Co-Designing MPI Library and Applications for InfiniBand Clusters

S. Sur, S. Potluri, K. Kandalla, H. Subramoni, K. Tomko[†] and D. K. Panda

Department of Computer Science and Engineering, The Ohio State University

[†]*Ohio Supercomputer Center*

{surs, potluri, kandalla, subramon, panda}@cse.ohio-state.edu
ktomko@osc.edu[†]

"Co-designing applications and communication libraries to leverage features of underlying communication network is imperative for achieving optimal performance on modern computing clusters."

Keywords: Co-design, MPI, InfiniBand, Applications, Clusters

I. Introduction

Scientific computing is credited for many of the technological breakthroughs of our generation. It is used in fields ranging from drug discovery, aerospace, weather prediction to seismic analysis and many others. Scientific computation often deals with very large amounts of data and its algorithms need to compute results from mathematical models. Due to its compute and data intensive nature, these applications are often parallel, i.e. they perform calculations simultaneously on multiple computers.

The TOP500 list [2] ranks the top supercomputing sites across the world. Recently, top systems have crossed the Petaflop (10^{15} floating point operations) barrier. It is expected that Exaflop (10^{18}) levels will be reached by the turn of the decade. This fast growth in the HPC field is being sustained by the low costs that are afforded by commodity components. Commodity components include general purpose processors from Intel, AMD, and IBM; GPGPUs from NVIDIA etc.; I/O buses such as PCI express and interconnection networks such as InfiniBand.

As machines grow ever larger and more powerful, the entire networking stack and application stack needs to evolve. An underlying design principle in HPC is to expose, not hide, system features that lead to better performance. However, as system complexity grows, this must be done in a manner that does not overwhelm application developers with detail.

In order to keep scaling applications on increasingly more powerful systems, it is imperative to explore new architectures from system point of view and new programming paradigms from application point of view. In this article we explore the co-design approach to taking advanced features from commodity network such as InfiniBand, incorporating the design into a state-of-the-art MPI communication library and finally modifying the applications to leverage these new features.

A. Major Bottlenecks in Continued Scaling of Applications

In practice, parallel applications experience additional overheads of messaging. One of the costs is due to unnecessary synchronization of processors (sending process must ensure that the receiving process has successfully received the message). The over-synchronization problem can be solved by adopting novel programming model concepts, such as, one-sided communication. In one-sided communication space to store messages is allocated before the actual exchange takes place.

Another factor is the overhead from processors communicating in groups (collective communication). Collective communication often involves large volumes of data and communication schedule to optimize usage of the network. While communication scheduling yields better utilization of the network, one delayed process may delay all other processes in the collective operation. Additionally, performing the communication scheduling tasks, such as waiting for messages and forwarding them leads to lost processor cycles. During these tasks, the main CPU cannot perform any useful work, thus lowering overall efficiency. With the help of advanced and intelligent networks, the communication schedule and progress tasks can be offloaded to the network adapter, freeing up the CPU for useful tasks.

At the same time, if all of the major components of the system architecture, messaging libraries and end applications evolved separately, the end result will be a loosely coupled system. Performance would then be bounded by the weakest link. It is important that these components are "co-designed" to extract the maximum performance from a system.

II. Background

A. InfiniBand Network Architecture

InfiniBand [1] is a very popular switched interconnect standard being used by over 40% of the Top500 Supercomputing systems [2]. Current generation InfiniBand network cards and switches with QDR speed can deliver 32 Gbps end-to-end bandwidth and about 1-1.5 μsec latency.

One of the major features of InfiniBand is Remote Direct Memory Access (RDMA). Using RDMA one process can remotely read or write memory contents of another process without any involvement of the remote processor. This feature is very powerful and when used intelligently in communication library design can provide benefits of reduced synchronization requirements in addition to pure latency benefits. The ConnectX-2 network interface is the latest InfiniBand adapter from Mellanox. Along with all the standard InfiniBand features, it offers a new network offloading feature called the CORE-Direct. Using this feature, arbitrary lists of send, receive and wait operations can be created. These lists can then be posted onto a work-request queue of the network adapter. Then, the network interface executes the tasks without involvement of the host processor. Using such task-lists, non-blocking collective operations may be designed by upper-level libraries.

B. The Message Passing Interface: MVAPICH2 High-Performance Design for InfiniBand

MPI has been the dominant parallel programming model for the past couple of decades. It has been widely ported and several open-source implementations have been made available. It has achieved very good performance and scalability. As a result, all modern super-computers support it. InfiniBand offers a low-level *verbs* interface with several types of Queue Pairs, with varying levels of services. This enables upper-level software, such as MPI implementations, to design flexible and high-performance connection management, buffer management, coalescing strategies, etc.

MVAPICH2 [8] is a high-performance implementation of MPI-2 standard on InfiniBand, Internet Wide Area RDMA Protocol (iWARP) and RoCE (RDMA over Converged Ethernet). The internal design of MVAPICH2 has been systematically designed to achieve very good scalability by exploiting various InfiniBand features, such as Unreliable Datagrams, Shared Receive Queues (SRQ), and eXtended Reliable Connections (XRC) along with connection management strategies such as on-demand connections and buffering strategies for message coalescing to improve memory efficiency. All of these optimizations have been combined into one unified runtime. To the best of our knowledge, this is the most scalable runtime on InfiniBand that offers high-performance and is open-source. Over the last ten years, MVAPICH2 has been used as a state-of-the-art MPI for research in communication runtimes. It has also been used as a production MPI library on a large number of InfiniBand clusters around the world.

C. Parallel Scientific Applications

Scientific applications employ a wide-range of numerical techniques. In our work, we used two applications that use two different techniques: finite difference methods and Fourier transforms. These are described below.

1) AWP-ODC: The Anelastic Wave Propagation code by Olsen, Dey and Cui, AWP-ODC, is a community model [7] used by researchers at the Southern California Earthquake Center (SCEC). AWP-ODC solves the 3D velocity-stress wave equation explicitly by a staggered-grid Finite Difference (FD) method. The volume representing the ground area to be modeled is decomposed into 3D rectangular sub-grids to parallelize the code. Each processor is responsible for performing stress and velocity calculations for its portion of the grid, as well as applying boundary conditions at the external edges of the volume if its sub-grid is on the boundary. Ghost cells, comprising a two-cell-thick padding layer, manage the most recently updated wave-field parameters exchanged from the edge of the neighboring sub-grids. Some of the most detailed simulations to date of earthquakes along the San Andreas fault were carried out using this code, including the well-known TeraShake, SCEC ShakeOut simulations. This application was a finalist for the Gordon Bell Prize in 2010 [3].

2) P3DFFT: The Parallel Three-Dimensional Fast Fourier Transforms (P3DFFT) library [10] from the San Diego Supercomputer Center (SDSC) is a portable, high performance, open source implementation based on the MPI programming model. It has been used in Direct Numerical Simulation (DNS) turbulence applications [4]. P3DFFT leverages the fast serial FFT implementations of either IBM's ESSL or the FFTW library for efficient 1D FFT calculation. The FFT computations require two costly MPI_Alltoall communication operations to perform matrix transpose operations. As shown in [6], it is possible to re-structure the P3DFFT library to leverage our proposed implementation of the MPI_Ialltoall operation.

III. Co-designing Scientific Applications with MPI Library

In this Section we describe our approach towards co-designing applications with MPI library. First, we discuss the enhancements to the MVAPICH2 library to leverage novel network features. Then, we describe the changes to applications to leverage the improved MVAPICH2 MPI library. Our overall co-design approach is described in Figure 1.

A. Designing the MPI Library with Novel Network Features

In this Section we describe our enhancements to the MPI library to incorporate novel network features. The network layer provides new features such as RDMA, Offload, Loop-back that can be leveraged in the MPI layer as shown in Figure 1. Additionally, modern multi-core computing platforms also provide several additional features for optimal data transfer among the cores within a node. These features also need to be exploited for best performance. The following sub-sections describe the design of two important MPI enhancements that utilize these features.

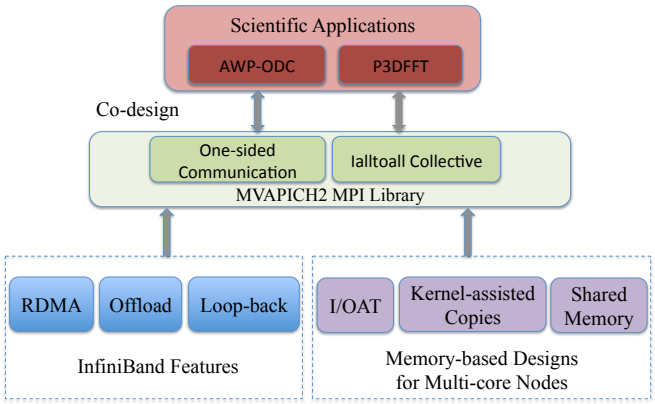


Fig. 1. Co-design Approach for MPI Library, Network and Scientific Applications

1) Improving One-sided Communication with RDMA: The one-sided model in MPI aims at reducing synchronization overheads that are inherent to communication using send/receive. Each process exposes a region of its memory (a window) to all the other processes in its communication group. Every process can then directly read from, write to or update window memory at any other process. All the parameters required for communication are provided by the origin process and does not require any intervention from the target. The communication operations are non-blocking in nature allowing for the data transfer to proceed asynchronously while the processor is free to do other useful work. The semantics of one-sided communication form a perfect match with the RDMA operations offered by the InfiniBand network. For example, an MPI_Put (write to a window) can be directly mapped on to an RDMA Write call by the MPI library. The asynchronous nature of both RDMA operations and MPI_Put allow for efficient computation and communication overlap [5].

The issue and completion of one-sided communication calls is controlled through synchronization operations. Though, all the communication operations require the participation of the origin process alone, synchronization operations can either be active (participation of both origin and target) or passive (participation of origin only). Passive synchronization is provided through MPI_Win_lock and MPI_Win_unlock calls. This mode of synchronization can be done only in a point-to-point fashion i.e., a process has to individually lock and unlock window at each process

it wants to communicate with. This leads to inefficiencies in applications which have a fixed communication pattern that involves multiple target processes. MPI_Win_fence is an example of active synchronization with collective semantics which requires the participation of all processes in the communicator. A flexible mode of active synchronization is provided through MPI_Win_post, MPI_Win_wait, MPI_Win_start and MPI_Win_complete calls. Using these calls, only a sub-group of processes in the communicator are allowed to synchronize. This leads to better performance, particularly in the case of applications where a process communicates only with a small set of processes, for example, its neighbors in the process grid. One of the applications we consider in this article, AWP-ODC, is a good use-case.

2) Designing Non-Blocking Alltoall Exchange with Collective Offload: The Alltoall Personalized exchange is the most communication intensive collective operation in the MPI-2.2 Standard. With N processes, the latency of a large message Alltoall operation is proportional to N^2 and this significantly affects the performance and scalability of various scientific applications. A fruitful strategy to improve application performance could be to design a high performance, non-blocking implementation for the Alltoall exchange, which may be leveraged by applications to overlap the Alltoall communication with computation. It is possible to use host-based approaches to design non-blocking collectives (libNBC) [12]. However, such methods require the host processor to progress the collective operation and this directly limits the overlap and is also not very portable.

MPI libraries can leverage the network offload feature in the ConnectX-2 InfiniBand adapter to design non-blocking collectives. However, with the current ConnectX-2 interface, the size of the task-lists that a process can post to the adapter is limited and this directly affects the scalability of collective operations, such as MPI_Alltoall. To overcome this limitation, we divide the entire Alltoall operation across multiple task-lists and we rely on a light-weight thread to post these task-lists. Since the adapter can execute the task-lists independently, the progress thread is required to be active for a very short duration, minimizing its contention with the application thread. We create a separate queue-pair *trigger-qp*, a completion-queue *trigger-cq* and a completion channel *trigger_comp_channel*, to allow a process to communicate with itself through InfiniBand's blocking progression mode. At the end of a task-list, a process enqueues a send task to itself on the trigger_qp. The offload-progress-thread posts the task-list, calls *ibv_get_cq_event* on the trigger_comp_channel and gets scheduled into a sleep state. The adapter executes the task-list and finally executes the send on the trigger_qp. This generates a network interrupt on the trigger_comp_channel, signaling the progress-thread to post the next task-list (if any).

B. Re-designing Scientific Applications to Leverage Improved MPI Library

1) Leveraging One-sided Communication in AWP-ODC: AWP-ODC spends most of its execution time to compute and exchange two variables: velocity and stress. Both velocity and stress have multiple components, each of which corresponds to a data grid. During the exchange phase, each process sends its data grid boundaries to the neighbors in all directions and similarly receives boundary data from them. The computation of each of the individual components within velocity and stress are independent of one another. However, there is a dependency across the velocity and stress components [11]. This understanding of the data dependencies and our knowledge of RDMA based one-sided designs in MVAPICH2 forms the basis of our co-designed version of AWP-ODC.

We use MPI-2 one-sided communication primitives to provide communication buffers into which neighboring processes can directly Put their data. This enables each process to complete the exchange of data without synchronizing with its neighboring processes. The only synchronization needed is before the point where the new data is used. In MVAPICH2, the Put operation is mapped directly to RDMA. This helps overlap the transfer of one component with the computation of others.

Typical commodity compute nodes have 16 to 48 cores on each node. A significant portion of communication happens intra-node which are carried out over shared memory. The shared memory communication channel requires the CPU to do the data copies. This does not allow for overlap between the communication and any computation. Kernel-assisted schemes provide better copy performance but still suffer from the lack of overlap. Technologies like I/OAT (Input Output Acceleration Technology) provide overlap using a DMA [9]. InfiniBand provides a loop-back model of communication which allows processes on the same node to communication through the network adapter. In our design, we use this channel to enable overlap. As the communication is completely hidden under useful computation, the increased latencies do not have any negative impact on the application performance.

2) Leveraging Collective Offload and Overlap in P3DFFT: The Cooley-Tukey algorithm for FFT used for 1D FFTs is computationally efficient, but the butterfly pattern of memory accesses of this algorithm makes it a challenge to scale. P3DFFT 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. There are two expensive Alltoall operations to transpose data among ROW and COLUMN (COL) communicators. Typically, the ROW communicator is mapped to cores within a node or adjacent nodes. The COL communicator spans multiple nodes. In our work, we focus on replacing the MPI_Alltoall on the column communicator with MPI_Ialltoall.

The FFT routines (forward and back) are re-structured as shown in Figure 2. The loop index j runs over the variables that need to be transformed. During the j iteration, we overlap the FFT operations and the XY (ROW) transpose for the j variable with the YZ (COL) transpose of the $(j-1)$ iteration, which relies on the MPI_Ialltoall operation.

```
1D FFT in x for V1
transpose x and y of V1
1D FFT in y for V1
Initiate y and z transpose with MPI_Ialltoall 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 y and z transpose with MPI_Ialltoall 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. 2. Algorithm for the forward transform in the redesigned multi variable, pipelined, overlapped version.

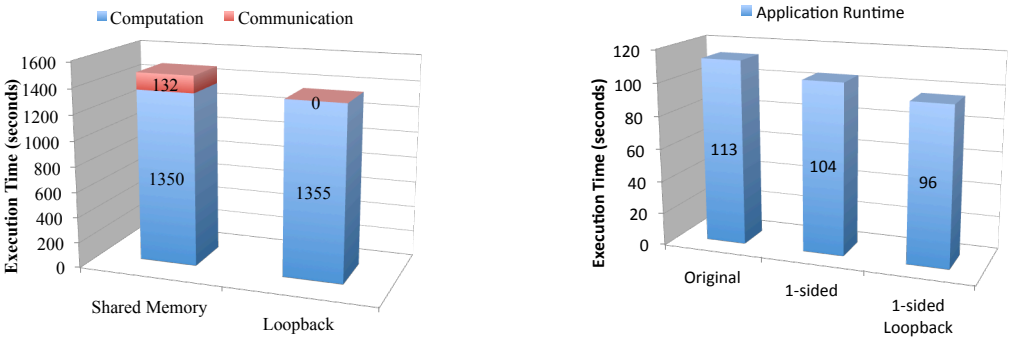
IV. Co-design Experiments

A. Improvements in AWP-ODC

We first demonstrate the effectiveness of loop-back transfers in overlapping communication and computation. The AWP-ODC application was run with 48 processes on a pair of AMD Magny-Cours machines (24 processors/node) connected using Quad Data Rate (32 Gbit/s) InfiniBand interconnect. Each process operates on 128x128x128 element data grids. Results in Figure 3(a) show a 132 second overhead when processes on a node communicate through shared memory. When the loop-back channel is used, data movement is handled asynchronously by the network adapter completely overlapping communication with computation. Therefore the communication time drops to zero while the computation time remains the same. The platform used for our next experiment, Ranger supercomputer at Texas Advanced Computing Center, is one of the largest InfiniBand clusters available for open science research. Each node on Ranger contains 16 processors and 32 GB of memory. The nodes are connected with Single Data Rate (8 Gbit/s) InfiniBand network. With a slower network, minimizing communication costs is important for efficiency of applications on Ranger. Figure 3(b) compares the performance of the original and our enhanced versions of AWP-ODC, on 8K processors. Using our overlap design and the loop-back channel, an improvement of 15% in the total application run-time is observed.

B. Improvements with P3DFFT library

This experiment demonstrates the benefit of collective communication offload to network adapter. It is run on a 512-core cluster. Each node has eight Intel Xeon cores running at 2.53 Ghz with 12 MB of L3 cache and 12 GB of memory. The nodes are connected through a Quad Data Rate (32 Gbit/s) InfiniBand network. We used the the *test_sine* kernel to evaluate the benefits of our modified P3DFFT library. In Figure 4, we compare the application run-times of the baseline blocking version with the library re-designed for overlapped collective communication using host-based and network-offload-based MPI_Ialltoall implementations. We run our test on 128 cores while varing the problem size, N, between 512 and 800. We can see that the kernel with our proposed MPI_Ialltoall consistently performs better than the one with blocking MPI_Alltoall by about 10%-23% and the kernel that uses the non-blocking MPI_Ialltoall operation using the host-based approach by about 10%-17%.



(a) Improved Overlap using Loop-back

(b) Reduced Application Run-time

Fig. 3. Performance Improvement in AWP-ODC from Design for Computation-Communication Overlap

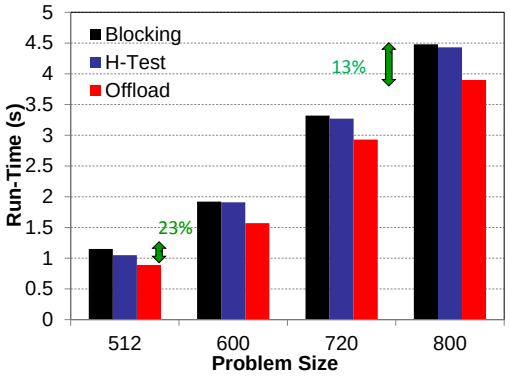


Fig. 4. Run-time comparison of the test_sine kernel with P3DFFT library

V. Conclusion

The field of High Performance Computing (HPC) is forging ahead with complex and high performing system architectures. It is predicted that by the turn of the decade, we would have surpassed Exaflop level of computing power. In order to provide balanced system performance, it is necessary to design processor, memory hierarchies and network architecture and topology in a cohesive manner. At the same time, end applications also need to be modified to fully leverage the features offered by the system. Our work is a step towards this goal. This article presented two examples of co-design of communication libraries with network architectures and applications. The MVAPICH2 MPI library was extended to incorporate support for collective offload through the recently introduced InfiniBand adapter based offload support. Using these techniques, a seismic simulation application, AWP-ODC can be sped up by 15% on 8K processor cores and three dimensional Fourier transforms can be improved up to 23% on 128 cores. This study demonstrates the effectiveness of co-designing MPI library and applications. Similar approaches can be used to co-design other components of the MPI library, components of other programming models and applications to realize the goal of exascale computing.

VI. Acknowledgments

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-0621484, #CCF-0702675, #CCF-0833169, #CCF-0916302, #CCF-0926691 and #0937842; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Appro, Chelsio, Dell, Fujitsu, Fulcrum, Microway, Obsidian, QLogic, and Sun Microsystems.

VII. Biographies

Sayantan Sur is a Research Scientist in the Department of Computer Science and Engineering at The Ohio State University. His research interests include high speed interconnection networks, high performance computing, fault tolerance and parallel computer architecture. Contact him at sayantan.sur@gmail.com.

Sreeram Potluri is a Ph.D. student in the Network-Based Computing Laboratory at The Ohio State University. His main research interests are high speed interconnects, accelerator technologies, parallel programming models and high-end computing applications. Contact him at potluri@cse.ohio-state.edu.

Krishna Kandalla is a Ph.D. student in the Department of Computer Science and Engineering at The Ohio State University. His research interests include High Performance Computing, High Speed Interconnects and MPI Collective Communication. Contact him at kandalla@cse.ohio-state.edu.

Hari Subramoni is a Ph.D. student in the Department of Computer Science and Engineering at The Ohio State University. His research interests include High Performance Computing, High Speed Interconnects, High Performance Data Transfers in InfiniBand WAN scenarios, Network Topology Aware Collective Communication and Differentiated Quality of Service in HPC. Contact him at subramon@cse.ohio-state.edu.

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. Contact her at ktomko@osc.edu.

Dhabaleswar K. (DK) Panda is a Professor of Computer Science and Engineering 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. Contact him at panda@cse.ohio-state.edu.

VIII. Contact Information

Sayantan Sur, 1300 SW Park Ave, Apt. 715, Portland, OR - 97201. Email: sayantan.sur@gmail.com Phone: 614-804-9898

Sreeram Potluri, Department of Computer Science and Engineering, 395 Dreese Laboratories, 2015 Neil Avenue, Columbus, OH - 43210. Email: potluri@cse.ohio-state.edu Phone: 614-208-5632

Krishna Kandalla, Department of Computer Science and Engineering, 395 Dreese Laboratories, 2015 Neil Avenue, Columbus, OH - 43210. Email: kandalla@cse.ohio-state.edu Phone: 614-619-3088

Hari Subramoni, Department of Computer Science and Engineering, 395 Dreese Laboratories, 2015 Neil Avenue, Columbus, OH - 43210. Email: subramon@cse.ohio-state.edu Phone: 614-961-2383

Karen Tomko, Ohio Supercomputer Center, 1224 Kinnear Road, Columbus, OH 43212, Email: ktomko@osc.edu Phone: 614-292-1091

Dhabaleswar K. (DK) Panda, Department of Computer Science and Engineering, 785 Dreese Laboratories, 2015 Neil Avenue, Columbus, OH - 43210. Email: panda@cse.ohio-state.edu Phone: 614-292-5199

References

[1] InfiniBand Trade Association. http://www.infinibandta.com.

[2] TOP 500 Supercomputer Sites. http://www.top500.org.

[3] Y. Cui, K.B. Olsen, T.H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D.K. Panda, A. Chourasia, J. Levesque, S.M. Day, and P. Maechling. Scalable Earthquake Simulation on Petascale Supercomputers. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC10), Gordon Bell Prize Finalist*, November 2010.

[4] D.A. Donis, P.K. Yeung and D. Pekurovsky. Turbulence Simulations on O(10^4) Processors. In *TeraGrid 2008*.

[5] W. Jiang, J. Liu, H. Jin, D. K. Panda, W. Gropp, and R. Thakur. High Performance MPI-2 One-Sided Communication over InfiniBand. In *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 04)*, Chicago, IL, April 2004.

[6] K. Kandalla, H. Subramoni, K. Tomko, D. Pekurovsky, S. Sur and D. K. Panda. High-Performance and Scalable Non-Blocking All-to-All with Collective Offload on InfiniBand Clusters: A Study with Parallel 3D FFT . In *International Supercomputing Conference, Hamburg, Germany*, June,2011.

[7] Olsen KB. *Simulation of three-dimensional wave propagation in the Salt Lake Basin*. PhD thesis, University of Utah, Salt Lake City, 1994.

[8] MVAPICH2: High Performance MPI over InfiniBand, iWARP and RoCE. http://mvapich.cse.ohio-state.edu/.

[9] P. Lai, S. Sur and D. K. Panda. Designing Truly One-Sided MPI-2 RMA Intra-node Communication on Multi-core Systems. In *International Supercomputing Conference (ISC 2010)*, June 2010.

[10] Parallel Three-Dimensional Fast Fourier Transforms (P3DFFT) library, San Diego Supercomputer Center (SDSC). http://code.google.com/p/p3dfft.

[11] S. Potluri, P. Lai, K. Tomko, S. Sur, Y. Cui, M. Tatineni, K. Schulz, W. Barth, A. Majumdar, and D. K. Panda. Quantifying Performance Benefits of Overlap using MPI-2 in a Seismic Modeling Application. In *International Conference on Supercomputing (ICS'10)*, 2010.

[12] T. Hoefler and J. Squyres and G. Bosilca and G. Fagg and A. Lumsdaine and W. Rehm. Non-Blocking Collective Operations for MPI-2. Technical report, Open Systems Lab, Indiana University, Aug. 2006.