MVAPICH-PRISM: A Proxy-based Communication Framework using InfiniBand and SCIF for Intel MIC Clusters [*]

Sreeram Potluri, Devendar Bureddy, Khaled Hamidouche, Akshay Venkatesh,
Krishna Kandalla, Hari Subramoni, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University
{potluri, bureddy, hamidouc, akshay, kandalla, subramon, panda}@cse.ohio-state.edu

ABSTRACT

Xeon Phi, based on the Intel Many Integrated Core (MIC) architecture, packs up to 1TFLOPs of performance on a single chip while providing x86_64 compatibility. On the other hand, InfiniBand is one of the most popular choices of interconnect for supercomputing systems. The software stack on Xeon Phi allows processes to directly access an InfiniBand HCA on the node and thus, provides a low latency path for internode communication. However, drawbacks in the state-of-the-art chipsets like Sandy Bridge limit the bandwidth available for these transfers. In this paper, we propose *MVAPICH-PRISM*, a novel proxy-based framework to optimize the communication performance on such systems. We present several designs and evaluate them using micro-benchmarks and application kernels. Our designs improve internode latency between Xeon Phi processes by up to 65% and internode bandwidth by up to five times. Our designs improve the performance of MPI_Alltoall operation by up to 65%, with 256 processes. They improve the performance of a 3D Stencil communication kernel and the P3DFFT library by 56% and 22% with 1,024 and 512 processes, respectively.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: :Concurrent Programming - Parallel Programming; C.1.3 [**Other Architecture Styles**]: :Heterogeneous (hybrid) systems; C.2.5 [**Local and Wide-Area Networks**]: :High-speed

General Terms

Design, Performance

Keywords

MIC, InfiniBand, RDMA, PCIe, Clusters, MPI

[*]This research is supported in part by National Science Foundation grants #OCI-0926691, #OCI-1148371 and #CCF-1213084.

1. INTRODUCTION AND PROBLEM STATEMENT

The emergence of accelerators such as NVIDIA GPUs and coprocessors such as Intel Many Integrated Cores (MICs) are changing the landscape of supercomputing. The unprecedented computational power per watt they offer is making them an increasingly indispensable component in modern system architectures. This is evident in the recent Top500 list [7] (June 2013) with a total of 54 systems using either accelerator or coprocessor technology, including Tianhe-2, Titan, Stampede, and Tianhe-1A in the top 10. Networking technologies, such as InfiniBand (IB) [1], have also rapidly evolved over the years to offer low latency and high bandwidth communication to address the increasing communication requirements of current generation peta-scale applications. Message Passing Interface (MPI) continues to be one of the most popular parallel programming models for developing scientific applications.

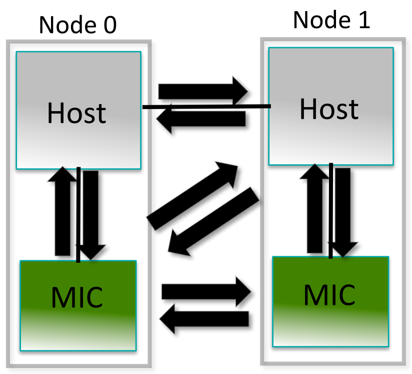


Figure 1: Symmetric Communication on Nodes with Intel MIC

Intel's Xeon Phi coprocessor [2], based on the Many Integrated Core (MIC) architecture, packs up to 1 TFLOPs of double precision performance in one chip. It runs a Linux operating system and provides x86 compatibility. It also supports several popular programming models including MPI, OpenMP, Thread Building Blocks, and others that are used on multi-core architectures. This dramatically reduces the effort in porting applications developed for multi-core systems, onto the new architecture. In the rest of the paper, we use MIC to refer to a current generation Xeon Phi coprocessor. MIC also offers a highly flexible usage model for application developers. It can be used in an *Offload* mode where application processes running on the host can offload compute to the coprocessor using compiler directives. It also offers a *Symmetric* mode where application processes can directly run on both the coprocessor and the host. This mode provides maximum control in the way applications are launched and allows application developers to take full advantage of the resources offered by both the host and the coprocessor. Several scientific applications have already been successfully ported to leverage the compute power offered by the MIC [13, 11]. However, in order to ensure that scientific applications can scale efficiently and fully harness the compute power offered by MIC clusters, communication runtimes need to be designed in a highly efficient manner to minimize the internode communication and synchronization overheads incurred by the parallel applications.

Modern supercomputing systems, such as TACC Stampede [6], are being designed to take advantage of general purpose multi-core processors, Intel MIC coprocessors, and high-speed interconnects like InfiniBand. The challenges of designing high-performance and scalable communication runtimes over high-speed networks, that address the communication requirements between general purpose processors across compute nodes, have been widely explored [4, 15, 3]. However, on emerging heterogeneous systems with MICs, we envision several additional levels of communication in the system: 1) between processes on the same MIC; 2) between processes on MIC and processes on host, within the same node; 3) between processes on MIC and processes on host, across nodes; and 4) between processes on MICs, across nodes, as demonstrated in Figure 1. Unless communication runtimes are designed to optimize these communication paths, it will not be possible for applications to efficiently leverage the computing capabilities offered by modern supercomputing systems. Previously, we explored the design space for optimizing the performance of communication within a MIC and between MIC and host, within the same node [18, 17], using the MVAPICH2 library [4].

Problem Statement: Existing MPI implementations executing on the MIC can directly rely on InfiniBand *verbs* API [1] to create network end-points and establish connections, when an IB HCA is available on the node. Hence, MPI processes running on the MIC can use IB to exchange data with other MPI processes running on a remote host or a remote MIC. Ideally, the communication performance should be similar whether the HCA is reading data from the MIC or writing data to the MIC. However, we observe that the communication costs of these operations are largely different on the state-of-the-art computing platforms like the Intel Sandy Bridge. We demonstrate these differences in Figure 2, using a node with two SNB E5-2670 processors. We observe that the peak bandwidth achieved in a network transfer operation, that involves the HCA reading from the MIC memory, is just about 962 MB/s, when the HCA and the MIC device share the same socket. On the contrary, the corresponding peak bandwidth when the HCA is writing into the MIC memory is about 5,280 MB/s. Further, we also note that if the MIC device and the InfiniBand HCA are connected to different sockets, the corresponding read/write peak bandwidths are significantly lower. Similarly, in Figure 3, we study the communication performance between MPI processes executing on the host processor and the MIC coprocessor, within the same compute node. We also compare this against the default communication path between two host processes, across the InfiniBand network. We observe that both of these communication paths offer very high communication bandwidth. Owing to this behavior, communication operations between MPI processes on such heterogeneous systems, will not be able to achieve peak network bandwidth if the underlying MPI library is based on the direct IB channel. Hence, it is critical to explore design alternatives to improve the communication performance on these emerging heterogeneous systems. These observations lead us to the following important challenges:

1. What are the challenges involved in designing a high performance MPI library that can transparently by-pass inefficient communication paths on heterogeneous MIC clusters?
2. Can we optimize communication operations involving MIC processes by efficiently utilizing resources on the host to stage the data transfers?
3. Can we further explore this design space to minimize the involvement of the host processors in implementing the data transfer operations?
4. What are the potential performance and scalability benefits of such design alternatives, across various communication benchmarks and application kernels?

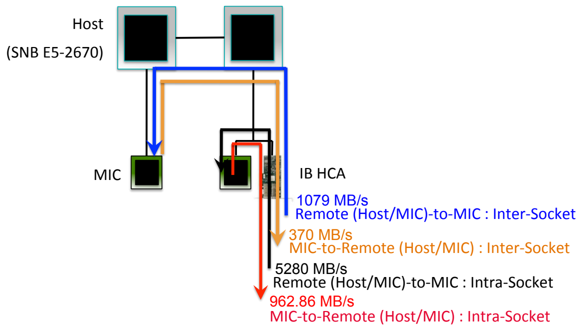

Host
(SNB E5-2670)
MIC
IB HCA
1079 MB/s
Remote (Host/MIC)-to-MIC : Inter-Socket
370 MB/s
MIC-to-Remote (Host/MIC) : Inter-Socket
5280 MB/s
Remote (Host/MIC)-to-MIC : Intra-Socket
962.86 MB/s
MIC-to-Remote (Host/MIC) : Intra-Socket

Figure 2: Peak bandwidth observed on different paths of internode communication available from/to the MIC, using IB

In this paper, we propose a novel framework *MVAPICH-PRISM*, a **PR**oxy-based communication framework using **I**nfiniBand and **S**CIF for Intel **M**IC clusters, to address the above important challenges. Our proxy-based framework can deliver low latency and high bandwidth communication performance on emerging heterogeneous MIC clusters by transparently by-passing the high overhead communication paths. We explore multiple design alternatives to efficiently utilize resources available on the host, while supporting the various execution models offered by MIC clusters. We carry out a detailed analysis of these alternatives and offer insights into their performance characteristics and associated trade-offs across various micro-benchmarks and application kernels. Our experimental evaluations show that the proposed designs improve internode latency between MICs by up to 65% and bandwidth by up to 5 times. They improve the performance of MPI_Alltoall operation by up to 65%, with 256 processes. The designs improve the performance of 3D Stencil communication kernel and P3DFFT library by 56% and 22% with 1,024 and 512 processes, respectively.

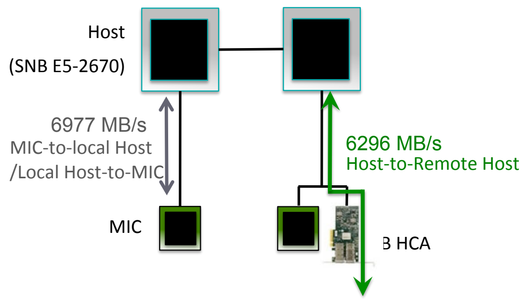


Figure 3: Peak Bandwidth observed on different paths of communication between Host and MIC using SCIF and between Hosts using IB

2. BACKGROUND

2.1 Intel MIC Architecture

Current generation Xeon Phi (SE10P) coprocessors are co-located on systems as PCI Express (PCIe) devices and are equipped with 61 processor cores that are interconnected by a high performance bi-directional ring. Each core is an in-order, dual-issue core that supports fetch and decode instructions from four hardware threads. With eight memory controllers and 16 memory channels in aggregate, a theoretical bandwidth of up to 352 GB/s is predicted. Additionally, the MIC coprocessors offer the following modes of operation for the MPI programming model [8]: 1) *Offload* mode, 2) *Coprocessor-only* mode, and 3) *Symmetric mode*.
In *Offload* mode, MPI processes are executed on one of the architectures — either the coprocessors or the host processors. The other architecture is used solely as an accelerator for MPI processes to offload computation onto. This is akin to the usage seen with most GPGPU clusters. Offload mode bears the benefit of targeting specific computation onto dedicated architectures using offload directives. Further, data movement and parallelism are managed by the compiler, with hints from the developer. In the *Symmetric* mode of operation, MPI processes uniformly span the domains of both the host and the coprocessor architectures. The developer has to explicitly manage parallelism across the two different architectures. However, he has more control on how the application executes. The *Coprocessor-only* mode is a subset of *Symmetric* mode, with all MPI processes being confined to the MIC architecture alone. In this paper we focus on both *Symmetric* and *Coprocessor-only* modes which involve MPI communication with the Xeon Phi coprocessor.

2.2 InfiniBand and Communication Channels on MIC Clusters

The InfiniBand Architecture [1] (IBA) defines a switched network fabric for interconnecting compute and I/O nodes. It is a high-speed, general purpose I/O interconnect that is widely used by scientific computing centers world-wide. The recently released TOP-500 [7] rankings (June 2013) reveal that more than 41% of the computing systems use InfiniBand as their primary interconnect.
To support the full spectrum of usage models for MIC, there are three modes of inter-process communication: shared-memory channel, SCIF, and IB-verbs based communication. For communicating processes that reside on the same MIC device, POSIX shared memory is supported, along with a multi-threaded memcpy. Alternatively, for communication operations within the same MIC, or between the MIC and host processes, within the same compute node, Intel's Symmetric Communication Interface (SCIF) generic communication API may be used. SCIF is a sockets-like API for communication between processes on MIC and host within the same system. SCIF API provides both send-receive semantics, as well as Remote Memory Access (RMA) semantics. Send-receive semantics involve both the processes, which serve as either source or destination, in the communication operation. RMA semantics define operations to register a region of memory in the user space as a window exposed for remote access. Upon registration, further data transfers can take place in a one-sided manner with just one process either reading from a window or writing into one [8].
Intel's Manycore Platform Software (MPSS) for MIC provides two ways of using IB verbs for communication on MIC clusters, as depicted in Figure 4. This allows applications to natively use MPI implementations that are built on top of InfiniBand verbs API. A direct OFED communication stack is provided to support a *Symmetric* mode of communication on just the MIC or between the MIC and the host processor. This harnesses the advantages of the *physical* InfiniBand Host Channel Adapter (HCA) for intranode and internode communication between a MIC and the host or between two MICs. To use IB directly and to enable processes on the MIC to talk with the HCA, Intel has facilitated a proxy-based approach. All privileged operations are staged through an IB proxy client, the coprocessor Communication Link (CCL) driver, residing on the host to make requests on behalf of the process running on the MIC. On completion of these privileged operations, ensuing data movement calls from the process on the MIC can be made in a direct manner to the HCA using PCIe peer-to-peer copies. Alternatively, MPSS's implementation of IB verbs over SCIF API, called IB-SCIF, may be used. This allows processes to use verbs API over a *virtual* HCA as all underlying operations are handled using SCIF. This is especially beneficial for MPI processes that reside in the MIC alone and provides ease of porting existing MPI applications to the MIC in *Coprocessor-only* mode.

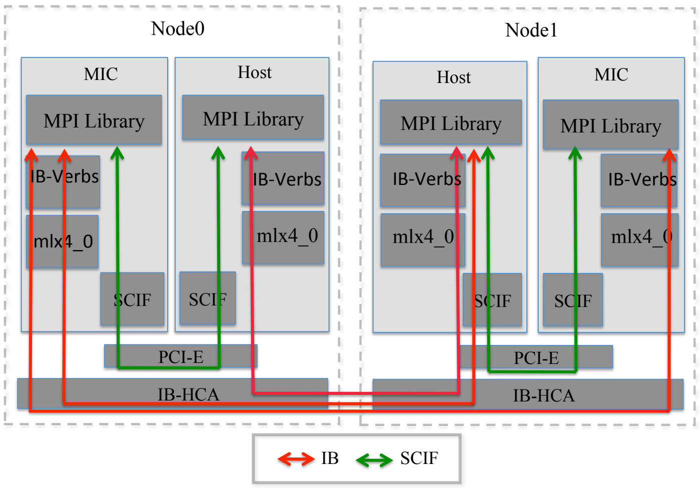


Figure 4: Communication Channels on MIC Clusters with InfiniBand

3. DESIGN

In this section, we discuss the proposed MVAPICH-PRISM framework for improving the performance of internode MPI communication operations on a cluster with Intel MIC coprocessors. The framework efficiently takes advantage of both IB and PCIe channels. It includes a design using Direct IB and multiple designs using host proxy to optimize the collection of communication paths possible in the symmetric mode.

3.1 Design Considerations

For the rest of this paper, we refer to processes located on the MIC as a *mic-process* and processes on the host processor as a *host-process*. The MPSS software stack allows access to the IB HCA from inside the MIC coprocessors through its CCL (Coprocessor Communication Link) driver. We refer to this as DirectIB. As MIC is connected to the host as a PCIe device, the data movement between MIC and the IB HCA is carried out as peer-to-peer communication over the PCIe bus. As highlighted in Section 1, current generation Sandy Bridge processor chipsets from Intel provide very limited peak bandwidth for peer-to-peer communication, especially when reads from MIC memory are involved. The InfiniBand HCA directly issues memory read/write operations when data needs to be exchanged with remote MIC or remote Host. Owing to the hardware limitation, this leads to very low communication

performance. This issue has to be addressed in order for communication libraries, and thus applications, to deliver good performance on clusters with MIC. The peer-to-peer communication limitations become even more evident when the IB HCA and the Xeon Phi are connected to different I/O Hubs (IOHs) or sockets. As clusters with multi-rail IB network and MICs are deployed, this scenario will also have to be considered. It is not clear as to which parts of these problems will be resolved in next generation of chipsets.
The MPSS stack provides the SCIF interface that implements communication over the PCIe bus, between a MIC and its Host. SCIF has been shown to deliver close to peak PCIe bandwidth performance through its RMA (Remote Memory Access) semantics. One way to work around the limitations with DirectIB is to stage data through the host, by taking advantage of the SCIF and host-to-host IB channel. However, if the process resides on the MIC, we need an agent/proxy on the host to stage its data. It is important that the communication staged through the host makes efficient use of both the PCIe and IB channels while the proxies cause minimal intervention to any application processes running on the host. We discuss several approaches to design such a proxy, as part of the MVAPICH-PRISM framework. DirectIB provides a low latency path for internode communication from the MIC while host-based staging provides a high bandwidth path. It is critical for communication runtimes to provide a hybrid design that takes advantage of both options based on the communication characteristics.

3.2 Direct IB

The OFA-CH3-IB interface in MVAPICH2 provides an implementation of MPI over IB verbs. Owing to CCL in MPSS, this interface can directly support MPI communication between processes running on a MIC and processes running on a remote Host or MIC. This path is depicted in Figure 5(a). An MPI transfer can be implemented as an IB transfer initiated by the mic-process. In the figure, we mark the initiator using a bolt symbol. Direct IB provides a low latency path for small message communication but the limitation in peer-to-peer bandwidth severely hinders the communication performance for large or concurrent messages. These limitations are expected to be more prominent when an MPI_Send is issued from a mic-process, as this involves a read by the IB adapter from the mic-process memory. In the following sections, we explore design alternatives to improve communication performance between remote MIC processes.

3.3 Passive Proxy

The first design that we consider for a proxy to handle host-staged communication from MIC aims at minimizing conflicts with any process running on the host. The proxy is only involved in setting up staging buffers on the host and is not directly involved in any communication. We achieve this by taking advantage of the RMA capabilities offered by SCIF and IB. There is an instance of the passive proxy launched on host of each node during the MPI job launch. During MPI library initialization, mic-processes establish a SCIF connection and an IB connection to the proxy running on the local host and requests buffer allocation. The proxy services an incoming request by allocating the requested buffer regions and registering it with the SCIF end-point of the corresponding process and the IB protection domain. SCIF registration returns an offset that can be used by the remote process to access the buffer using RMA functions. IB registration returns an rkey that, coupled with the virtual address of the buffer, can be used in a similar fashion. This information is returned to the requesting process. The mic-process manages this buffer region as a pool of buffers that it uses for internode communication, as described later. Processes running on a MIC can co-ordinate to allocate a common buffer region at the proxy. The buffer region can be managed as a pool of buffers and shared between mic-processes through shared memory on the MIC. It is to be noted that SCIF registration information is specific to a process end-point and hence registration requests have to be made by and serviced for each mic-process individually, although they all share the same buffer region. Every process establishes an IB connection with the passive proxies running on all remote nodes. This can be done in an on-demand fashion as has been shown in earlier work [20]. The proxy goes to sleep once it services the initial requests of all the processes and until a mic-process signals it for extended buffer allocation or for MPI library finalization.
When a mic-process has to send a message to a process on a remote MIC or host, it pulls a buffer from its local host buffer pool and writes the data into it, using an RMA *scif_writeto*. This takes advantage of the high bandwidth PCIe channel between the MIC and host. As the write completes, it sends that virtual address of the host buffer and the corresponding IB rkey to the remote process through Direct-IB. We note that the overheads of this step are minimal, considering that the amount of data transferred to the remote process is small and the operation is not bandwidth sensitive. The remote process can issue an IB RDMA operation to read data from the remote host. The remote process sends a finish message, when the read completes and the buffer is released back to the pool, at the sender process. A similar procedure is used even if the remote process is on the host. It is to be noted that the passive proxy is not involved in the communication. It is important for the communication runtime to make simultaneous use of the PCIe and IB channels to achieve maximum performance. We achieve this by pipelining SCIF and IB operations.
Figure 5(b) depicts a MIC to MIC internode transfer through the passive proxy. The sender mic-process initiates the SCIF write while the remote mic-process initiates the IB RDMA read. To be able to understand the performance characteristics of this design, it is important to analyze the performance characteristics of the channels involved. The transfer along the IB channel, as shown in the Figure 5(b), involves data movement from the host to a remote MIC. As seen in Figure 2, though this channel is not incur high overheads, its peak bandwidth is limited to 5.2 GB/s. On the other hand, the data movement from MIC to its host is issued using the SCIF channel and can reach a peak bandwidth of 6.9 GB/sec. The passive proxy-based internode communication between mic-processes is hence limited by the bandwidth offered by the IB channel. On the other hand, in our earlier work [17], we have shown how the performance of SCIF transfers differ depending on whether they are initiated on the MIC or on the host. The transfers initiated from MIC yield lower performance for medium messages, that will impact the overall performance of this design. In order to alleviate these overheads, we explore alternate designs that utilize the DMA engines that are available on the host processors.

3.4 Active Proxy

With an increasing number of cores per node and with architectures which are geared toward having a spare core, it can be viable to dedicate a core for the communication proxy. Our active proxy design considers this scenario and utilizes a core on the host to progress communication. Through this, we take advantage of the best communication channels possible for data staging through the host.
One Hop: The first variant for an active proxy is derived from the design for passive proxy presented above. However, the availability of a dedicated processor core allows the proxy to initiate and progress communication that is staged through the host. The ac-

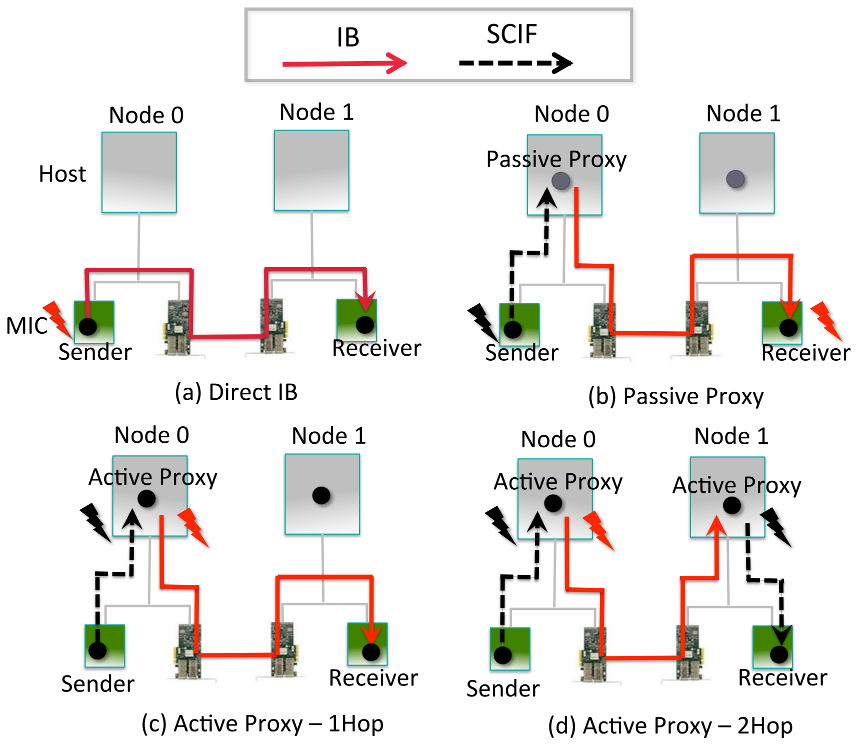


Figure 5: Comparing Design Alternatives for Inter-node MIC-MIC Communication using IB

tive proxy also takes care of allocating and managing the staged buffers on the host. In this design, there are proxies launched on the host of each node, as is the case with the passive proxy. The active proxy, after establishing SCIF connections with all the local mic-processes, listens for any incoming request messages. Like in the case of passive proxy, all mic-processes establish IB connections to proxies on the remote hosts on an on-demand basis. Each request message received from local mic-processes has information of the source and destination buffers between which the data has to be transferred. The proxy takes care of reading data from the local mic-process using SCIF and then writing the data into the remote process buffers using IB RDMA write. These two transfers are pipelined to utilize the PCIe and IB channels concurrently. The major advantage of this design, compared to that of passive proxy, is that the SCIF transfers are initiated from the host, and hence deliver higher performance for medium messages. This is depicted in Figure 5(c). This comes at the cost of sparing a dedicated core for communication progress. On the other hand, the overall performance is still bound by the bandwidth offered by the IB channel, as is the case with passive proxy.

Two Hop: The two-hop design for active proxy aims at taking advantage of the high bandwidth channels available between MIC and the host, on the same compute node. For a transfer between a mic-process and a remote mic-process, the data is first staged to the local host then to the remote host and finally to the remote mic-process. This flow is depicted in Figure 5(d). Through this, it takes advantage of the SCIF channels and the high performing host-to-host IB channel. However, it pays the price of two hops that can result in added latency and pipelining overheads. The transfers are pipelined to minimize the impact of the additional hop. The functionality of this proxy is similar to that of the 1-hop case, except that message forwarding information has to be stored at the proxies.

3.5 Discussion

Each of the designs presented above have strengths and weaknesses. DirectIB provides a low latency channel while proxy-based designs aim at achieving high bandwidth. Passive proxy requires very little involvement from host cores, but suffers from MIC initiated SCIF transfers. The 1-hop and 2-hop active proxy designs take advantage of a dedicated core and try to use the best communication channels available. 2-hop chooses better channels compared to 1-hop but at the cost of an additional stage in the transfer pipeline. Active proxy can also become a bottleneck for request processing, as the number of processes on the MIC increases. Passive proxy allows for more parallelism in data movement, as each process issues and handles its own data movement operations. Choosing the best design to use depends on the user and the application. We offer the user a choice by selecting the design as a runtime parameter. If the user does not specify an option, then the selected design depends on the application characteristics. If an application uses all the compute cores in a node, we recommend the passive design in order to avoid conflicts with the existing host processes. Otherwise, we select the active design that aims to achieve peak bandwidth performance.

4. EXPERIMENTAL RESULTS

In this section, we describe our experimental setup and evaluate the performance of the various designs in MVAPICH-PRISM presented in Section 3. We use different micro-benchmarks and application kernels in this process.

4.1 Experimental Setup

We use the Stampede supercomputing system at TACC [6] for our experiments. Each Stampede node is a dual socket containing Intel Sandy Bridge (E5-2680) dual octa-core processors, running at 2.70GHz. It has 32GB 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 2.6.32-279.el6.x86_64. The KNC runs MPSS 2.1.4346-16. Additional details of the KNC architecture are provided in Section 2.1. The compiler suite used is the Intel Composer_xe_2013.2.146. We compare the performance of internode communication using the default OFA-CH3-IB channel in MVAPICH2 that uses DirectIB with the different proxy-based designs presented in Section 3. We use OSU micro-benchmarks 4.0 for evaluation performance of point-to-point and collective communication. We then present evaluation using a 3DStencil communication kernel and the P3DFFT library.

4.2 Point-to-point Benchmarks

We present a comparison of the internode latency between processes running on MIC coprocessors in Figure 6. All the designs use DirectIB for transferring small size messages, as this offers the least latency path. As we move to larger messages, where the peer-to-peer bandwidth bottleneck comes into play, proxy-based designs outperform DirectIB. The Passive-Proxy design improves the latency for 4MByte messages by 57.5% when compared to the DirectIB-based design. The Active-Proxy-1Hop delivers better latency than the Passive-Proxy from the virtue of host initiated SCIF transfers [8]. For a medium length message, the Active-Proxy-2Hop looses out to Active-Proxy-1Hop by a small margin because of the extra hop involved. For large messages, it gains slightly from the high performance channels it uses. It delivers 65% improvement in latency compared to the DirectIB-based transfers. All the three proxy-based designs use pipelining to overlap SCIF and IB transfers.

Figure 7(a) shows a comparison of internode uni-directional MPI bandwidth using the different designs discussed in the paper. The default version of MVAPICH2 that uses DirectIB is limited by the peak intra-IOH peer-to-peer PCIe read bandwidth on SandyBridge platform which is around 1,000 Million Bytes/sec. As discussed in Section 3.3, the Passive-Proxy design achieves close to the peak IB write bandwidth from host to the remote MIC which is equal to the peak peer-to-peer write bandwidth from IB HCA to MIC. It provides a 4.6X improvement over the bandwidth offered by Direct IB, for 4MB messages. However, we observe that the bandwidth performance grows slowly with the message size which is because of the limited performance of MIC initiated DMA transfers. Active-Proxy-1Hop design overcomes the limitation in Passive-Proxy as the SCIF transfers are initiated by the proxy from the host. However, its bandwidth is also limited by peer-to-peer PCIe write bandwidth. Active-Proxy-2Hop tries in order to take advantage of the best channels available and overcome the limitations of peer-peer PCIe communication. It achieves the maximum uni-directional bandwidth performance among all the designs but does not achieve close to the peak bandwidth due to overheads in the two stage pipeline involved. The Active-Proxy-2Hop delivers up to 5.08x improvement in uni-directional bandwidth compared to DirectIB.

Figure 7(b) shows the internode bi-directional bandwidth achieved using the different designs. Passive-Proxy delivers up to 2.75X improvement compared to the DirectIB design. However, we see that this is much lower than the peak bi-directional bandwidth that the IB channel between host and a remote MIC coprocessor offers. Our analysis shows that this is because of the rate at which completions are generated by the MPSS stack for the MIC initiated SCIF writes. When a large sequence of SCIF writes are issued, we have observed alternating phases of polling and back-to-back completion of multiple transfers. This results in idle phases for one of the channels in the pipeline, thus limiting the bi-directional bandwidth achieved. Active-Proxy-1Hop, which uses host initiated SCIF transfers, avoids the issues and allows for a more balanced pipeline. It achieves up to 4.34X improvement compared to DirectIB. Active-Proxy-2Hop incurs considerable overhead due to the two hops involved. It delivers up to 3.54X improvement compared to DirectIB but loses to Active-Proxy-1Hop.

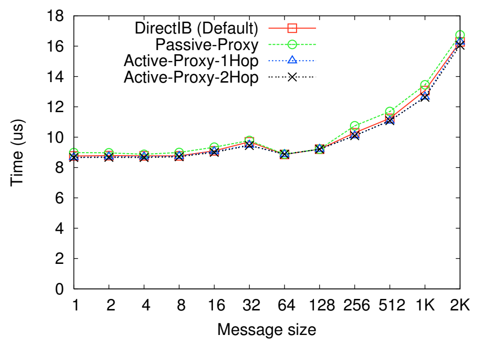
Figures 8 and 9 show the impact of our designs on the latency and bandwidth performance of communication between a MIC and a remote host. Notably, Active-Proxy-1Hop and Active-Proxy-2Hop are the same for MIC to remote host transfers as only the proxy at the node of the sending process is involved in the transfer. They perform better than Passive-Proxy as was the case with MIC to MIC transfers. They deliver close to a 55%, 4.76X, and 4.58X improvement in latency, bandwidth, and bi-bandwidth, respectively, when compared to DirectIB.

4.3 Collective Benchmarks

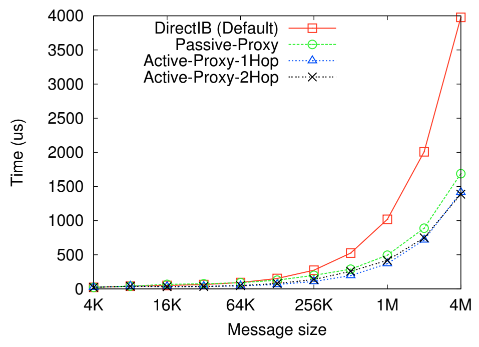
Figure 10 shows the impact of the proposed designs on the performance of MPI_Alltoall communication operation among processes running on the MIC. With four and eight processes per MIC, we see that Active-Proxy-1Hop delivers the best performance among all designs. However, we see a clear trend of Passive-Proxy performing better as the number of processes per MIC increases. In the designs with an active proxy, the proxy receives and handles any data movement into and out of the coprocessor. With the alltoall communication pattern, this processing overhead increases significantly as the number of processes per MIC increases. One way to address this is to enable multiple threads/processes to act as proxies on the host. On the other hand, in the passive proxy mode, the communication is progressed by the processes themselves and allows for an even distribution of the load. This can potentially allow for better scalability. Passive-Proxy delivers 60% and 65% improvement in latency for a 256KByte Alltoall communication with 128 and 256 processes, respectively.

We present the performance results for MPI_Gather operation using the proposed designs in Figure 11. We see that active proxy-based designs benefit the performance while the Passive-Proxy does not provide any benefits. The MPI_Gather operation uses a binomial algorithm in this experiment. With a binomial tree, there are multiple processes sending to the co-root process of the sub-trees. In the case of the Passive-Proxy, the root and the co-root processes are required to issue and progress RDMA reads to receive data from their children. On the other hand, with the active proxy-based designs, the proxy of each child process writes into the memory at the co-root, thus providing parallel progress. This results in benefits using Active-Proxy-1Hop and Active-Proxy-2Hop. We see a 37% improvement in MPI_Gather operation with 512KByte transfer on 512 processes.

Figure 12 shows the impact of our designs on the MPI_Allgather operation with a varying number of processes running on MICs. With four processes/MIC and eight processes/MIC, we do not see any improvement in performance, using the proxy-based designs. To understand this behavior, we ran the experiments with only two processes/MIC and we see considerable impact of proxy based designs on performance. For large messages, MVAPICH2 uses the ring operation to implement the MPI_Allgather operation. The communication performance of the ring exchange is bound by the slowest communication path. As we increase the number of processes executing in the MIC, we observe that the latency of the intra-MIC exchanges increases and the intra-MIC transfers determine the latency of the Allgather operation. Hence, the true benefits offered by the proxy designs are not apparent with large message MPI_Allgather operations.

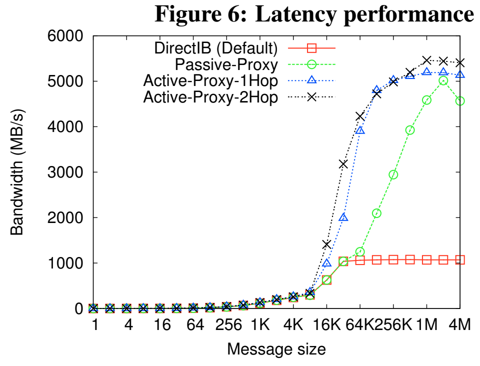


(a) Latency - Small Messages

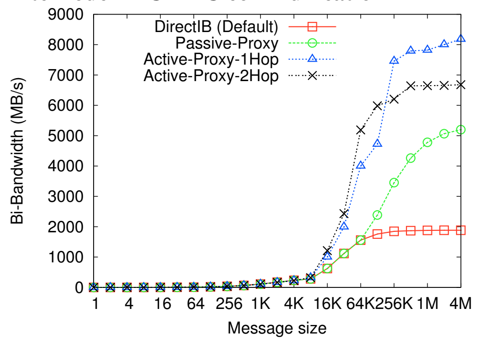


(b) Latency - Large Messages

Figure 6: Latency performance of internode MIC-MIC communication

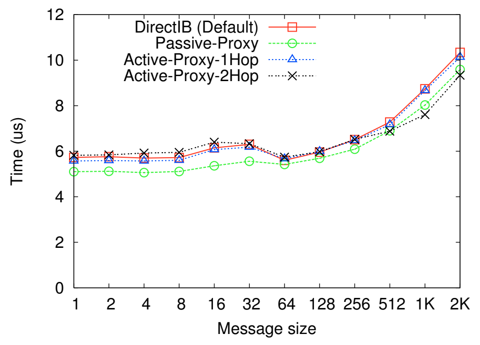


(a) Bandwidth

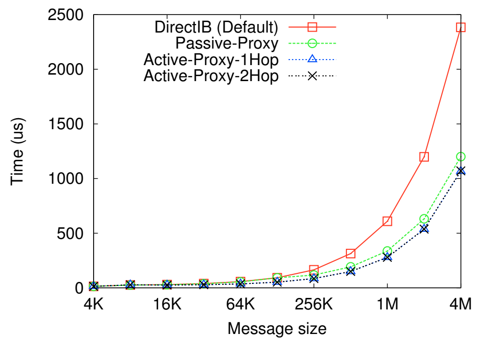


(b) Bi-directional Bandwidth

Figure 7: Bandwidth performance of internode MIC-MIC communication

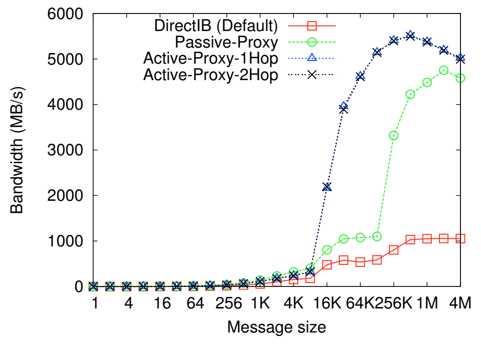


(a) Latency - Small Messages

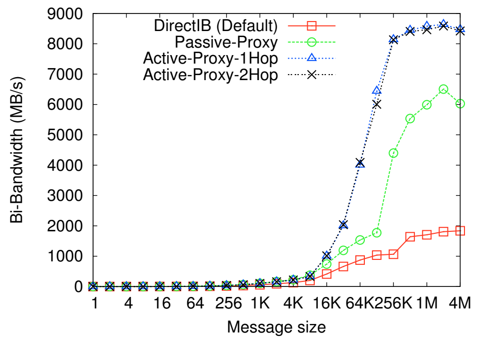


(b) Latency - Large Messages

Figure 8: Latency performance of internode MIC-Host communication

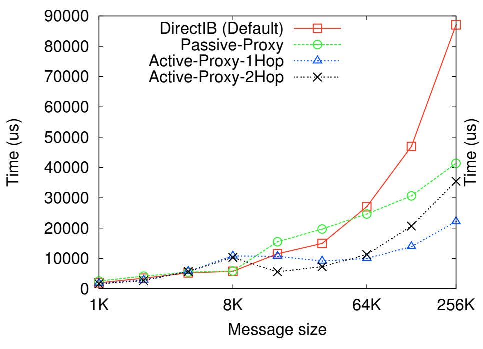


(a) Bandwidth

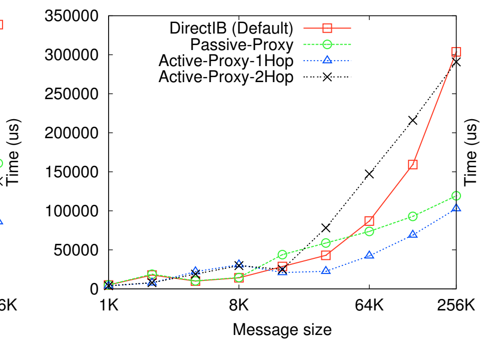


(b) Bi-directional Bandwidth

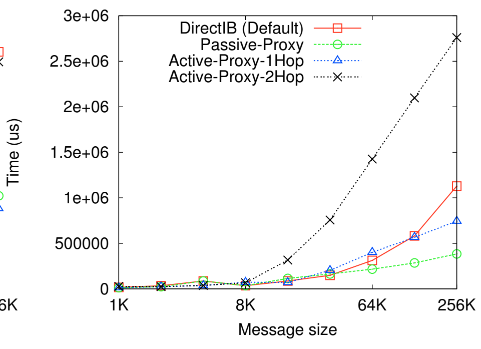
Figure 9: Bandwidth performance of internode MIC-Host communication



(a) 64 Procs - 4 Procs/MIC

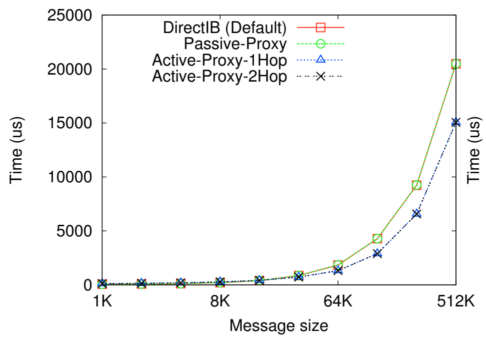


(b) 128 Procs - 8 Procs/MIC

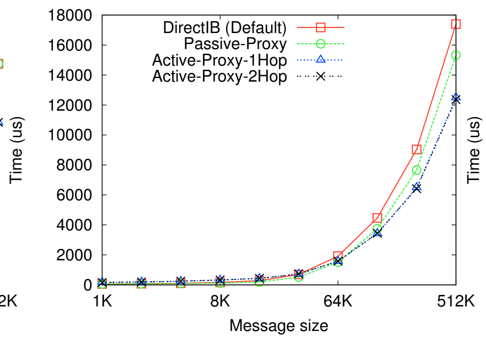


(c) 256 Procs - 16 Procs/MIC

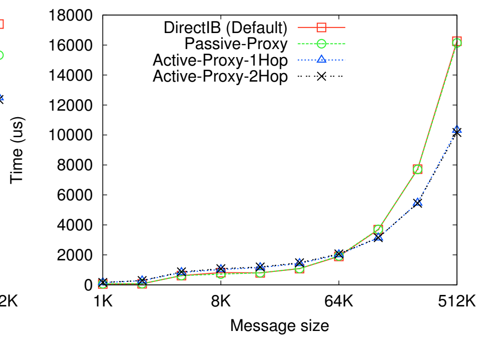
Figure 10: Performance of Alltoall communication with 16 MIC coprocessors, up to 256 processes



(a) 128 Procs - 4 Procs/MIC

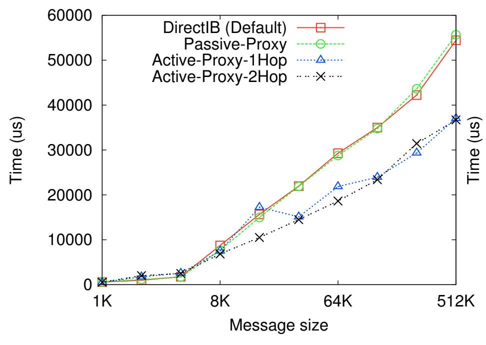


(b) 256 Procs - 8 Procs/MIC

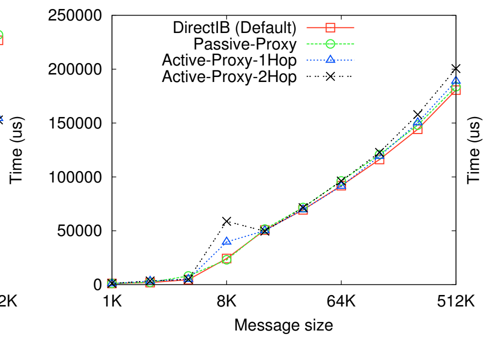


(c) 512 Procs - 16 Procs/MIC

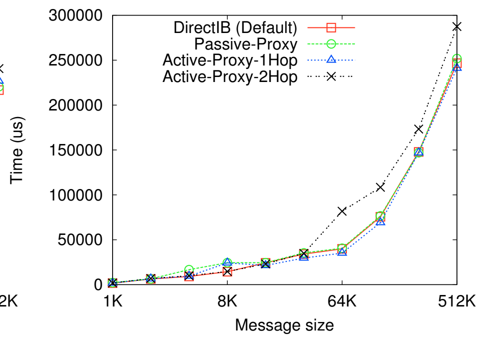
Figure 11: Performance of MPI_Gather communication with 32 MIC coprocessors, up to 512 processes



(a) 64 Procs - 2 Procs/MIC

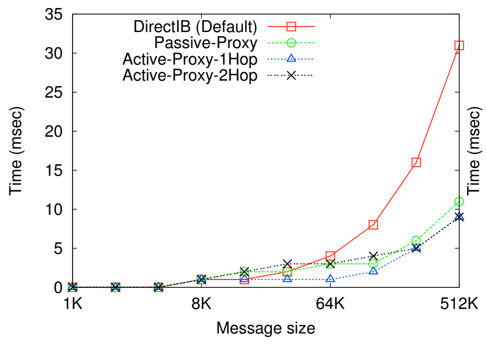


(b) 128 Procs - 4 Procs/MIC

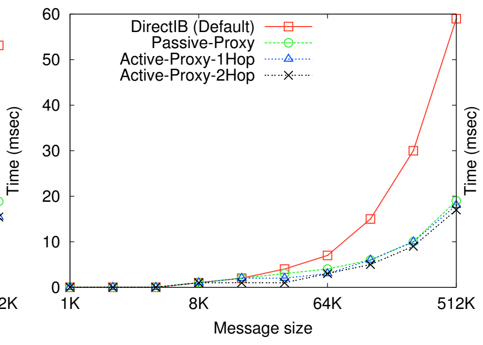


(c) 256 Procs - 8 Procs/MIC

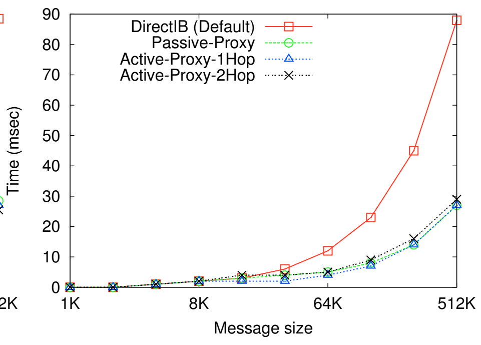
Figure 12: Performance of MPI_Allgather communication with 32 MIC coprocessors, up to 256 processes



(a) 128 Procs - 4 Procs/MIC

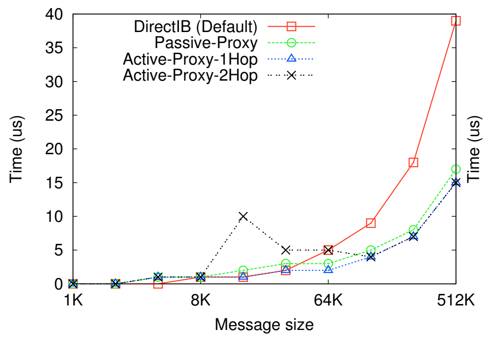


(b) 256 Procs - 8 Procs/MIC

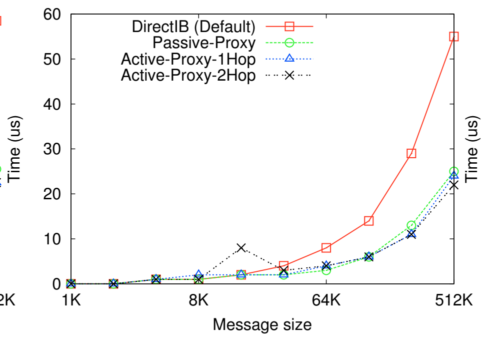


(c) 512 Procs - 16 Procs/MIC

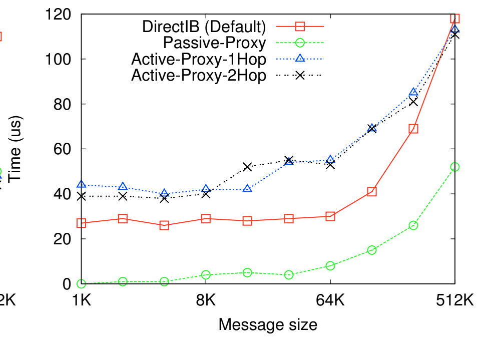
Figure 13: Performance of 3DStencil Communication Kernel with 32 MIC coprocessors, up to 512 processes



(a) 256 Procs - 4 Procs/MIC + 4 Procs/Host

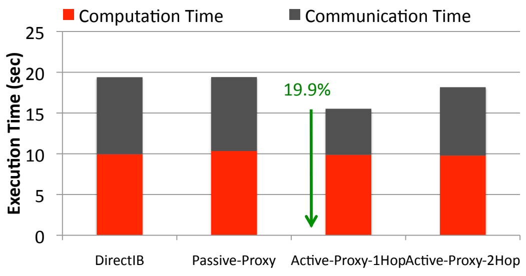


(b) 512 Procs - 8 Procs/MIC + 8 Procs/Host

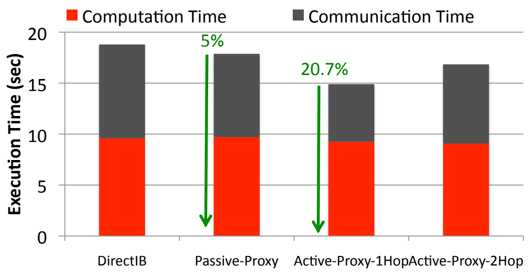


(c) 1,024 Procs - 16 Procs/MIC + 16 Procs/Host

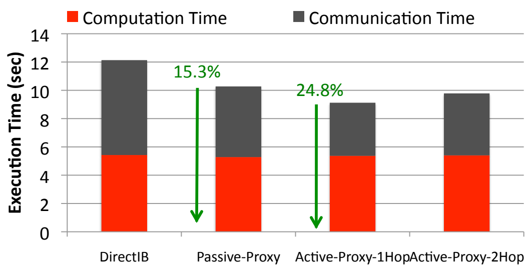
Figure 14: Performance of 3DStencil Communication Kernel with 32 MIC coprocessors and hosts, up to 1,024 processes



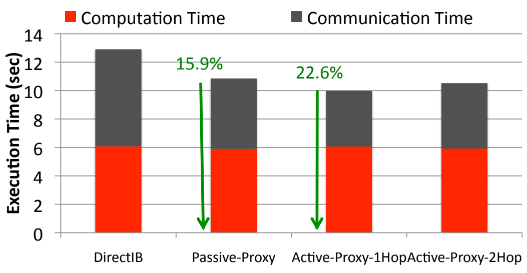
(a) 64 MICs : 256 Procs : 16 Threads/Proc : 2Kx2Kx2K



(b) 64 MICs : 256 Procs : 16 Threads/Proc : 1Kx1Kx8K



(c) 64 MICs : 512 Procs : 16 Threads/Proc : 2Kx2Kx2K



(d) 64 MICs : 512 Procs : 16 Threads/Proc : 1Kx1Kx8K

Figure 15: Performance of P3DFFT in Pure MIC configuration

4.4 Communication and Application Kernels

We use a 3DStencil communication kernel to further evaluate the impact of the proposed designs. Nearest-neighbor communication pattern is very common is high-performance scientific applications. This communication pattern is latency critical and does not cause congestion in the network. Due to these characteristics, we see that any of the proxy-based designs shows a significant improvement in the communication performance. We see an improvement of 70% in the performance with 512KByte messages. We also run the 3DStencil benchmark with processes running both on the host and the MIC. When there are 16 processes running on each host, we observe the overhead of the active proxy's intervention with the benchmark execution. Using Passive-proxy, we see an improvement of 56% in the performance with 1,024 processes on 32 nodes, for 512KByte messages.
Figure 15(a-d) shows the impact of our designs on P3DFFT, a popular library for parallel three-dimensional Fast Fourier Transforms [16]. It is written using Fortran and MPI. The version considered for our runs initializes a 3D array with a 3D sine wave, then performs a 3D forward Fourier transform and then backward transform in each iteration. We run the kernel with two different data sets on 64 MICs, with 256 and 512 processes. We observe that, with 256 processes and 16 threads/process (4,096 threads), Active-Proxy-1Hop shows an improvement of around 19.9% and 20.7% in the overall execution time with 2Kx2Kx2K and 1Kx1Kx8K data grids, respectively. As process count increases, we see that Passive-Proxy shows increasing benefits. This confirms the behavior we have seen with MPI_Alltoall, the predominent communication in P3DFFT. With 512 processes and 16 threads/process (8,192 threads), we see up to 15.9% and 22.2% improvement in the overall execution time using Passive Proxy and Active-Proxy, respectively. We see an improvement of 27.5% and 42.7% improvement in the communication time, respectively.

4.5 Summary

We have seen that the different designs presented in the MVAPICH-PRISM framework deliver better performance with different communication patterns and scenarios. Figure 16 depicts this using a three dimensional space. Active proxy-based designs work around the channel limitations but incur staging and CPU utilization. This was evident in our experiments with MPI_Alltoall in Figure 10. As we increase the number of processes, the staging and CPU utilization overheads limit the performance benefits from better channel utilization in active proxy-based designs. Passive-proxy incurs no additional CPU utilization and addresses the key limitation in the internode communication path. For latency critical applications like 3DStencil, we have seen that it delivers close to the performance of the active proxy-based designs. It delivers higher performance than active proxy-based designs when a large number of transfers are concurrently staged through the host. In such cases, the active proxy becomes a processing bottleneck. All of the designs use DirectIB for small message transfers to take advantage of the low latency path.

5. RELATED WORK

Proxy-based designs to forward communication have been widely used in literature. Most of the frameworks for cluster-wide GPU virtualization [9, 21] use proxy-based designs to move data and to execute computation on remote GPUs. While the challenges of enabling the use of remote GPUs through proxy-based designs have been explored, to the best of our knowledge, the current state-of-the-art technologies do not allow parallel applications to efficiently use MICs in the symmetric and many-core hosted modes at scale.

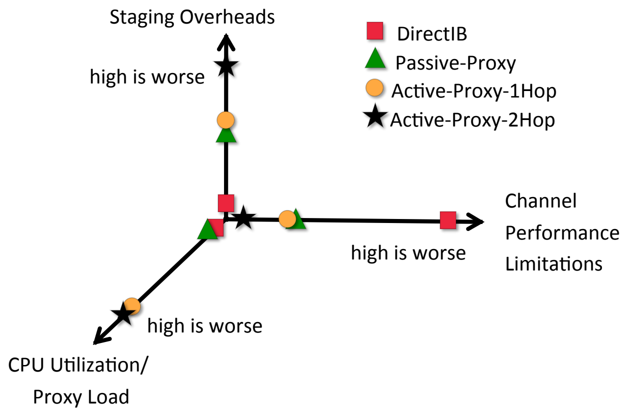


Figure 16: Performance characteristics of the proposed designs

Researchers have explored various ways of programming applications to utilize MIC's raw computing power. Being a fairly new terrain, most explorations have tried to make use of the architecture's Offload mode of computation [11, 12] . MPI-based programming is a more natural fit for large scale applications such as the Weather and Research Forecasting (WRF) model. Larry Meadow's work investigates the performance of WRF on the Xeon Phi using symmetric mode of computation [13]. MPICH2-1.5, an implementation of MPI, also has support for MIC architecture using shared memory, TCP/IP, and SCIF based communication [14]. Direct communication between MIC accelerators across nodes is supported by DCFA-MPI [19] and Intel MPI [3]. In addition, Intel MPI has a proxy-based design to overcome the PCI-e bottleneck on Sandy Bridge [5]. However, ours is the first work to discuss the design alternatives for proxy-based designs and analyse their tradeoffs. In our previous work [18, 17], we addressed efficient communication between processes within a single node where processes resided either within the same MIC card or on both the MIC and the host processor. There has also been work from other researchers to model communication performance within a SMP Xeon Phi coprocessor [10]. However, to truly harness the compute power of modern clusters with a large number of heterogeneous nodes, internode designs that allow efficient scaling becomes necessary. In this paper, we propose novel designs to optimize the internode point-to-point and collective primitives for the symmetric and many-core hosted modes, over InfiniBand network.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we designed MVAPICH-PRISM: a proxy-based multichannel design in MVAPICH2 using IB and SCIF for Intel MICs. Our design allows applications to overcome the performance bottleneck imposed by state-of-the-art Sandy Bridge systems and extract the full compute potential of the MIC coprocessors. To the best of our knowledge, this is the first design that enables application developers to efficiently use the MIC in many-core hosted and symmetric modes with high performance and scalability. Our experimental results showed that, using MVAPICH-PRISM, we improve inter-node latency between MICs by up to 65% and bandwidth by up to 5 times. MVAPICH-PRISM improves the performance of MPI_Alltoall operation by up to 65%, with 256 processes. It improves the performance of 3D Stencil communication kernel and P3DFFT library by 56% and 22% with 1,024 and 512 processes, respectively. As part of future work, we plan to explore designs for MPI collective and one-sided communication using the proposed MVAPICH-PRISM framework.

7. REFERENCES

[1] InfiniBand Trade Association, InfiniBand Architecture Specification, Volume 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] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE. http://mvapich.cse.ohio-state.edu/.

[5] OFS for Xeon Phi. https://www.openfabrics.org/images/docs/2013_Dev_Workshop/Mon_0422/2013_Workshop_Mon_1430_OpenFabrics_OFS_software_for_Xeon_Phi.pdf.

[6] TACC Stampede. http://www.tacc.utexas.edu/resources/hpc.

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

[8] 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.

[9] J. Duato, A. J. Pena, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-Orti. Enabling CUDA Acceleration within Virtual Machines using rCUDA. In *High Performance Computing (HiPC), 2011 18th International Conference on*, pages 1–10. IEEE, 2011.

[10] S. R. Garea and T. Hoefler. Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi. Jun. 2013. HPDC'13.

[11] L. Koesterke, J. Boisseau, J. Cazes, K. Milfeld, D. Stanzione. Early Experiences with the Intel Many Integrated Cores Accelerated Computing Technology. In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, November 2012.

[12] M. Deisher, M. Smelyanskiy, B. Nickerson, V. W. Lee, and M. Chuvelev. Designing and Dynamically Load Balancing Hybrid LU for Multi/many-core. In *Computer Science - Research and Development*, 2011.

[13] L. Meadows. Experiments with WRF on Intel Many Integrated Core (Intel MIC) Architecture. In *Lecture Notes in Computer Science*, volume 7312, pages 130 –139, 2012.

[14] MPICH:High-performance and Portable MPI. http://www.mpich.org/.

[15] Open MPI : Open Source High Performance Computing. http://www.open-mpi.org.

[16] D. Pekurovsky. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions. *SIAM Journal on Scientific Computing*, 34(4):C192–C209, 2012.

[17] S. Potluri, A. Venkatesh, D. Bureddy, K. Kandalla and D. K. Panda. Efficient Intra-node Communication on Intel-MIC Clusters. In *Int'l Symposium on Cluster, Cloud, and Grid Computing (CCGrid)*, May 2013.

[18] S. Potluri, K. Tomko, D. Bureddy and D. K. Panda. Intra-MIC MPI Communication using MVAPICH2: Early Experience. In *TACC-Intel Highly Parallel Computing Symposium*, April 2012.

[19] M. Si and Y. Ishikawa. An MPI Library Implementing Direct Communication for Many-Core based Accelerators. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1527–1528. IEEE, 2012.

[20] J. Wu, J. Liu, P. Wyckoff, and D. K. Panda. Impact of On-Demand Connection Management in MPI over VIA. In *Proceedings of Cluster '02*, Sep 2002.

[21] S. Xiao, P. Balaji, Q. Zhu, R. Thakur, S. Coghlan, H. Lin, G. Wen, J. Hong, and W. Feng. VOCL: An Optimized Environment for Transparent Virtualization of Graphics Processing Units. 2012.