High-Performance and Scalable Design of MPI-3 RMA on Xeon Phi Clusters

Mingzhe Li(✉), Khaled Hamidouche, Xiaoyi Lu, Jian Lin, and Dhabaleswar K. (DK) Panda

Department of Computer Science and Engineering, The Ohio State University, Columbus, USA
{limin,hamidouc,luxi,linjia,panda}@cse.ohio-state.edu

Abstract. Intel Many Integrated Core (MIC) architectures have been playing a key role in modern supercomputing systems due to the features of high performance and low power consumption. This makes them become an attractive choice to accelerate HPC applications. MPI-3 RMA is an important part of the MPI-3 standard. It provides one-sided semantics that reduce the synchronization overhead and allow overlapping of communication with computation. This makes the RMA model the first target for developing scalable applications with irregular communication patterns. However, an efficient runtime support for MPI-3 RMA with simultaneous use of both processors and co-processors is still not well exploited. In this paper, we propose high-performance and scalable runtime-level designs for MPI-3 RMA involving both the host and Xeon Phi processors. We incorporate our designs into the popular MVAPICH2 MPI library. To the best of our knowledge, this is the first research work that proposes high-performance runtime designs for MPI-3 RMA on Intel Xeon Phi clusters. Experimental evaluations indicate a reduction of 5X in the uni-directional MPI_Put and MPI_Get latency for 4 MB messages between two Xeon Phis, compared to an out-of-the-box version of MVAPICH2. Application evaluations in the symmetric mode show performance improvements of 25 % at the scale of 1,024 processes.

1 Introduction

The emerging co-processors such as Intel Many Integrated Cores (MICs) [2] and accelerators such as NVIDIA GPUs have been widely used to accelerate scientific applications. The features of high performance and low power consumption per watt they offer have made them as key components in modern HPC systems. In the recent Top500 list [16] (November 2014), 75 systems in total are using either accelerator or co-processor technology, including Tianhe-2, Titan, and Stampede in the top 10 supercomputers. Networking technologies, such as InfiniBand (IB) [1], have also rapidly evolved over the years to offer low latency

This research is supported in part by National Science Foundation grants #OCI-1148371, #CCF-1213084 and #CNS-1419123.

J.L. Träff et al. (Eds.): Euro-Par 2015, LNCS 9233, pp. 625–637, 2015.
DOI: 10.1007/978-3-662-48096-0_48

and high bandwidth communication to address the increasing communication requirements of current generation peta-scale applications.

Intel's Xeon Phi co-processor, based on the Many Integrated Core (MIC) architecture, packs up to 1 TFLOP of double precision performance in one chip. Several programming models, including MPI, OpenMP and others, used on multi-core architectures can run on Xeon Phi. So applications developed for multi-core systems could be easily ported to Xeon Phi. Xeon Phi also offers three usage modes for application developers: (1) Offload mode: It can be used as an accelerator to offload compute intensive regions of an application, using compiler directives. (2) Native mode: It can also be used in a many-core hosted mode to run applications. (3) Symmetric mode: It also offers a Symmetric mode where processes can be launched on both the co-processor and host.

Several scientific applications have already been successfully ported to leverage the compute power offered by Xeon Phi [7,8]. To maximize the performance of applications, several studies have optimized runtime systems for the send-recv based communication using new features available on Xeon Phi clusters. One-sided communication has been seen as a suitable model for Exascale Computing especially for applications with irregular communication patterns [5]. MPI-3 RMA provides one-sided semantics that reduce the synchronization overhead and allow overlapping of communication with computation. However, an efficient runtime support for MPI-3 RMA with simultaneous use of both processors and co-processors is still not well exploited.

To design a high-performance runtime for MPI-3 RMA, we need to cover window creation, communication and synchronization operations with existing and new channels available on Xeon Phi Clusters. For applications running in Native or Symmetric modes, different communication paths can be involved in MPI processes, such as Intra-MIC (MIC-MIC), Intra-Node (HOST-MIC), Inter-Node (HOST-MIC), Inter-Node (MIC-MIC), etc. To maximize the utilization of compute resources on MIC-based systems, it is imperative that all these operations are efficiently designed. In this work, we propose efficient and truly one-sided MPI-3 RMA designs to address the following important challenges:

– What are the bottlenecks involved in optimizing data transfers inside a node or across nodes with processor and co-processor?
– Can all the communication operations be implemented in a truly one-sided manner?
– What are the potential benefits of such a runtime design on the performance and scalability of parallel applications?

In this paper, we present a high-performance and scalable design of MPI-3 RMA on Intel Xeon Phi clusters. We also carry out a detailed analysis of our designs and evaluate them with various micro-benchmarks and applications. Our experimental evaluations show that our proposed design could reduce the uni-directional MPI_Put and MPI_Get latency for 4 MB messages from Xeon Phi to Xeon Phi by 5X. The design improves the performance of MPI-3 RMA based Graph500 by 25 % with 1,024 processes in symmetric mode. To the best of our knowledge, this is the first research work that proposes high-performance runtime designs for MPI-3 RMA on Intel Xeon Phi clusters.

2 Background

In this section, we briefly discuss MPI-3 RMA programming model. Later, we describe the communication channels available on Xeon Phi Clusters.

2.1 MPI-3 RMA

The MPI one-sided interface enables direct access to the memory of other processes through a window. Window is a memory region which can be accessed by processes in the same communicator. MPI-3 RMA introduces six communication routines to access windows. Synchronization modes provided by MPI-3 can be classified as passive (no explicit participation from the target) or active (involves both origin and target). All of these communication operations are non-blocking and are not guaranteed to complete, either locally or remotely, until a consequent synchronization operation (e.g. MPI_Win_unlock) occurs.

2.2 Communication Channels on Intel Xeon Phi Clusters

For processes running on MIC Clusters, there are three modes of inter-process communication: the shared-memory channel, the Symmetric Communication Interface (SCIF) channel, and the IB-verbs channel. SCIF is a socket-like API for communication between processes on the MIC and host within the same system [4]. The Intel Manycore Platform Software (MPSS) for MIC provides two ways of using IB verbs for communications on MIC clusters. A direct OFED communication stack is provided to support the Symmetric mode of communication on just the MIC or between the MIC and host. Alternatively, the MPSS also provides implementation of IB verbs over the SCIF API, called IB-SCIF, for intra-node communications. Our previous study showed that the current generation processor chipsets from Intel have a limited peak bandwidth when the HCA is directly accessing the accelerator memory. A Proxy-based approach is usually used to overcome this limitation [13]. Our work differentiates from previous design in that we implement MPI-3 RMA communications with truly one-sided schemes. Detailed description of the designs is presented in Sect. 3.3.

3 Proposed Designs

In this section, we present efficient MPI-3 RMA communication runtime designs for Intel MIC clusters. Figure 1 gives an overview of our proposed designs including window management, communication and synchronization operations.

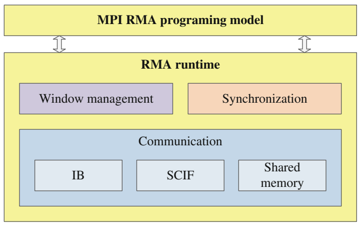


Fig. 1. Overview of MVAPICH2 MPI-3 RMA over Xeon Phi Clusters

3.1 Designs for Window Creation

In the window creation phase, each process discovers its neighbors and its location whether it is running on the MIC or HOST. The process with the lowest MPI rank of all running processes on the same MIC is assigned as the MIC leader process. The window is registered with the IB HCA and PCIe device to enable both IB verbs and SCIF communications. After registrations, all MPI processes call MPI_Allgather to get other processes' window information (window base, remote descriptor for IB network, remote offset for SCIF, etc.). Then, the MIC leader process sends all the window information to a helper process running on the host. The purpose of this helper process and this message exchange is mainly for the truly one-sided implementation which will be discussed in detail in later subsection. To provide high-performance one-sided communications, SCIF requires the source and target buffers to be 64-bit aligned. Thus, we align windows inside the library if the start address given by the user is not aligned. Further, as the registration is an expensive operation, we enhance the registration cache scheme used by MVAPICH2, to enable caching SCIF registration handlers. The whole procedure used for MPI_Win_allocate is shown in Algorithm 1. For a traditional window that is created by MPI_Win_create, each process needs to store address information of all other processes for data transfers. For newly introduced window routines in MPI-3 RMA, each process only needs to store per node address information instead of per process. The MPI-3 RMA window can also be implemented as a symmetric heap where the window base on all processes is the same. In this case, each process only needs to store one window address which can be used for all processes.

3.2 Designs for Intra-node Communication

In this subsection, we propose designs for the intra-node MIC-MIC, HOST-MIC and MIC-HOST communications. Here we focus on MPI_Get and MPI_Put operations.

Intra-node MIC-MIC Communication. For RMA communications operating on windows created by MPI_Win_create, the communication is implemented over the two-sided based shared memory channel. The origin process copies a header and the data into a shared memory region allocated during MPI_Init and the receiver process keeps polling for any incoming messages, detects the header and copies the data into the destination buffer. For RMA communications operating on a window backed by shared memory, the origin process could directly access the remote process window and finish the transfer by just one copy from the origin to the target buffer. These two designs work well for small messages. For large messages, relatively weak performance of single core on the MIC incurs high overhead with memcpy operation for data transfers. An alternative design is to use the SCIF channel to transfer large messages. The MPI_Put is mapped to *scif_writeto* which transfers the data from its source buffer to the target process's window. The MPI_Get is mapped to *scif_readfrom* to read the data from a target process's window into its own buffer.

Intra-node MIC-HOST and HOST-MIC Communication. There are two design alternatives for each of these two communication paths. One design is going through the native IB channel, and the other one is using the SCIF channel. We find that a single channel could perform the best for all message sizes, so we introduce a hybrid design which automatically chooses the best channel based on the communication pattern and message sizes. For small messages, the native IB channel always delivers better performance than the SCIF channel, so we use native IB channel for small messages in this scenario. For large message transfers, IB reads from the MIC memory will hit the bandwidth limitation introduced in [13]. This limitation adds significant overhead for MIC-HOST MPI_Put and HOST-MIC MPI_Get operations. To avoid this bottleneck, we take advantage of the SCIF channel for these two transfers. In our experiments, we find that SCIF transfers also deliver better bandwidth than HOST initialized IB transfers for large messages. So our design uses the low latency IB channel for small messages and the SCIF channel for large messages.

Algorithm 1. Details of MPI_Win_allocate

```
1  /* MPI Process running on the HOST or MIC */
2  if local_rank = 0 then
3      shm_open(size)
4      Bcast(node_comm, shm_hnd)
5  else
6      Bcast(node_comm, shm_hnd)
7      win_base ← mmap()
8  end
9  rkey ← ib_register(win_base)
10 scif_off ← scif_register(win_base)
11 /* MPI processes exchange window info */
12 Allgather(comm, {rkey, scif_off, win_base})
13 if on_mic & local_rank = 0 then
14     /* Send information to helper process */
15     Send(helper_process, {rkey, scif_off, win_base})
16 end
17 /* Helper process running on Host */
18 Recv({rkey, scif_off, win_base})
19 for k = 0; k < node_size; k++ do
20     buffer.scif_off[k] ← scif_off[k]
21     buffer.win_base[k] ← win_base[k]
22 end
```

3.3 Designs for Inter-node Communication

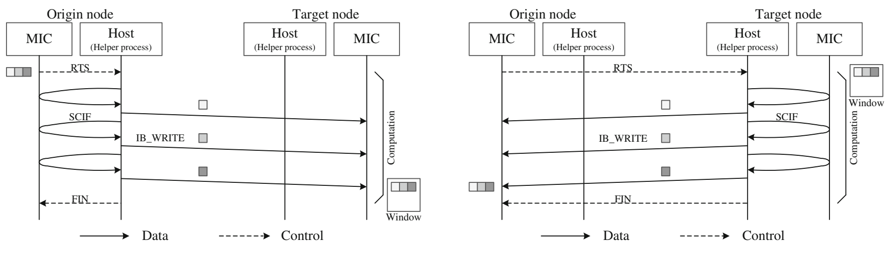
In this subsection, we present design alternatives for inter-node MIC-MIC, HOST-MIC and MIC-HOST communications.

Inter-node MIC-MIC Communication. For small messages, the IB channel could be used for data exchange between MICs sitting on different nodes. For large data transfers between MICs, the IB transfer reading from a remote MIC will hit the same bandwidth limitation. To overcome this limitation, a two step based approach is proposed. The first step transfers the data from the MIC to the HOST, and then the HOST initializes IB transfers to the remote target in the second step. In order to achieve this two-step based transfer, we need to launch a helper process on the HOST to take care of the intermediate transfer. To implement this design in a truly one-sided manner, there are several challenges:

- Which communication channel should be used for each step?
- How can the helper process directly read/write from/to the window of MPI processes?
- How to avoid an active participation of the target process, to achieve best computation/communication overlap?

For the first step, we choose the SCIF channel that always performs the best for intra-node large message transfers. There are two alternative designs for the first step: (1) The MIC process sends the first handshake message to notify the helper process. The helper process responds with an intermediate address after receiving the notification. The real data transfer starts in the third step. (2) The other way is that the MIC process sends its source address and the SCIF offset together with the notification message. When the helper process receives the data, it could directly read the source buffer via the SCIF channel. In order to reduce the participation of the source and target to the minimum and thus achieve the best overlap, we choose the second way in our design. For the second step, we choose the IB channel to transfer data from the helper process to the remote target process. There are also two alternative designs here: (1) The helper process explicitly exchanges address and other window related information with the target process. In other words, this approach requires the participation of the target and an explicit synchronization during the data transfer. (2) The helper process stores all window related information during the window creation phase as indicated in Sect. 3.1. In this way, the helper process could directly access the remote target process window. With the objective of achieving a truly one-sided design and avoid a synchronization with the target process, our solution uses the second design.

In addition to ensure the one-sided semantics, the selected paths offer the best performance as they involve the DMA SCIF read initiation from the Host to the remote MIC. However, we still need to manage data transferred in each channel to get the peak bandwidth. Congestion in either channel will hurt the overall performance. To make the overall transfer work efficiently, we transfer



(a) MPI_PutOperation (b) MPI_GetOperation

Fig. 2. Detailed designs of inter-node MIC-MIC communications

the data in a pipelined manner. The data is moved block by block from the MIC process to the helper process via PCIe bus using SCIF and from the helper process to the target process via the IB channel. As long as the helper process receives one complete block from the MIC process, it issues RDMA write to the remote target process. When the helper process finishes writing all blocks, it will send a FIN message to the origin process. This two-step pipeline based design is shown in Fig. 2(a) and (b).

Inter-node MIC-HOST and HOST-MIC Communication. For small message transfers, the native IB channel always performs the best, so our design takes advantage of the native IB channel for small messages. For large messages, an MPI_Put from an MIC to a HOST or an MPI_Get from a HOST to an MIC both will hit the bandwidth limitation if directly using the IB native channel. To avoid this bottleneck, we use the same two-step pipeline based design to stage the data to a helper process first, then initializing IB transfers. For the MIC-HOST MPI_Get and HOST-MIC MPI_Put, we use direct IB channel transfers for all message sizes.

3.4 Design for Atomic Operations

Since the IB HCA has already supported the *fetch_and_add* and *compare_and_swap* operations with 64 bits element size, our first design takes advantage of this hardware feature. We use this feature for MPI atomic operation with 64 bits element size. This design only involves the origin process for the atomic operations. For other message sizes which are not supported by the IB hardware, MPI atomic operations are implemented over two-sided based communications. When the origin process issues an atomic operation to the target process, it sends out a packet including a header and the data to the target process. After the target process receives the packet, it uses the kernel based atomic support to do the operation and returns the final result to the origin process. We do not need to consider the two-step based pipeline design here, since atomic operations only involve small messages.

3.5 Synchronization

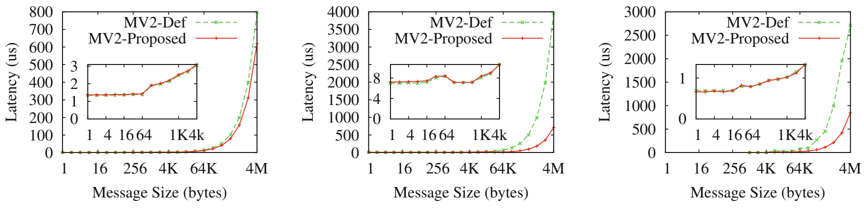
Synchronization operations ensure that window regions are ready to be accessed by other processes and all previous communication operations have been completed. In this work, we keep the same control messages and designs used for both active synchronization and passive synchronization in the Host. Our work extends current design by ensuring that the data transmission going through our proposed channels also completes when the synchronization operation returns. For transfers via SCIF channel, we call SCIF APIs to ensure that all issued SCIF operations have completed in synchronization routines. For transfers via the two-step based pipeline channel, we check the FIN message returned from the helper process to ensure that the data has been received by target process. Note that waiting for the FIN message involves only a memory polling which leads to the best performance.

4 Experimental Evaluation

In this section, we describe the experiments to evaluate the efficacy of the proposed design.

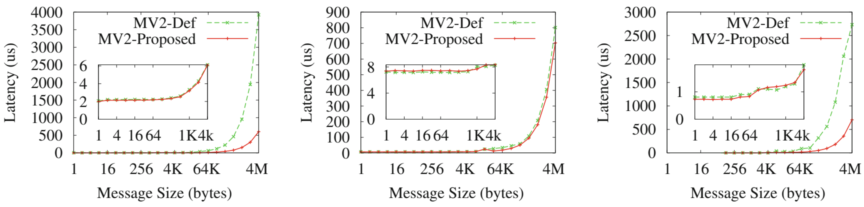
4.1 Experimental Setup

We use the TACC Stampede [15] system for experiments. Each Stampede node has a dual socket containing Intel Sandy Bridge (E5-2680) dual octa-core



(a) Host to Xeon Phi (b) Xeon Phi to Host (c) Xeon Phi to Xeon Phi

Fig. 3. Intra-node put latency



(a) Host to Xeon Phi (b) Xeon Phi to Host (c) Xeon Phi to Xeon Phi

Fig. 4. Intra-node get latency

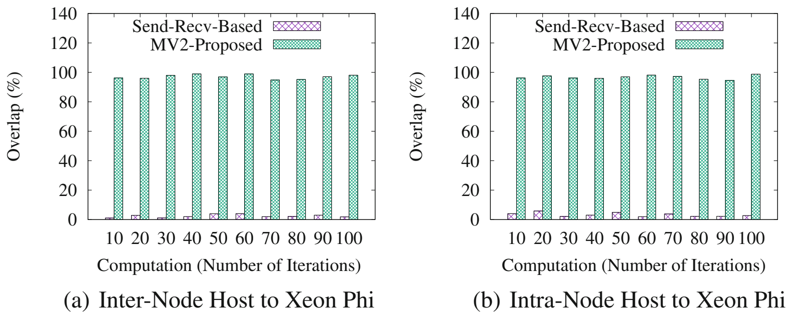


Fig. 5. Communication computation overlap evaluation

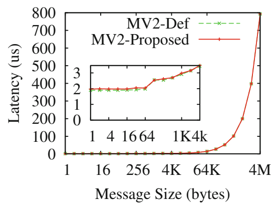
processors running at 2.70 GHz. It has 32 GB of memory, a SE10P (B0-KNC) coprocessor and a Mellanox IB FDR MT4099 HCA. The host processors are running CentOS release 6.3 (Final), with kernel version 2.6.32-431.17.1.el6.x86_64. The KNC runs MPSS 2.1.4346-16. The compiler suite used is Intel composer_xe_2013.2.146. Our designs are integrated in the popular MVAPICH2 [12] MPI library. We use OSU Micro-Benchmarks (OMB) 4.4 for evaluation performance of point-to-point communication. We then present evaluation using an RMA based LU kernel and MPI-3 RMA based Graph500 benchmark [9].

In this section, we first present experiments studying the impact of our designs for intra-node and inter-node communication on performance of point-to-point operations. 'MV2-Def' is the default version of MVAPICH2-MIC using the shared memory channel and native IB interface for MPI-3 RMA communication operations. 'MV2-Proposed' represents our proposed designs using the shared memory, SCIF and IB channels for inter-node and intra-node communications.

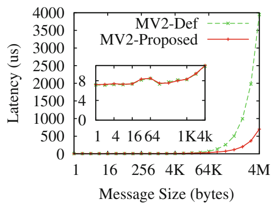
4.2 Micro-Benchmark Level Evaluation

We present the performance results of MPI_Put and MPI_Get uni-directional latency benchmarks in Figs. 3 and 4, respectively. For the intra-node MIC-MIC communication, MV2-Def uses the shared memory channel for all message sizes, whereas MV2-Proposed uses the shared memory channel for small messages and the SCIF channel for large messages. From the results, we see that the SCIF channel helps reduce the large message latency significantly. For the intra-node MIC-HOST MPI_Put and HOST-MIC MPI_Get results, MV2-Def uses IB channels for all message sizes, whereas our proposed design uses the IB channel for small messages and the SCIF channel for large messages. The HOST-MIC MPI_Get operation latency for 4 MB messages are 3949.6 and 721 μs respectively for MV2-Def and MV2-Proposed, which is a 5X reduction.

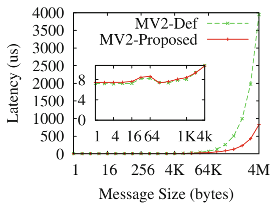
The Inter-node MPI_Put and MPI_Get performance results are presented in Figs. 6 and 7, respectively. For the inter-node MIC-HOST MPI_Put and HOST-MIC MPI_Get, MV2-Def uses the IB channel for all message sizes, which has overhead for large messages; MV2-Proposed uses the SCIF channel staging data from the MIC to the Host, then the HOST issues IB transfer to the



(a) Host to Xeon Phi

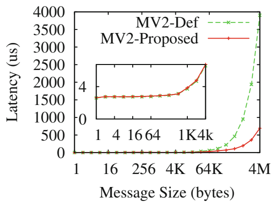


(b) Xeon Phi to Host

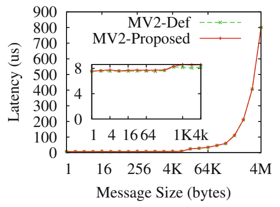


(c) Xeon Phi to Xeon Phi

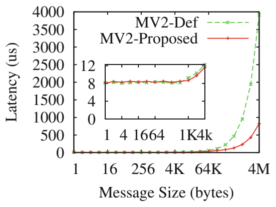
Fig. 6. Inter-node put latency



(a) Host to Xeon Phi



(b) Xeon Phi to Host



(c) Xeon Phi to Xeon Phi

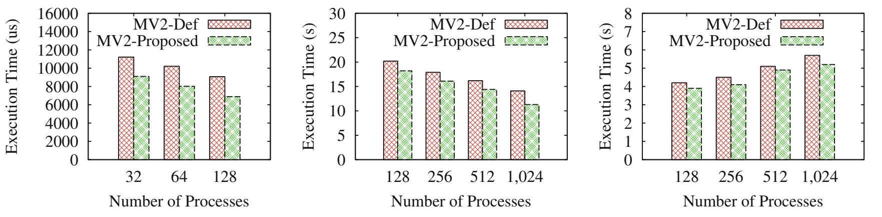
Fig. 7. Inter-node get latency

destination process. For the MIC-HOST MPI_Put of 4 MB message sizes, the latencies are 3947.9 and 700 μs for MV2-Def and MV2-Proposed, respectively, which is a 5X reduction. For the inter-node HOST-MIC MPI_Put and MIC-HOST MPI_Get, both MV2-Def and MV2-Proposed use direct IB transfer for all messages, because th Host initialized IB transfer doesn't have bandwidth limitation. For the inter-node MIC-MIC results, MV2-Def uses the IB channel for all message sizes, where MV2-Proposed uses two-step based transfers. That is why we also see big improvements in this case.

Figure 5 demonstrates the impact of the proposed design on communication computation overlap. The overlap benchmark is a two process test that measures the latency observed at the origin MPI process for MPI_Get and MPI_Win_flush on the window at the target process. During this time, the target process goes into a busy loop (mimicking computation) which is increased in each step. In Fig. 5, Send-recv-based means implementing communication over two-sided based designs, which involves both the origin and the target process in the communication. Our proposed design is a truly one-sided based implementation. By achieving near 98 % communication-computation overlap, the communication time of our proposed design is constant when the computation time increases, whereas the communication time keeps increases for the two-sided based design.

4.3 Application Level Evaluation

We use a modified version of the SPLASH LU benchmark written with MPI RMA to demonstrate the benefits of our proposed designs. This experiment



(a) SPLASHLU Native Mode (b) Graph500 Symmetric Mode (c) Graph500 Native Mode

Fig. 8. Application evaluations

uses 16 MPI processes per Xeon Phi in the native mode. The SPLASH LU benchmark does dense LU factorization. The dense 8192×8192 matrix is divided into blocks of size 128×128 each. Figure 8(a) shows the execution time of the benchmark for the proposed design from 32 to 128 processes. We observed that MV2-Proposed outperforms the MV2-Def by a factor of 24 % on 128 processes. The improvement seen by our design is due to the overlapping of the computation and communication time and efficient inter-node and intra-node Xeon Phi to Xeon Phi communication.

We also show the impact of our designs using an MPI-3 RMA based Graph500 benchmark [9] with processes running across MICs and hosts. In this experiment, we run 16 processes on each host and Xeon Phi. The results are presented in Fig. 8(b). We run the test with increasing number of processes on the MIC and Host from 128 to 1,024. We use scale 22 and edge factor 16 for the evaluation. We see that MV2-Proposed consistently delivers benefits compared to MV2-Def. The performance benefits come from our proposed design which optimizes all these communications. With 1,024 processes (512 on the host and 512 on the MIC), the execution times are 15.1 and 11.3 s, respectively, which is a 25 % improvement.

We then present the performance evaluation results of the same Graph500 benchmark for native mode. In this experiment, each node is running 16 Xeon Phi processes, and the total number of processes varies from 128 to 1,024. We present the BFS kernel time in Fig. 8(c). We can see that the proposed design performs better than MV2-Def at all system scales. MV2-Proposed performs better than MV2-Def because of two reasons: (1) The low latency and high bandwidth SCIF channel benefits intra-node large message sizes. (2) Our proposed two-step based design benefits inter-node data transfers. One thing we noticed was that the execution time increases with the increase in system size. This could mean that the Graph500 has some scalability limitations on native mode.

5 Related Work

Many researchers have explored different ways to utilize computing power of Xeon Phi. Currently, most explorations have tried to make use of the architecture's offload mode of computation [7]. Larry Meadow's work investigates the performance of WRF on the Xeon Phi using symmetric mode of computation and shows significant speedup [8]. MPICH-3.1 also has supports for MIC architecture using shared memory, TCP/IP, and SCIF based communication [11]. Direct communication between MIC accelerators across nodes is supported by DCFA-MPI [14] and Intel MPI [3]. Potluri et al. address efficient communication between processes within a single node or across different nodes for MPI two-sided communications [13]. Luo et al. [10] present studies supporting UPC on MIC architecture. Gerstenberger et al. have shown their work of implementing MPI-3 One-sided interface over RDMA networks with buffer-less protocols [6]. However, our work differentiates from previous work in that we propose truly one-sided and high-performance designs for MPI-3 RMA for symmetric and many-core hosted mode over InfiniBand networks.

6 Conclusion

In this work, we propose truly one-sided and high-performance designs for MPI-3 RMA on Intel Xeon Phi clusters. We present our designs for the window creation, communication and synchronization routines in MPI-3 RMA. We take advantage of Intel's low level communication API, SCIF, in addition to the standard communication channels like shared memory and IB verbs to efficiently support all possible data movement paths in the Native/Symmetric mode. Our proposed designs improve the point-to-point latency of intra-node and inter-node Xeon Phi to Xeon Phi uni-directional MPI_Put and MPI_Get by 5X compared to an out-of-the-box version of MVAPICH2-MIC. We also evaluate our designs with a SPLASH LU application kernel and MPI-3 RMA based Graph500. The MPI RMA based SPLASH LU benchmark indicates a reduction of 24 % at 128 processes with the native mode. The Graph500 benchmark shows improvements of up to 25 % improvement with 1,024 processes. As part of our future work, we plan to re-design MPI applications with MPI-3 RMA over Xeon Phi and evaluate the impact of our designs on a broader range of applications.

References

1. InfiniBand Trade Association, InfiniBand Architecture Specification, vol. 1, Release 1.0. http://www.infinibandta.com
2. Intel MIC Architecture. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html
3. Intel MPI Library. http://software.intel.com/en-us/intel-mpi-library/
4. XEON-PHI Software Developer's Guide. http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-software-developers-guide.pdf
5. Cantonnet, F., Yao, Y., Zahran, M., EI-Ghazawi, T.: Productivity analysis of the UPC language. In: 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), p. 254. IEEE (2004)
6. Gerstenberger, R., Besta, M., Hoefler, T.: Enabling highly-scalable remote memory access programming with MPI-3 one sided. In: Proceedings of the 2013 ACM/IEEE Conference on Supercomputing, SC 2013, Denver, Colorado, USA (2013)

7. Koesterke, L., Boisseau, J., Cazes, J., Milfeld, K., Stanzione, D.: Early experiences with the intel many integrated cores accelerated computing technology. In: Proceedings of the TeraGrid Conference: Extreme Digital Discovery (2012)
8. Meadows, L.: Experiments with WRF on intel many integrated core (intel MIC) architecture. In: Chapman, B.M., Massaioli, F., Müller, M.S., Rorro, M. (eds.) IWOMP 2012. LNCS, vol. 7312, pp. 130–139. Springer, Heidelberg (2012)
9. Li, M., Lu, X., Potluri, S., Hamidouche, K., Tomko, K., Panda, D.K.: 2014 IEEE International Conference on Scalable Graph500 Design with MPI-3 RMA. In: Cluster Computing (CLUSTER), Madrid, Spain (2014)
10. Luo, M., Li, M., Venkatesh, A., Lu, X., Panda, D.K.: UPC on MIC: early experiences with native and symmetric modes. In: 7th International Conference on PGAS Programming Models (PGAS) (2013)
11. MPICH:High-performance and Portable MPI. http://www.mpich.org/
12. Panda, D.K., Tomko, K., Schulz, K., Majumdar, A.: The MVAPICH project: evolution and sustainability of an open source production quality MPI library for HPC. In: Workshop on Sustainable Software for Science: Practice and Experiences, held in conjunction with Int'l Conference on Supercomputing (WSSPE) (2013)
13. Potluri, S., Bureddy, D., Hamidouche, K., Venkatesh, A., Kandalla, K., Subramoni, H., Panda, D.K.: MVAPICH-PRISM: a proxy-based communication framework using InfiniBand and SCIF for intel MIC clusters. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC) (2013)
14. Si, M., Ishikawa, Y.: An MPI library implementing direct communication for many-core based accelerators. In: 2012 SC Companion High Performance Computing, Networking, Storage and Analysis (SCC), pp. 1527–1528. IEEE (2012)
15. TACC Stampede Cluster. http://www.xsede.org/resources/overview
16. Top500 Supercomputing System. http://www.top500.org