# Intra-MIC MPI Communication using MVAPICH2: Early Experience

Sreeram Potluri*, Karen Tomko†, Devendar Bureddy*, and Dhabaleswar K. Panda*

* Department of Computer Science and Engineering
Ohio State University
{potluri, bureddy, panda}@cse.ohio-state.edu

†Ohio Supercomputer Center
ktomko@osc.edu

## Abstract

*Knights Ferry (KNF) is the first instantiation of the Many Integrated Core (MIC) architecture from Intel. It is a development platform that is enabling scientific application and library developers to prepare for the upcoming products based on the MIC architecture. Intel MIC architecture, while providing the compute potential of a many-core accelerator, has the key advantage of supporting the existing programming models, libraries and tools developed for multi-core platforms. Message Passing Interface (MPI) is the most popular parallel programming model in the scientific computing domain. MVAPICH2 is one of the most widely-used open-source implementations of MPI-2 on high performance clusters. This paper presents our early experiences using MVAPICH2 for MPI communication between processes on the MIC (Intra-MIC). We compare the performance and scalability of the standard version of MVAPICH2 with two versions of MVAPICH2 optimized for KNF. We present results using point-to-point and collective communication benchmarks.*

## I. Introduction

Multi-core processors from Intel have played a key role in pushing the realms of computing into the petaflop ($10^{15}$ calculations/second) era. As a move toward exascale ($10^{18}$ calculations/second) computing, Intel has unveiled its Many Integrated Core (MIC) Architecture [1]. The MIC architecture provides higher compute density than the current multi-core processors by packing a larger number of smaller cores that are equipped with hardware threads and wider vector units into a single MIC co-processor and has demonstrated that it can perform one teraflop ($10^{12}$ calculations/second) double precision [2].

The software development environment, which includes compilers, libraries and tools, plays an important role in enabling applications to achieve the peak performance offered by the underlying hardware. The MIC architecture, along with providing higher parallel computing capability, provides compatibility with all the existing x86 based applications, libraries and tools. This dramatically minimizes the effort required in porting existing parallel code onto the MIC hardware and leverages the man-years of engineering in x86 development tools.

High Performance Computing (HPC) is one of the key areas towards which Intel MIC architecture is targeted. Though applications, libraries and tools can run on the MIC architecture with minimal or no changes, they will have to be tuned carefully to achieve the best performance as is the case with any new hardware platform. Reducing communication overheads will remain a key factor in scaling HPC applications on clusters with the MIC co-processors. Message Passing Interface (MPI) has been the most popular programming model for developing parallel applications in the HPC domain [3]. Hence tuning MPI libraries on this new platform will impact the performance of a wide range of HPC applications.

Intel has provided Knights Ferry (KNF), a design and development kit, for software developers to port and tune their applications and libraries for future products based on the MIC architecture. KNF provides two modes of operation: offload mode, where parts of computation marked with the appropriate offload directives are automatically executed on the co-processor and native mode, where processes can be launched on the co-processor just as on a multi-core node. The later provides explicit control of the co-processor resources to the application developer. And in this mode, multiple MPI processes can run on the MIC, communicating as they would on a multi-core node.

MVAPICH2 [4] is a high-performance open source MPI implementation for multi-core clusters with modern interconnects. It is widely used in the HPC domain. In this work, we focus on optimizing MVAPICH2 for KNF. We tune and enhance designs for intra-KNF communication and compare their performance with the default version of MVAPICH2. We present results using point-to-point, multi-pair and collective communication benchmarks.

# II. Background

## A. Intel MIC Architecture - Knights Ferry

The Many Integrated Core (MIC) architecture from Intel provides a high degree of parallel processing through smaller, low-power cores. This aims to boost the performance of highly parallel applications that currently run on multi-core nodes. The first product using the Intel MIC architecture, codenamed Knights Corner, has been targeted towards high performance computing applications. The key advantage of the MIC architecture is its x86 compatibility which allows the huge repertoire of existing tools, libraries and applications to run on it, with little or no modification. Intel has provided a design and development kit called Knights Ferry, which is the first instantiation of the MIC architecture. This is enabling software and hardware developers to prepare for the upcoming products based on the MIC architecture. Each KNF consists of up to 32 Aubrey Isle cores, running at up to 1.2Ghz. Each core has 4 hardware threads, 32KB of level one instruction and data caches, and 256KB of coherent level two cache. KNF is equipped with 2 GB GDDR5 shared memory [1]. A block diagram showing the placement of cores and the memory hierarchy is depicted in Figure 1.

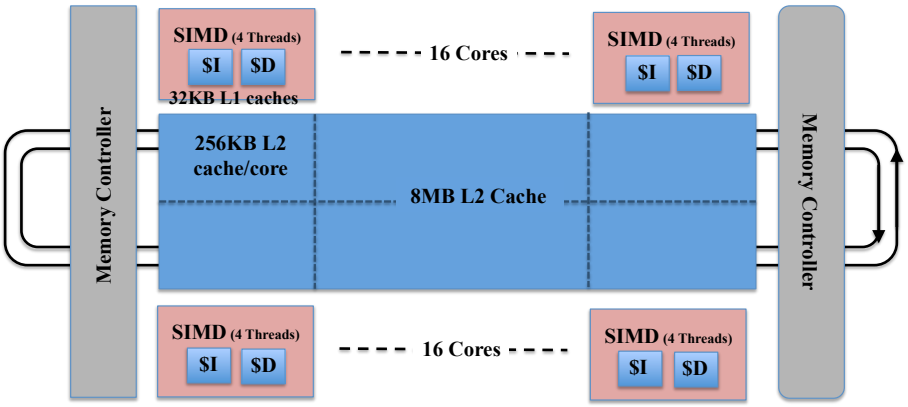



**Fig. 1.** A block diagram of core placement and memory hierarchy on the Knights Ferry co-processor

## B. MVAPICH2 MPI Library

MVAPICH2 [4] is a high-performance MPI implementation for multi-core clusters with InfiniBand, 10Gigabit Ethernet/iWARP and the emerging RDMA over Converged Enhanced Ethernet (RoCE) interconnects. MVAPICH2 provides both user-space shared-memory-based and kernel-based approaches for intra-node communication. In this paper, we focus on approaches for the user-space, in the context of intra-MIC communication on KNF. For small and control messages, each pair of processes has two circular shared memory buffers, holding messages in each direction. Each send/receive involves two copies. The sending process writes data from its source buffer into the shared buffer. The receiving process copies the data from this shared buffer into its destination buffer. Eager protocol is used for transferring these messages. For large messages, each process maintains a pool of fixed sized buffers, which is used by the process to send messages to any other process. Such a shared pool limits the memory footprint even as the number of processes increases, improves L2 cache utilization through buffer reuse and allows for pipelining of messages between processes. The protocol used for large messages is rendezvous protocol. Detailed descriptions of these designs have been provided in earlier publications [5], [6].

# III. Enhancing and Tuning Intra-MIC Communication in MVAPICH2

As outlined in Section II-B, MVAPICH2 uses multiple user-space shared memory based designs for intra-node communication. The selection among these designs depends on parameters such as message size, problem size, system architecture and others. The performance of each of these design choices is also sensitive to parameters such as the size of memory copies, the number of the communication buffers and more. This calls for tuning these designs for each platform to get maximum performance. In this paper, we have tuned MVAPICH2 for the KNF platform and we refer to this version as ”Optimized V1” in the the following sections. The standard release version of MVAPICH2 is referred to as ”Default”.

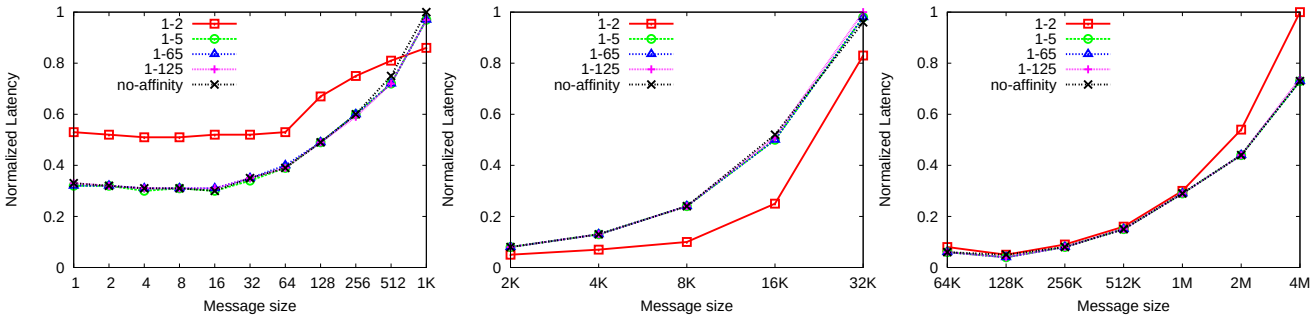
We have further enhanced our design for intra-MIC communication using low level experimental interface provided by Intel. We refer to this version as "Optimized V2". In the following sections, we compare the performance of these three versions using various benchmarks.

## IV. Experimental Evaluation

Our experimental environment is a dual socket node containing two Intel Westmere (X5680) hex-core processors, running at 3.33GHz, with 24GB of memory and a KNF co-processor connected via PCIe. The host processors are running Red Hat Enterprise Linux Server release 6.1 (Santiago), with kernel version 2.6.32-131.0.15.el6.x86_64. The KNF is a D0 1.20GHz card with 32 cores running the Alpha 9 Intel MIC software stack, with an additional pre-alpha patch. Additional details of the KNF architecture are provided in Section II-A. We have used MVAPICH2 1.8a2 for our experiments. In this section, we first present our experiments studying the impact of processor affinity on MPI latency performance. Then we compare three versions of MVAPICH2: "Default", "Optimized V1" and "Optimized V2" which are introduced in Section III. We have used OSU Micro Benchmarks (OMB) [7] and Intel Micro Benchmarks (IMB) [8] to run the various experiments. As KNF is a development platform, we only present normalized performance numbers on a scale of 0 to 1. In other words, results in each graph are normalized with respect to the largest value in that graph.

### A. Micro Benchmarks

*1) Impact of Processor Affinity:* As a first step of understanding the communication performance on KNF, we carried out an experiment to see how the placement of processes across different cores affects performance. As shown in Figure 1, KNF has 32 cores and each core has 4 hardware threads. For the discussion we give each thread on the KNF a logical rank. So ranks 1 to 4 represent the threads on the first core and ranks 5-8 represent threads on the second core and so on. We ran the MPI Latency test from OMB, binding the first process to core 1 and varying the binding of second process to different cores. We also ran an experiment without any binding. As shown in Figure 2, we do not see any impact of binding except when both the processes are bound to the same core (to different threads, 1:2 in this case). We observe higher latencies for very small messages which we believe is due to shared processor resources between the hardware threads. However, we see improved latency for medium messages, which we attribute to the shared L1 cache. For large messages, we observe a slight degradation. We observed a similar behavior with bandwidth experiments as well. For all the following experiments we have bound each process to a different core.

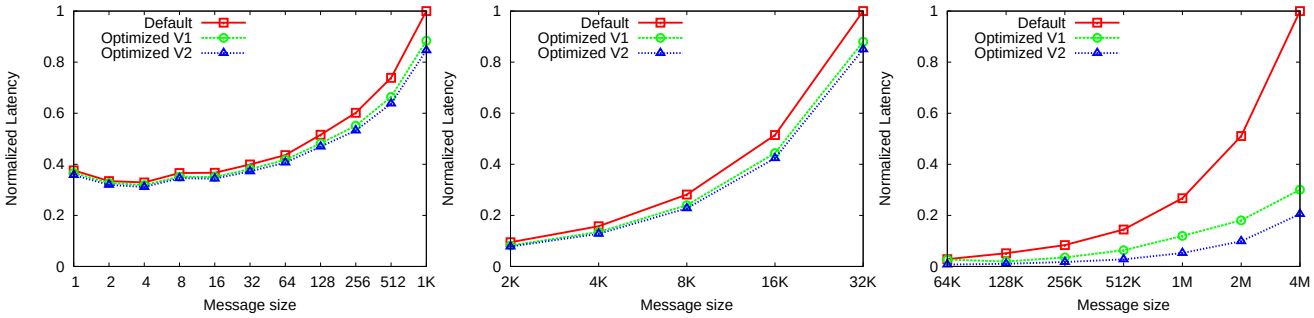



(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 2.** Impact of Processor Affinity on MPI Latency Performance

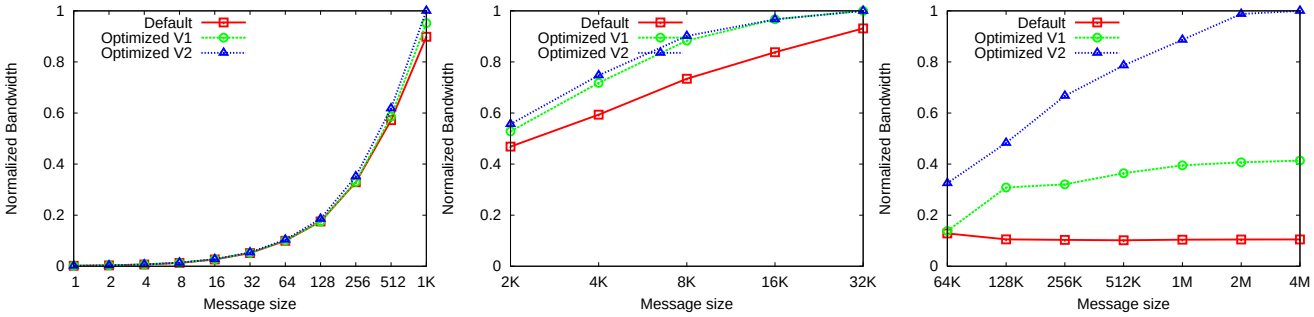
*2) Point-to-point Communication:* In Figure 3, we compare MPI latency performance using "Default", "Optimized V1" and "Optimized V2" versions of MVAPICH2. We observe that "Optimized V1" gains performance over "Default" for all message sizes due to the tuned shared memory designs in MVAPICH2. "Optimized V2" uses similar tuning parameters as "Optimized V1" but uses Intel's low level experimental interface for large message communication. We see that this benefits the MPI latency performance. We observe a similar behavior with both bandwidth and bi-directional bandwidth performance tests as well. These results are shown in Figures 4 and 5.

*3) Mulit-pair Performance:* In Figures 6 and 7, we show the results of multi-pair MPI latency test from the OMB suite. We use this to present the relative scalability of the different versions of MVAPICH2. Using 16 or 32 processes, we observe that "Optimized V1" consistently performs better than the "Default" version. "Optimized V2" consistently does better than "Optimized V1" for very large messages. However, we observe a slight overhead with 32 processes and message sizes between 64KBytes and 128KBytes. We are currently investigating this bottleneck.




(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 3.** MPI Latency Performance

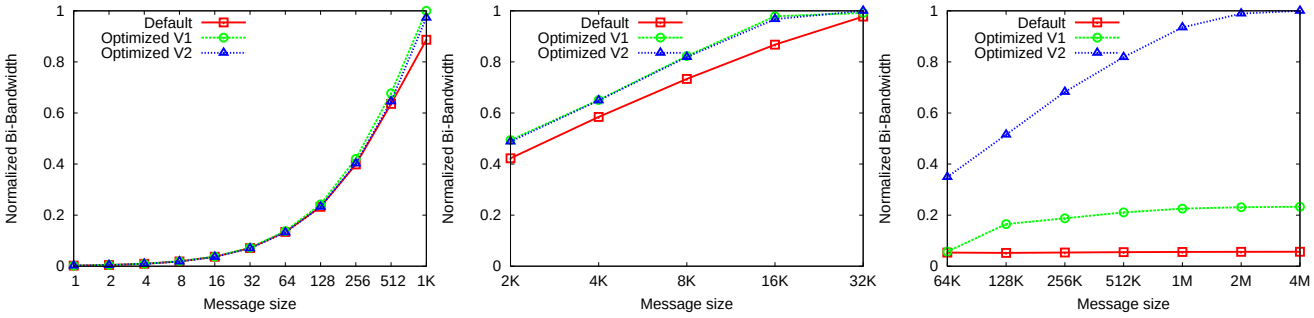



(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 4.** MPI Bandwidth Performance

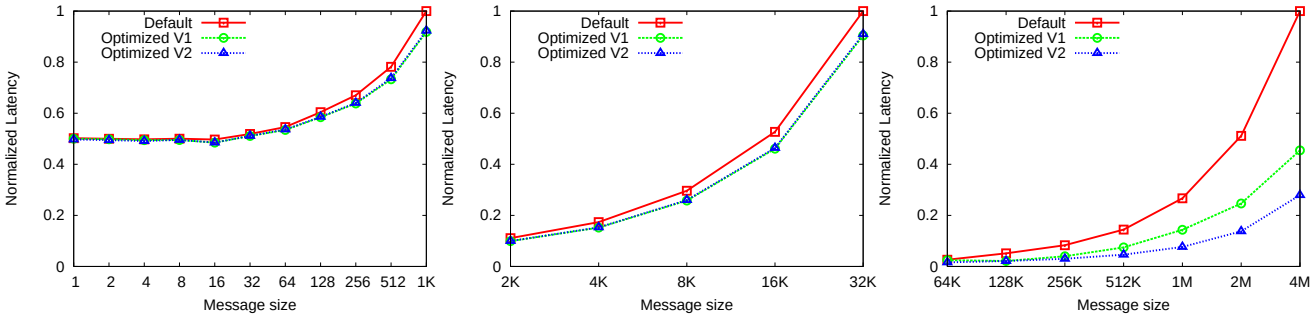
*4) Collective Communication:* MVAPICH2 by default uses collectives specific shared memory buffers. We have turned this feature off to evaluate the impact of our tuned point-to-point shared memory design in the presence of collective communication patterns. Figure 8 presents a comparison of performance of the MPI Broadcast operation across different versions. We see that both ”Optimized V1” and ”Optimized V2” perform better than ”Default” for small messages. MVAPICH2 uses a tree based algorithm for broadcasts involving these message sizes. We attribute the performance benefit to better cache utilization. ”Optimized V1” and ”Optimized V2” perform similarly except for very large messages where using lower level interface in ”Optimized V2” achieves better performance.

We present the performance comparison for MPI Scatter operation in Figure 9. We do not observe similar benefits for smaller message as in the case of Broadcast as Scatter involves sending different data to each of the peer processes and



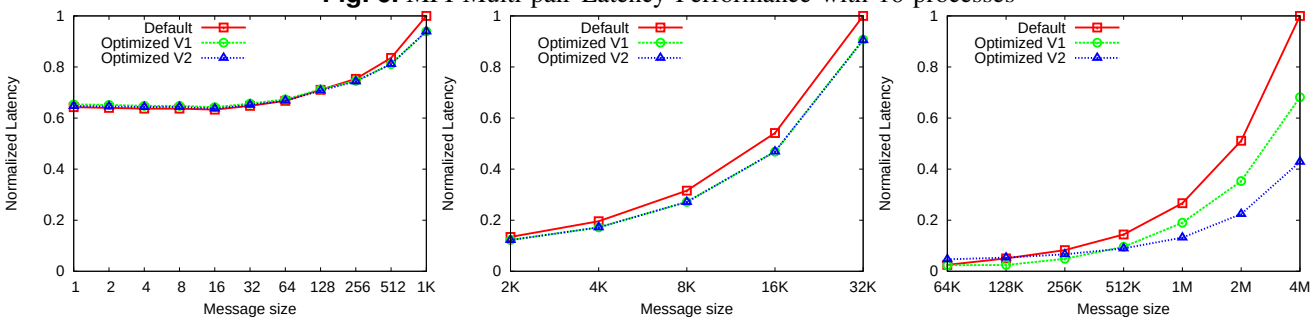

(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 5.** MPI Bi-directional Bandwidth Performance




(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 6.** MPI Multi-pair Latency Performance with 16 processes




(a) Small Messages (b) Medium Messages (c) Large Messages

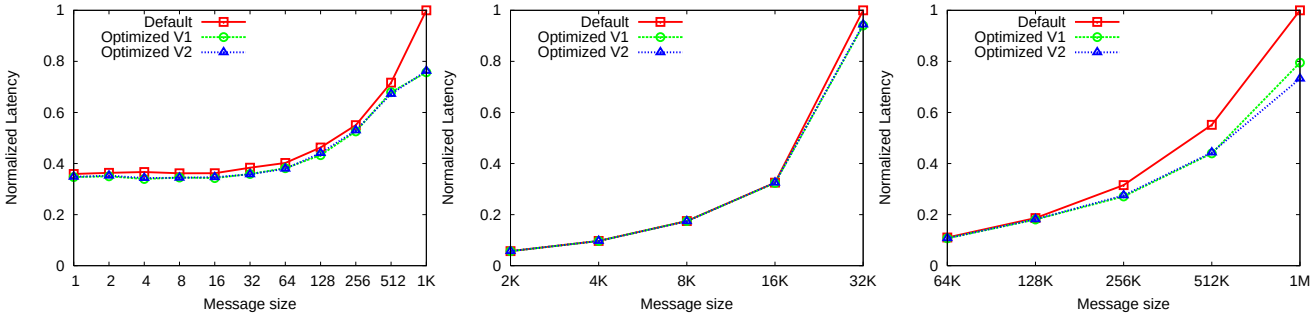
**Fig. 7.** MPI Multi-pair Latency Performance with 32 processes

hence negates the impact of better cache utilization. We observe that both ”Optimization V1” and ”Optimization V2” gain performance for large message sizes. ”Optimization V2” performs slightly worse than ”Optimization V1” for medium messages because of the limitation we have observed in Section IV-A3.

In Figure 10, we show the results for MPI Allgather operation. We have experienced an issue running MPI Allgather with the ”Optimization V2” version of MVAPICH2 and we are in the process of investigating it. ”Optimization V1” starts performing better than ”Default” from 4KByte message size where MVAPICH2 starts using a ring based algorithm for the Allgather.

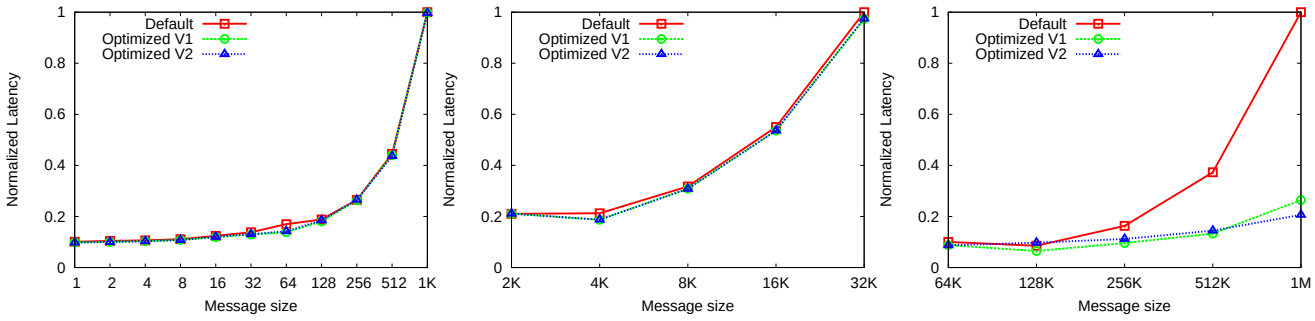
## V. Conclusion

Intel MIC architecture, while providing very high compute density, has the key advantage of supporting all the existing x86 based software with little or no porting effort. However it is important that these software are tuned to achieve the



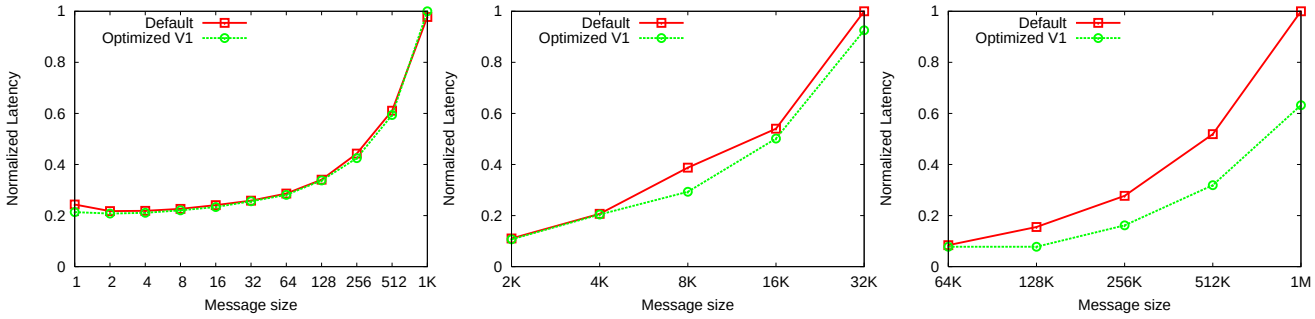

(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 8.** MPI Broadcast Performance with 32 processes




(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 9.** MPI Scatter Performance with 32 processes




(a) Small Messages (b) Medium Messages (c) Large Messages

**Fig. 10.** MPI Allgather Performance with 32 processes

best performance. In this paper, we present our experience with MVAPICH2 MPI library on Knights Ferry. We evaluate, tune and enhance the designs in MVAPICH2 for Intra-KNF communication. We plan to continue evaluating MVAPICH2 on the upcoming products based on the MIC architecture. We aim to tune and enhance our designs for communication within a MIC, between two MICs and between a MIC and a Host. We would like to evaluate, tune and enhance collective communication on the MIC architecture. We plan to study the impact of these optimizations in MVAPICH2 on end applications.

## VI. Acknowledgments

We would like to thank Timothy C. Prince, Paul J. Besl and Linda L. Kenworthy from Intel for providing us the information we needed about the Intel MIC architecture and the Knights Ferry. We would like to thank Doug Johnson and John M. Eisenlohr from Ohio Supercomputer Center for hosting and supporting the experimental environment. We would also like to thank Krishna Chaitanya Kandalla for his support in understanding the behavior of collective communication.

## References

[1] T. Elgar, "Intel Many Integrated Core (MIC) Architecture," in *2nd UK GPU Computing Conference*, December 2010. [Online]. Available: http://www.many-core.group.cam.ac.uk/ukgpucc2/programme.shtml

[2] Intel Insights at SC11. [Online]. Available: "http://newsroom.intel.com/servlet/JiveServlet/download/38-6968/Intel_SC11_presentation.pdf"

[3] *MPI-2: Extensions to the Message-Passing Interface*, Message Passing Interface Forum, Jul 1997.

[4] MVAPICH2: High Performance MPI over InfiniBand, 10GigE/iWARP and RoCE, http://mvapich.cse.ohio-state.edu/.

[5] L. Chai, A. Hartono, and D. K. Panda, "Designing Efficient MPI Intra-node Communication Support for Modern Computer Architectures," in *Proceedings of Int'l IEEE Conference on Cluster Computing*, September 2006.

[6] H.-W. Jin, S. Sur, L. Chai, and D. K. Panda, "Limic: Support for high-performance mpi intra-node communication on linux cluster." in *ICPP*, 2005, pp. 184–191.

[7] OSU Micro Benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/.

[8] "Intel MPI Benchmark," http://software.intel.com/en-us/articles/intel-mpi-benchmarks/.