blocking operations. To overcome this limitation, researchers have explored host-based designs to overlap computation with collective operations. One of the major challenges of attempting overlap with collective operations is that the MPI communication stack needs to be progressed while the processors are busy in computation. One of the obvious approaches is to halt the computation on the processors and call MPI_Test to progress communication [6]. While this approach provides some benefits, the MPI application developer has to guess the correct number of times to call MPI_Test. Too few calls result in poor communication/computation overlap. Too many calls will unnecessarily increase the overhead of the MPI_Test calls. This is very hard to do in the context of real applications because: *i)* the number of test calls varies according to computation time and network speed, prohibiting performance portability, and *ii)* the computation for the application may be done within a third party library (such is the case with our example of P3DFFT) limiting opportunities for inserting test calls. Another approach is to use separate threads that call MPI_Test, as suggested in [5] and schedule them on separate compute cores. However, this solution reduces the number of compute cycles that are available to the application. We also expect the base latency of these approaches to be higher than the optimized blocking operations, owing to the costs associated with setting up the schedules. This leads us to the broad challenge: *Can we design a scalable, high performance interface for non-blocking collectives that delivers the best latency and overlap, in a portable manner?*

Recently, Mellanox has introduced network offload features in their ConnectX-2 [7] adapter. In [3, 4, 12], researchers have explored various facets of this interface. Using this feature, generic lists of communication tasks can be offloaded to the network interface. Such an interface eliminates the need for the host processor to progress communication and provides a low-level mechanism which can be leveraged to design non-blocking collective communication algorithms. However, in order to leverage the full benefits of this low-level mechanism, MPI libraries must be designed in a highly efficient manner.

The three-dimensional FFT (3D-FFT) is commonly used in many scientific applications [2, 13]. 3D-FFT kernels typically perform large message MPI_Alltoall exchanges to implement the transpose operations and spend nearly 50% of their run-times in the MPI_Alltoall operations [13]. Hence, the performance and scalability of these applications is highly dependent on the choice of the MPI_Alltoall implementation. We believe that overlapping the communication intensive MPI_Alltoall operation with application-level computation holds much promise for improving the performance and scalability of an entire class of such applications. This challenge leads us to the following questions:

1. Is it possible to achieve near perfect communication/computation overlap by offloading collective operations to the network interface?
2. Can we leverage the InfiniBand network offload feature to design the MPI_Ialltoall operation efficiently?
3. Can the throughput of applications be improved by overlapping communication with computation through our proposed non-blocking designs?
4. Finally, can we re-design a 3-D FFT kernel to leverage our proposed MPI_Ialltoall operation to achieve better run-times?

In this paper, we present our designs for MPI_Alltoall and MPI_Ialltoall (MPI-3 interface) operations that leverage the network offload feature offered by the InfiniBand ConnectX-2 adapter. We also re-design a Parallel Three-Dimensional Fast Fourier Transform (P3DFFT) [11] library, to leverage our MPI_Ialltoall implementation to

achieve communication/computation overlap. P3DFFT is an open-source library that implements three-dimensional FFT algorithms and is based on the MPI programming model. Our studies indicate that we are able to achieve an improvement of about 23% in the overall application run-time, compared to the default blocking version and about 17% when compared to the host-based implementations of the MPI_Ialltoall operation.

2 Background and related work

In this section we give the necessary background information for our work.

2.1 InfiniBand and ConnectX-2 network interface

InfiniBand is a popular switched interconnect standard being used by 41% of the Top500 Supercomputing systems [14]. Current generation InfiniBand network cards and switches can deliver 32 Gbps bandwidth and about 1–1.5 µs latency. The ConnectX-2 [7] network interface is the latest adapter from Mellanox. Along with all of the standard InfiniBand features, it offers a new network offloading feature called CORE-Direct. Using this feature, we can create arbitrary lists of send, receive and wait operations and post them to a work-request queue of the network card. Once the task-list has been posted, the network interface executes it and eliminates the need for the host processor to progress them. Using such task-lists, non-blocking collective operations may be designed by upper-level libraries. Switch-based collectives have been proposed by Voltaire [15], for non-personalized collectives (MPI_Barrier and MPI_Reduce). Since we focus on the MPI_Alltoall operation, which has a significantly higher communication volume, this technology is not explored here.

2.2 Message passing interface

The Message Passing Interface (MPI) [8] is one of the popular programming models used for designing parallel applications. In our work, we use the MVAPICH2 [9] software stack, a high performance MPI implementation over InfiniBand and RDMA networks, used by more than 1,500 organizations worldwide.

2.3 P3DFFT and its applications

Many applications in areas including Direct Numerical Simulations of Turbulence, astrophysics, and material science rely on highly scalable 3D FFTs [2, 13]. The Parallel Three-Dimensional Fast Fourier Transforms (P3DFFT) library [11] from the San Diego Supercomputer Center (SDSC) is a portable, high performance, open source implementation based on the MPI programming model. It leverages the fast serial FFT implementations of either IBM's ESSL or FFTW. P3DFFT uses a 2D, or pencil, decomposition and overcomes an important limitation to scalability inherent in FFT libraries by increasing the degree of parallelism up to N^2, where N is the linear size of the data grid. It has been used in various Direct Numerical Simulation (DNS) turbulence applications [2]. In this paper, we have re-structured the P3DFFT library to leverage our proposed implementation of the MPI_Ialltoall operation and use one of the sample programs provided with the P3DFFT library distribution, *test_sine*, to evaluate the performance benefits.

3 Design space of collective algorithms

In Fig. 1, we compare host-based and network-offload designs for the MPI_Alltoall operation across three dimensions: latency, portability and overlap. The Alltoall Personalized Exchange operation (MPI_Alltoall) is the most dense collective operation defined in the MPI Standard. MVAPICH2 uses the hypercube algorithm for small messages and the pair-wise exchange algorithm for larger messages.

Latency: Across MPI implementations, the host-based blocking algorithms are optimized for latency. However, we expect the latency of host-based non-blocking versions, such as in LibNBC [6] to be poorer than the default host-based algorithm used in MVAPICH2, depending on the cost of creating the schedules and the choice of the algorithms. We believe that the network-offload designs should perform comparably with the host-based pair-wise exchange algorithm.

Overlap: The default host-based blocking MPI_Alltoall operation does not offer any overlap. LibNBC allows users to achieve communication/computation overlap, but it is also a host-based approach. Applications can be designed to initiate collective operations and use MPI_Test calls to progress them, while performing compute tasks. LibNBC's threaded progression mode spawns a new thread that progresses the collective schedule in the background while the application performs computation. These schemes have been demonstrated to perform well when idle cores can be spared in each compute node. With modern network interfaces that offer support for offloading tasks, we can offload task-lists of communication operations to the network, and utilize the computing cycles of the host processors, while the NIC executes the task-lists. We believe that such network-offload based designs have the potential to maximize overlap.

Portability: The host-based blocking operations are designed to be portable across various systems. However, if applications are designed to use the host-based non-blocking operations along with MPI_Test calls, it is necessary to determine the right frequency of MPI_Test calls to maximize overlap. This is neither trivial nor portable and places a high burden on application scientists. Host-based non-blocking

approaches that rely on real-time threads to achieve overlap are often tricky and lead to complications within the MPI libraries. With network-offload designs, the NIC can independently progress the collective operations in the background and the computing cycles of the host processors are available for computation.

4 Designing non-blocking algorithms with collective offload

Researchers have demonstrated the overlap capabilities offered by the ConnectX-2 network interface with MPI_Barrier [3]. In [4], a set of primitives that can be used to design collective operations to leverage the network offload feature were proposed. However, neither of these have designed scalable, non-blocking versions for data-moving collective operations. In this section, we discuss our proposed ideas for designing a high performance, scalable MPI_Ialltoall operation, the most dense collective operation.

4.1 Point-to-point communication protocols for overlap with offload

We use a separate set of InfiniBand Completion Queues (CQ) and queue-pairs (QP) for all offload communication. We also use separate QP's for small and large message transfers. For small messages, we use a protocol similar to the existing eager protocol in MVAPICH2. The traditional rendezvous protocol requires intermediate intervention to process the hand-shake packets and start the actual data transfer. However, the current generation CX-2 interface does not support hardware-level tag-matching. Hence, we rely on the InfiniBand *Receiver Not Ready* (RNR) feature to maximize overlap for large messages.

4.2 Designing a scalable non-blocking Alltoall personalized exchange algorithm

The pair-wise exchange algorithm is commonly used to implement large message Alltoall exchanges across MPI libraries. For P processes, each process performs P iterations, performing a blocking *sendrecv* operation in each step. We define a "Communication-Window" to include a variable number of send-tasks and a wait task. The current generation CX-2 interface allows us to post task-lists containing a fixed number of entries. Hence, this directly limits the scalability of an offload version of the Alltoall operation. In order to address this limitation, we propose using a light-weight *Offload-progress-thread*. The host processor creates the entire task-list when MPI_Ialltoall is invoked. But, we split the operation into multiple phases, and we rely on our

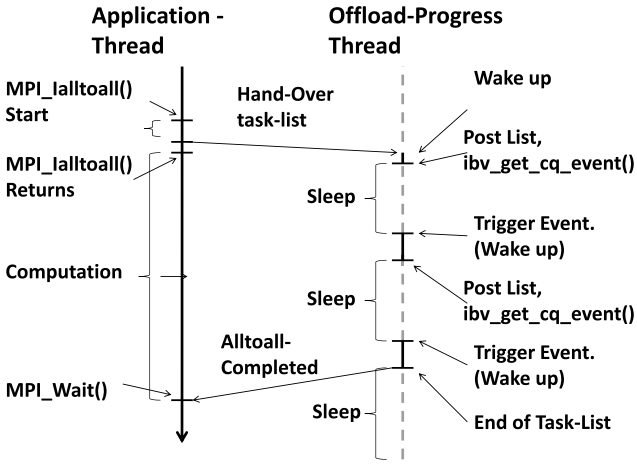


Fig. 2 Designing scalable offload Ialltoall

Offload-progress-thread to post a sub-task-list, a portion of the overall task-list, in each phase.

However, as discussed in Sect. 3, it is necessary to minimize the resource contention between the application thread and the offload-thread. We describe our approach to tackle this problem in Fig. 2. We use a "trigger" operation in each phase of our algorithm. We create a separate *trigger_qp* and a light-weight *trigger_cq*, to allow a process to communicate with itself. We also create a completion channel, *trigger_comp_channel*, associated with the trigger_cq. At the end of a sub-task-list, each process en-queues a send task to itself on the trigger_qp. The offload-progress-thread posts a sub-task-list and calls *ibv_get_cq_event*, and is scheduled into a sleep state. The NIC independently executes the sub-task-list and finally executes the send on the trigger_qp. This generates a network interrupt on the trigger_comp_channel, signaling the offload-progress-thread to wake up and post the next sub-task-list (if any).

Through such a design, we are able to work around the limitation imposed by the current generation CX-2 interface and scale the Alltoall operation. We have also ensured that the progression thread remains active for a very short time.

The other major challenge concerning the design of an efficient interface for non-blocking collectives deals with handling un-expected messages which arise due to process skews. For small messages, the unexpected messages can be buffered internally by pre-posting buffers on the small message QP. For large messages, we leverage the *Receiver Not Ready* (RNR) feature, as the network guarantees to perform the data transfer only after the target process has posted its receive operation.

A non-blocking interface for collective operations should also allow applications to initiate multiple non-blocking operations and wait on them later. For a collective like MPI_Alltoall, the network can easily get congested if multiple Alltoall operations progress in parallel. In our design, we

append the task-lists created by different MPI_Ialltoall operations and post them sequentially, in the same order. The application can post multiple MPI_Ialltoall operations and call MPI_Wait on them later, without any intermediate intervention. Our modified P3DFFT kernel leverages this feature to overlap two different MPI_Ialltoall operations with compute tasks, simultaneously.

5 Redesigning P3DFFT to achieve communication/computation overlap

The Cooley-Tukey algorithm for FFT used for 1D FFTs is very efficient computationally, $O(N \log N)$. However, the butterfly pattern of memory accesses of this algorithm makes it challenging to achieve good parallelism. To perform a 3D FFT, the 1D transform must be applied in each of the three dimensions. Two primary strategies are possible:

1. Direct approach: Develop a parallel 1D FFT and communicate as necessary to carry out an FFT on data that is distributed.
2. Transpose approach: Rearrange data prior to each 1D FFT such that the data for the FFT is available locally and a serial 1D FFT can be used.

While both methods require expensive communication operations, the commonly used transpose approach affords the opportunity to combine many smaller messages as a larger buffer in a single all-to-all exchange. In order to scale this approach to a large number of processors, a 2D domain decomposition (*pencils*) is used in P3DFFT. This algorithm first performs a 1D FFT along the X dimension, followed by a transpose between the X and the Y dimensions. The same pattern is then repeated across the Y and the Z dimensions, followed by a 1D FFT along the Z dimension.

The original data array is typically distributed as pencils along the X dimension, with the Y, Z dimensions being split among processors in rows and columns of the 2D processor grid. In the first transpose, the Y dimension is gathered to become local while the X dimension is split among the row processors. This involves an All-to-All exchange in rows, which is implemented in the baseline version as MPI_Alltoall over Cartesian sub-communicator *ROW*. The second transpose similarly brings together all data for the Z dimension and splits the Y dimension within columns, which involves an All-to-All over communicator *COL*.

Each of the two transposes requires P_{row} (or P_{col}) MPI_Alltoall exchanges of N^3/P_{row} (or N^3/P_{col}) elements. This typically implies that applications have to perform several large message MPI_Alltoall operations, and therefore the performance is bandwidth-bound. The row transpose typically takes much less time than the column one since the tasks in the ROW communicator fall in the same node (or on a few adjacent nodes), so the fraction of communication happening over the network is smaller.

```
1D FFT in x for V1
transpose x and y of V1
1D FFT in y for V1
Initiate transpose y and z of V1
do Vj = V2 to Vn
    1D FFT in x for Vj
    transpose x and y of Vj
    1D FFT in y for Vj
    Initiate transpose y and z of Vj
    Wait for transpose complete for Vj−1
    1D FFT in z for Vj−1
enddo
Wait for transpose complete for Vn
1D FFT in z for Vn
```

Fig. 3 Algorithm for the forward transform in the redesigned multi-variable, pipelined, overlapped version

The *test_sine* kernel is a sample driver for the P3DFFT library. The baseline version generates a 3D sine function and calls a forward and backward transform repeated over a number of iterations and verifies the results obtained.

In many applications of 3D FFT, it is necessary to transform several independent arrays (variables) at a time. When designing a version enhanced with overlap of communication and computation we chose to avoid splitting the transposes into smaller chunks. Instead we keep the bulk transposes in place by overlapping communication and computation stages for different variables. Thus the initial and result sine array and intermediate Fourier space array in the *test_sine* kernel are augmented with an extra dimension representing the number of variables, which is also passed to the forward and backward FFT routines. The forward FFT routine is restructured as shown in Fig. 3 and described below. The backward routine is similarly restructured.

Here the loop index j runs over the variables that need to be transformed. We start with a prologue which performs the first three stages of the 3D FFT algorithm for variable 1 (V_1), namely transform in X, row transpose, and transform in Y. Then we initiate (post) an all-to-all exchange in the COL communicator for this variable and let the non-blocking communication proceed. Assuming it does not interfere with the CPUs, the latter can work on XY transform of the second variable. This is the overlap of computation for XY transform of V_2 with network communication for V_1. After posting the exchange for V_2, we are then ready to complete the exchange for the first variable with a Wait call followed by a transform of the resulting array (now in shape of Z-pencils) in Z dimension, thus completing the algorithm for V_1. Meanwhile the non-blocking exchange for variable V_2 is ongoing behind the scenes, thus achieving an overlap of FFT in Z for V_1 with Column transpose for V_2. The cycle continues for the rest of the variables, always keeping one exchange in the background.

This design is only one of many possible (and work on this continues). It should be kept in mind that in practice the column transpose dominates the row transpose since the latter occurs among contiguously numbered tasks which are typically placed on cores of the same node. Assuming the size of ROW communicator is not too large, the row transpose occurs entirely within the nodes. This decision is consistent with the current trend of increasing cores per node on large system, therefore we chose not to overlap the row transpose with computation. However it is overlapped with the column transpose in this pipe-lined design.

To summarize, we have developed a version of the 3D FFT kernel which achieves communication/computation overlap. This version allows us to study latency hiding with different versions of MPI_Ialltoall operations. For this paper, we consider both the Host-Based and our proposed network offload implementations for the MPI_Ialltoall as replacements for the MPI_Alltoall operation.

6 Experimental results

We detail the results of our experimental evaluation in this section.

6.1 Experimental setup

Each node of our 512-core testbed has eight Intel Xeon cores running at 2.53 GHz with 12 MB L3 cache. The cores are organized as two sockets with four cores per socket. Each node also has 12 GB of memory and Gen2 PCI-Express bus. They are equipped with MT26428 QDR ConnectX-2 HCAs with PCI-Ex interfaces. We used a 171-port Mellanox QDR switch, with 11 leafs, each having 16 ports. Each node is connected to the switch using one QDR link. The HCA as well as the switches use the latest firmware. The operating system used is Red Hat Enterprise Linux Server release 5.4 (Tikanga), with the 2.6.18–164.el5 kernel version. OFED version 1.5.1 is used on all machines, and the OpenSM version is 3.3.7. We would like to note that there were non-deterministic race-conditions while running LibNBC's basic threaded version and LibNBC's real-time thread option was crashing the nodes, while running with super-user permissions. For the rest of this paper, we refer to LibNBC's MPI_Test approach as "Host-Based-Test" and its threaded approach as "Host-Based-Thread."

6.2 Benchmark suite

In this paper, we use modified versions of the OSU Micro-Benchmarks, which are a part of the MVAPICH2 software package. We use the *osu_alltoall* benchmark to measure the average latency of the Network-Offload based MPI_Alltoall operation for various message sizes.

Overlap benchmark: In this benchmark, we measure the overlap percentage between host-based and offload-based MPI_Ialltoall implementations. For each message size, we first measure the baseline latency, with no overlapping computation. We then insert a compute loop between MPI_Ialltoall and MPI_Wait calls that runs for the same duration as the baseline latency for each message size. We measure the time consumed to complete both the compute and communication tasks and calculate the percentage overlap achieved for the MPI_Ialltoall implementation. For the Host-Based-Test version, we also introduce MPI_Test calls at different frequencies to examine its impact on the overlap percentage. We experimented with a large range of test-frequencies, but report about four of them for the sake of brevity. In Fig. 5, Host-Based-Test-X indicates that we invoked the MPI_Test call X times to progress the communication, while performing compute tasks.

Throughput benchmark: In this benchmark, we perform floating point matrix-matrix operations by invoking the cblas_dgemm function supported by the Intel MKL Library (10.2.1.017), between the MPI_Ialltoall and the MPI_Wait operations. We measure the overall time required for completion and compute the GFLOPS rating for the given case and compare it against the theoretical peak FLOPS rating for our system. We also report the average results from multiple runs to eliminate any experimental errors.

6.3 Latency of various Alltoall schemes

In Figs. 4(a), (b) and (c), we compare the latency of the default host-based MPI_Alltoall operation in MVAPICH2, with the Host-based Test and Thread approaches along with our proposed Network-Offload based MPI_Alltoall algorithm, for 64, 128 and 256 processes. For the Host-based Test and Thread approaches, we post the NBC_Ialltoall operation, immediately followed by the NBC_Wait() operation to measure the latency when no overlap is attempted. We can observe that our proposed Network-Offload design delivers nearly the same latency as the default host-based pair-wise exchange algorithm. We also observe that the Host-based Test and Threaded implementations have poorer latency.

6.4 Computation/communication overlap

In Fig. 5, we compare the communication/computation overlap measured through our overlap benchmark, across various system sizes. We compare our proposed Network-Offload implementation of MPI_Ialltoall with the host-based implementations that rely on either using MPI_Test calls (Host-Based-Test) or a separate progress-thread (Host-Based-Thread) for achieving overlap. We observe that our proposed network-offload implementation delivers near-perfect overlap, about 95–99% for different message sizes,

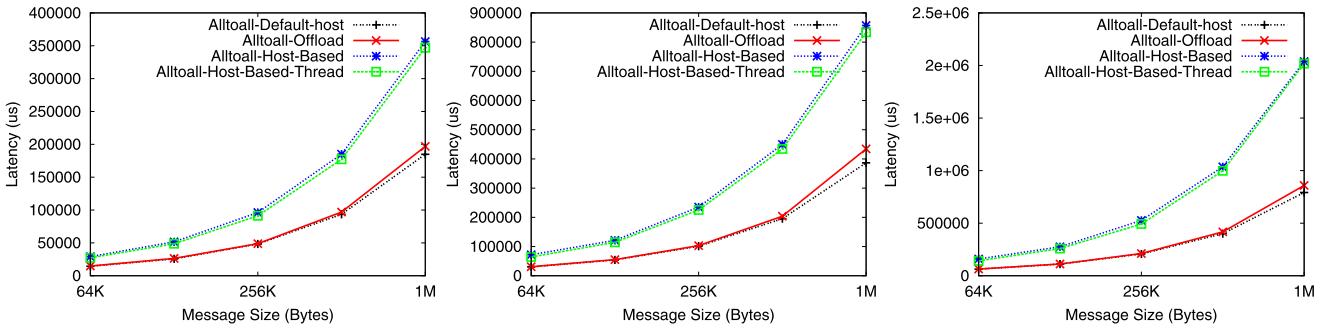


Fig. 4 Latency comparison: (**a**) 64 processes, (**b**) 128 processes, and (**c**) 256 processes

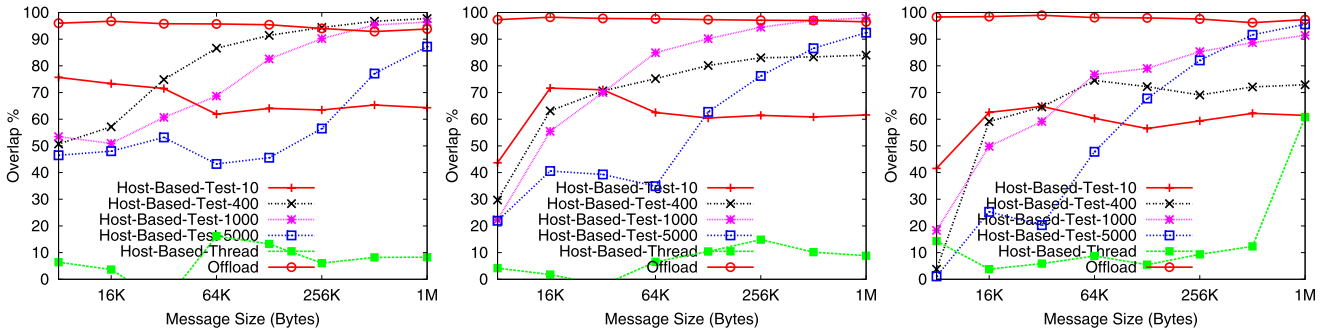


Fig. 5 Overlap comparison: (**a**) 64 processes, (**b**) 128 processes, and (**c**) 256 processes

across different system sizes. With the Host-Based-Test version, we observe that we can get good overlap when we use the right number of MPI_Test calls. We would also like to point out that the optimal frequency of MPI_Test calls varies across system sizes. For example, with 64 processes, having about 400 calls to MPI_Test appears to deliver better overlap. However, with 256 processes, we observe better overlap when we make 1,000 calls to MPI_Test. This is consistent with our claims in Sect. 3. It is not an easy task to determine the right frequency of MPI_Test operations in a portable manner. We also observe that the Host-Based-Thread version delivers poor overlap in these experiments. This is probably because all our experiments were run in a fully-subscribed manner (8 processes in each node utilizing all of the 8 cores). It is to be noted that typical supercomputer users also use all available cores within compute nodes. Thus, the Host-Based-Thread version is prone to poor overlap in practical usage scenarios.

6.5 Application throughput

In this section, we discuss the impact of various non-blocking Alltoall designs on the overall throughput of DGEMM, a matrix-matrix multiplication program. For a given number of processes, we fix the message length to be 128 kB and we vary the problem size for the matrix-matrix operation gradually and measure the throughput in GFLOPS. We compare it to the estimated theoretical peak of the system, based on the CPU clock frequency (2.53 GHz) and the fact that the Xeon processors used have four floating point units (10 GF double precision per core). With DGEMM, dividing the matrix-matrix multiplication into a large number of computation chunks is complicated since computation is proportional to (N^3), i.e., it is not a linear relationship. In order to study overlap benefits with the Host-based Test approach, we interleave the NBC_Test calls with multiple calls to cblas_dgemm, with adjusted matrix dimensions, so that the overall compute operation remains equivalent to making one cblas_dgemm with matrix of size N. We observed that the overall throughput of the Host-Based-Test implementation does not vary for a range of different frequencies of MPI_Test calls.

In Fig. 6, we compare the throughput of DGEMM when it is overlapped with different implementations of non-blocking Alltoall operations for 256 and 512 processes. We can observe that as the problem size increases, the throughput of the version with network-offload is higher than the host-based version and both the versions saturate at very large values of N for 256 processes. We also note that the network-offload version performs even better with 512 processes. We would like to note that scientific applications are typically run in a manner to utilize about 50–60% of the memory available per core, and many systems like the IBM-

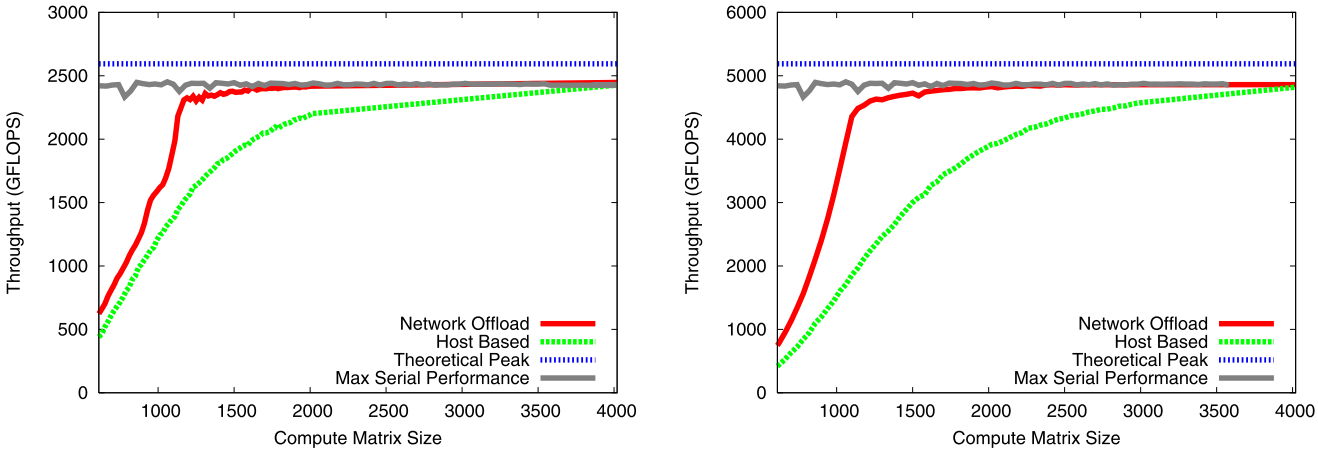


Fig. 6 Throughput comparison of various schemes with (**a**) 256 and (**b**) 512 processes

Table 1 Application run-times with different Alltoall schemes

Job size	64			128		
Alltoall Impl	Offload	H-Test	Blocking	Offload	H-Test	Blocking
Problem size						
512	1.81	2.02	2.29	0.89	1.05	1.15
600	3.14	3.72	3.84	1.57	1.91	1.92
720	5.36	6.34	6.63	2.93	3.27	3.32
800	7.99	8.83	9.09	3.90	4.43	4.48

Blue Gene, offer smaller memory per core. Therefore, realistically, one cannot arbitrarily increase N.

6.6 P3DFFT kernel performance comparison

To evaluate how our network offload Alltoall operation can be utilized to improve the performance of applications which require many 3D FFT operations, we performed a study with the P3DFFT Sine kernel. As described in Sect. 5, we replaced the two most expensive Alltoall operations in the P3DFFT Sine kernel (namely the column wise transposes which occur in both the forward and backward transforms) with non-blocking alternatives, for various 3D FFT problem sizes. In Table 1, we compare the application run-times of the baseline blocking version, the redesigned P3DFFT kernel with overlapped collective communication with the host-based and Network-Offload based MPI_Ialltoall implementations. We report experimental results for different "Job Sizes," with 64 and 128 processes. We also vary the problem size, N, between 512 and 800. From Fig. 7, we can see that the kernel with our proposed MPI_Ialltoall consistently performs better than the one with MPI_Alltoall by about 10%–23% and the kernel that uses the non-blocking MPI_Ialltoall operation offered by the Host-Based-Test approach by about 10%–17%.

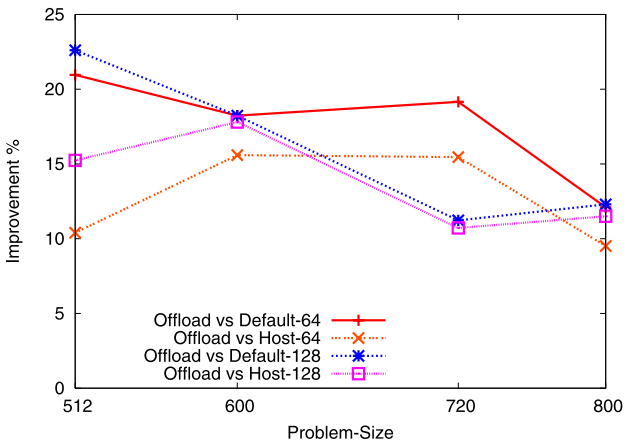


Fig. 7 Performance improvement comparison with 64 and 128 processes

As discussed in Sect. 6.5, it is not trivial to split up compute tasks to accommodate a large number of MPI_Test calls, particularly when the computation is being performed by a third-party library. In our redesigned P3DFFT the computation work that is overlapped with the Alltoall is carried out by either the FFTW or ESSL FFT libraries. For the Host-based-Test version, we have split the FFT calls into several calls, interleaved with MPI_Test calls. We have modified the P3DFFT library to support a small range of test call frequencies, with the maximum amount equal to one test call per each 1D FFT in the pencil assigned to a process. We report the best run times obtained for each problem size.

7 Conclusion

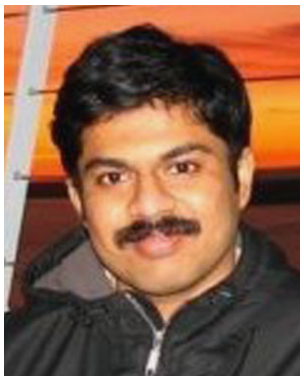
In this paper, we have designed a high performance, scalable Network-Offload based non-blocking algorithm for the Alltoall Personalized Exchange operation. Simultaneously,

we have also re-designed the P3DFFT library and a sample application kernel to study the benefits of overlapping application level computation with the Alltoall operation. Our experimental evaluation shows that we achieve near perfect overlap of computation and communication (99%) and we also deliver very good application throughput, with 512 processes. We also see an improvement of about 23% in the overall run-time with the modified P3DFFT kernel when compared to the default blocking version and about 17% better than the Host-Based Non-Blocking Alltoall implementations. We plan to incorporate the proposed network-offload implementation of MPI_Ialltoall() in a future MVAPICH2 release and contribute an efficient implementation of P3DFFT for modern InfiniBand clusters based on the ConnectX-2 network adapters. We also plan to design non-blocking collective operations (such as MPI_Bcast, MPI_Reduce, MPI_Allreduce, etc.) based on the ConnectX-2 network offload feature.

References

1. Mamidala AR, Kumar R, De D, Panda DK (2008) MPI collectives on modern multicore clusters: performance optimizations and communication characteristics. In: 8th IEEE international symposium on cluster computing and the grid, Lyon, pp 130–137
2. Donis DA, Yeung PK, Pekurovsky D (2008) Turbulence simulations on O(10^4) processors. In: TeraGrid
3. Graham R, Poole S, Shamis P, Bloch G, Boch N, Chapman H, Kagan M, Shahar A, Rabinovitz I, Shainer G (2010) Overlapping computation and communication: barrier algorithms and Connectx-2 CORE-direct capabilities. In: Proceedings of the 22nd IEEE international parallel & distributed processing symposium, workshop on communication architectures for clusters (CAC)'10
4. Subramoni H, Kandalla K, Sur S, Panda DK (2010) Design and evaluation of generalized collective communication primitives with overlap using ConnectX-2 offload engine. In: The 18th annual symposium on high performance interconnects, HotI
5. Hoefler T, Lumsdaine A (2008) Message progression in parallel computing—to thread or not to thread. In: Proceedings of the IEEE international conference on cluster computing
6. Hoefler T, Squyres J, Rehm W, Lumsdaine A (2006) A case for non-blocking collective operations. In: Frontiers of high performance computing and networking. ISPA 2006 workshops. Lecture notes in computer science, vol 4331, pp 155–164
7. Mellanox technologies. ConnectX-2 Architecture. http://www.hpcwire.com/features/Mellanox-Rolls-Out-Next-Iteration-of-ConnectX-57046327.html
8. MPI Forum. MPI: a message passing interface. www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf
9. MVAPICH2. http://mvapich.cse.ohio-state.edu/
10. Karonis NT, de Supinski BR, Foster I, Gropp W, Lusk E, Bresnahan J (2000) Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In: Proceedings of the 14th international symposium on parallel and distributed processing, p 377
11. Parallel three-dimensional fast Fourier transforms (P3DFFT) library, San Diego Supercomputer Center (SDSC). http://code.google.com/p/p3dfft
12. Graham R, Poole S, Shamis P, Bloch G, Boch N, Chapman H, Kagan M, Shahar A, Rabinovitz I, Shainer G (2010) ConnectX2 InfiniBand management queues: new support for network offloaded collective operations. In: CCGrid'10, Melbourne, Australia, May 17–20
13. Laizet S, Lamballais E, Vassilicos JC (2010) A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution DNS of fractal generated turbulence. Comput Fluids 39:471–484
14. Top500. Top500 supercomputing systems, Oct 2010
15. Voltaire. Fabric collective accelerator (FCA)

Krishna Kandalla is a Ph.D. student in the Department of Computer Science and Engineering at The Ohio State University. He is a member of the Network-Based Computing Laboratory lead by Dr. D.K. Panda. His research interests include High Performance Computing, High Speed Inteconnects and MPI Collective Communication. His recent research involves designing non-blocking collective algorithms by leveraging the network offload feature offered by the latest InfiniBand network interfaces. He has also worked on designing multi-core-aware, network-topology-aware and power-aware algorithms for MPI collective operations and has published papers in various international conferences and workshops. More details are available at: http://www.cse.ohio-state.edu/~kandalla.



Hari Subramoni is a Ph.D. student in the Department of Computer Science and Engineering at The Ohio State University. He is a member of the Network-Based Computing Laboratory lead by Dr. D.K. Panda. His research interests include High Performance Computing, High Speed Inteconnects, High Performance Data Transfers in InfiniBand WAN scenarios, Network Topology Aware Collective Communication and Differentiated Quality of Service in HPC. His recent research involves utilizing multiple virtual lanes to alleviate network contention in large scale InfiniBand Networks. More details are available at: http://www.cse.ohio-state.edu/~subramon.

Karen Tomko is a Senior Research Scientist at the Ohio Supercomputer Center. Her research interests are in the field of high performance computing, application optimization and accelerator technologies. She has collaborated with computational scientists for 15 years. Her experience ranges from optimization and parallelization of crash-worthiness simulation to development of a run-time adaptive parallelization scheme for wireless communications simulations. Current projects include collaborating with physicists on development of a multi-scale quantum simulation for the

study of strongly correlated materials and collaboration OSU's MVAPICH/MVAPICH2 group on development of scalable communication libraries for HEC applications. Dr. Tomko is also involved in defining and reviewing curricular guidelines for virtual computational science initiatives at the statewide and national level. She received her Ph.D. from the University of Michigan in 1995, and has previously held faculty positions at University of Cincinnati and Wright State University.

Dmitry Pekurovsky is on staff at San Diego Supercomputer Center at the University of California at San Diego. He has a Ph.D. in computational physics. His interests include creating and optimizing HPC software for a number of areas in computational science and engineering. He is the main author of an open-source package P3DFFT for highly scalable three-dimensional Fast Fourier Transforms.

Sayantan Sur is a Research Scientist at the Department of Computer Science at The Ohio State University. His research interests include high speed interconnection networks, high performance computing, fault tolerance and parallel computer architecture. He has published more than 20 papers in major conferences and journals related to these research areas. He is a member of the Network-Based Computing Laboratory lead by Prof. D.K. Panda. He is currently collaborating with National Laboratories, Supercomputer Centers, and leading InfiniBand companies on designing various subsystems of next generation high performance computing platforms. He has contributed significantly to the MVAPICH/MVAPICH2 (High Performance MPI over InfiniBand and 10GigE/iWARP) open-source software packages. The software developed as a part of this effort is currently used by over 1,500 organizations in 60 countries. In the past, he has held the position of Post-doctoral researcher at IBM T.J. Watson Research Center, Hawthorne and Member Technical Staff at Sun Microsystems. Dr. Sur received his Ph.D. degree from The Ohio State University in 2007. More details are available at: http://www.cse.ohio-state.edu/~surs.

Dhabaleswar K. Panda is a Professor of Computer Science at the Ohio State University. His research interests include parallel computer architecture, high performance networking, InfiniBand, exascale computing, programming models, high performance file systems and storage, accelerators, virtualization and cloud computing. He has published over 285 papers (including multiple Best Paper Awards) in major journals and international conferences related to these research areas. Dr. Panda and his research group members have been doing extensive research on modern networking technologies including InfiniBand, 10GigE/iWARP and RDMA over Converged Enhanced Ethernet (RoCE). His research group is currently collaborating with National Laboratories and leading companies on designing various subsystems of next generation high-end systems. The MVAPICH/MVAPICH2 (High Performance MPI over InfiniBand, 10GigE/iWARP and RoCE) open-source software packages, developed by his research group (http://mvapich.cse.ohio-state.edu), are currently being used by more than 1,500 organizations worldwide (in 60 countries). This software has enabled many InfiniBand clusters to get into the latest TOP500 ranking. These software packages are also available with the Open Fabrics stack for network vendors (InfiniBand, iWARP and RoCE), server vendors and Linux distributors. Dr. Panda's research is supported by funding from US National Science Foundation, US Department of Energy, and several industry including Intel, Cisco, SUN, Mellanox, and QLogic. Dr. Panda is an IEEE Fellow and a member of ACM.