Regular Paper
Journal of Computing Science and Engineering,
Vol. 12, No. 4, December 2018, pp. 170-179

Exploring the Performance Impact of Emerging Many-Core Architectures on MPI Communication

Joong-Yeon Cho and Hyun-Wook Jin*
Department of Computer Science and Engineering, Konkuk University, Seoul, Korea
jycho@konkuk.ac.kr, jinh@konkuk.ac.kr

Dukyun Nam
Division of Supercomputing, Korea Institute of Science and Technology Information (KISTI), Daejeon, Korea
dynam@kisti.re.kr

Abstract
As major architectural changes emerge to resolve the scalability issues in many-core processors, it is critical to understand their impact on the performance of parallel programming models and run-time supports. For example, the Intel Xeon Phi KNL processor is equipped with a high-bandwidth memory and deploys a mesh-based processor interconnect. In this paper, we comprehensively analyze the impact of high-bandwidth memory and processor interconnects on the message passing interface (MPI) communication bandwidth. The results show that the bandwidth of MPI intra-node communication can be improved up to 372% by exploiting the high-bandwidth memory. In addition, we show that the bandwidth of MPI inter-node communication can be improved up to 143% with optimal core affinity. Our comprehensive study provides insight into optimization of the performance of MPI communication in emerging many-core architectures.

Category: Cloud Computing / High Performance Computing

Keywords: Many-core; MPI; High-bandwidth memory; Processor interconnect; Core affinity

I. INTRODUCTION

Each computing node in contemporary supercomputers is equipped with a large number of cores allowing many parallel processes to run simultaneously. For example, an Intel Xeon Phi Knights Landing (KNL) processor contains 64–72 cores in a single processor package [1]. However, as the number of cores increases, the many-core systems experience a bottleneck when multiple cores access the shared resources, such as system bus and memory simultaneously. Since the parallel processes running in supercomputers perform the same instructions mostly, they are very likely to interfere with each other in accessing the shared resources. To resolve this problem, prominent architectural changes have been incorporated in new processors. The Intel KNL processor, for instance, is equipped with multi-channel DRAM (MCDRAM), which provides five-fold higher bandwidth than DDR4. In addition, the Intel Xeon processors, from Sandy Bridge to Broadwell, have ring-based processor interconnects to connect cores inside a processor package, while the KNL and Skylake processors use mesh interconnects.

As major architectural changes in processors emerge, it is essential to understand their impact on the performance

Open Access http://dx.doi.org/10.5626/JCSE.2018.12.4.170 http://jcse.kiise.org

Received 25 November 2018; **Accepted** 13 December 2018
*Corresponding Author

of parallel programming models and run-time supports for high-performance computing. Although there were remarkable researches to analyze the impact of many-core architectures on the performance of parallel computing [2-4], they focused on the application-level impact. Thus, they could not reveal the implications of the underlying programming models and run-time environments. The intra-node communication between parallel processes, for example, becomes more important, with the increasing number of cores installed in the system. Thus, it is important to refine the parallel programming models to exploit emerging architectural features, to enhance their practical applications.

In this paper, we aim to analyze comprehensively the performance impact of emerging many-core architectures on message passing interface (MPI) communication. MPI is a *de facto* standard for communication of parallel processes [5]. More precisely, we quantitatively measured the impact of high-bandwidth memory in KNL processor and the processor interconnects in KNL and Haswell processors on the performance of MPI communication. Our experimental results show that we can improve the bandwidth of intra-node point-to-point communication up to 372% by allocating the communication buffers to high-bandwidth memory, and reduce the latency of intra-node collective communication by 74%. Moreover, we show that carefully addressing the core affinity can improve the bandwidth of inter-node point-to-point communication up to 143% on Omni-Path and up to 39% on 40-GigE, respectively. These detailed analyses with emerging architectures provide insights for optimization of MPI communication on many-core systems.

The rest of the paper is organized as follows: in Section II, we briefly describe the architectural characteristics of the KNL and Haswell processors. In addition, we provide an overview of internal implementation of MPI communication. We analyze the impact of high-bandwidth memory on MPI intra-node communication in Section III. We evaluate the impact of processor interconnects on MPI inter- and intra-node communication with different core affinities in Section IV. We discuss the related work in Section V. Finally, we conclude the paper in Section VI.

II. BACKGROUND

In this section, we describe the architectural characteristics of Intel KNL and Haswell processors, which are used as experimental systems (Sections III and IV). In addition, we explain the internal implementation of MPI intra- and inter-node communication.

A. Many-Core Architectures

The first-generation of Intel Xeon Phi known as Knights Corner (KNC) was a coprocessor available as a PCIe card, which required a separate host processor. The next-generation Xeon Phi processor designated as KNL differs from the previous generation in that the processor is a stand-alone, bootable processor. In addition, it consists of a high-bandwidth 3D-stacked DRAM called MCDRAM, which provides 400+GB/s of bandwidth. Fig. 1 shows the overall architecture of the KNL processor. It can contain up to 36 tiles, each of which consists of two cores and 1 MB shared L2 cache. Tiles are connected by a 2D mesh interconnect. In the recent Intel Xeon Skylake processors, cores are also connected by a mesh architecture. The

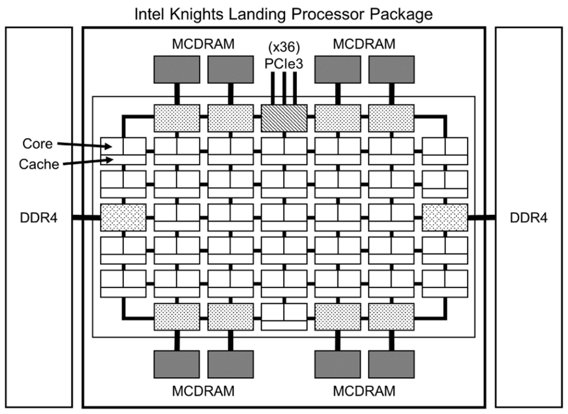

Fig. 1. Overall architecture of KNL processor with 34 tiles.

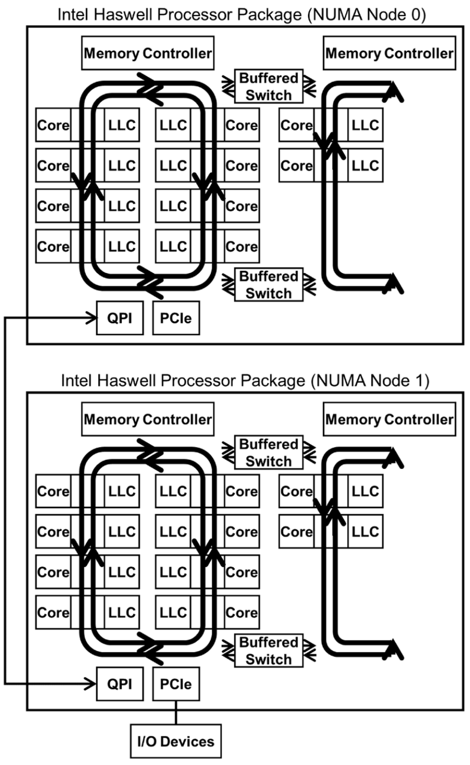
Intel Haswell Processor Package (NUMA Node 0)
Memory Controller
Memory Controller
Buffered Switch
Core
LLC
QPI
PCIe
Intel Haswell Processor Package (NUMA Node 1)
I/O Devices

Fig. 2. Overall architecture of 2-way NUMA system with two deca-core Haswell processors.

KNL processor carries 16 GB of MCDRAM, which can be configured in one of three modes: *Cache*, *Flat*, and *Hybrid*. In the *Cache* mode, MCDRAM operates transparently for applications such as cache, while used as an addressable memory module with the *Flat* mode. In the *Hybrid* mode, a portion of MCDRAM is used as addressable memory and the rest is used as cache.

Among the Intel Xeon processors ranging from Sandy Bridge to Broadwell, the last level cache (LLC) is divided into different slices, each of which is associated with each core. The cores and the LLC slices are connected by ring-based interconnects as shown in Fig. 2. This figure depicts our Haswell-based experimental system, where two deca-core processor packages are installed (i.e., total 20 cores) on each machine. These two processor packages (i.e., NUMA nodes) are connected through QuickPath Interconnect (QPI). Inside a Haswell processor package, a ring-based interconnect is connected up to eight cores to reduce the latency and mitigate the bandwidth constraints between cores. Thus, the deca-core processor package in Fig. 2 comprises two ring-based interconnects: one for eight cores and the other for two cores. Communication between these two rings occurs via a buffered switch.

B. MPI Communication

In general, MPI implementations provide two different communication modes for different message sizes: *Eager* and *Rendezvous*. In the *Eager* mode, the sender can immediately send messages to the receiver assuming that buffers for the messages are ready at the receiver side. Since the receiving process may not wait for a message (i.e., has not registered a receiving buffer yet), the MPI implementations provide internal buffers to save the received messages causing an additional copy of data for every message. In the *Rendezvous* mode, the messages received are directly moved into the destination buffers without intermediate data copies; however, this mode requires agreement between the two processes by exchanging the control messages to ensure that receiving buffers are registered before sending messages. In general, the *Eager* mode is used for small messages, without a significant copy overhead, while the *Rendezvous* mode is useful for large messages. For example, MVAPICH2 provides a configurable threshold known as MV2_SMP_EAGERSIZE to select one of these modes based on the message size [6].

Regarding intra-node communication, MPI implementations provide different data paths for *Eager* and *Rendezvous* modes. A well-known mechanism for *Eager* mode entails message delivery through an intermediate buffer shared between processes running on the same machine, where the message is internally copied twice: first, from the user buffer to the shared buffer, and then, from the shared buffer to the destination buffer [7-9]. This shared-buffer-based intra-node communication provides higher bandwidth and lower latency than the NIC-based loopback that requires messages to traverse the I/O bus twice. However, the shared-buffer-based communication is inefficient for large messages, because it consumes processor resources for copying messages. To reduce this copy overhead and save the processor resources, the memory-mapping-based mechanism is used for *Rendezvous* mode [10-12]. In this mechanism, the source buffer is directly mapped into the virtual address space of the destination process. Therefore, the message can be copied directly from the source to the destination buffer. This memory-mapping-based communication can reduce the number of copies into one. However, it is not efficient for small messages due to the larger memory mapping overhead compared with the copy overhead of small messages. Thus, MPI implementations use the shared-buffer-based mechanism for small messages (i.e., *Eager* mode), and the memory-mapping-based mechanism for large messages (i.e., *Rendezvous* mode).

The MPI implementations for inter-node communication vary depending on the type of interconnection network that is targeted. The special-purpose interconnection networks, such as InfiniBand and Omni-Path, provide a lightweight user-level communication library. Since this thin library bypasses the operating system in the communication path and deletes an intermediate data copy, it has very low communication overheads. In general, the MPI implementations support the *Eager* and *Rendezvous* modes for inter-node communication over these special-purpose interconnection networks. The MPI implementations allocate internal buffers for inter-node communication in *Eager* mode during the initialization phase. On the other hand, the general-purpose, Ethernet-based interconnection networks use TCP/IP that always saves the messages to the in-kernel intermediate buffers on the receiver side, regardless of the readiness of the receiving process. Thus, MPI needs neither intermediate buffers of the *Eager* mode nor handshaking of the *Rendezvous* mode. Consequently, MPI implementations over general-purpose interconnection networks do not need to be distinguished between *Eager* and *Rendezvous* modes.

III. IMPACT OF HIGH-BANDWIDTH MEMORY

In this section, we analyze the impact of high-bandwidth memory on MPI intra-node communication. Toward this end, we allocate communication buffers to MCDRAM in KNL processor and measure the communication bandwidth and latency.

A. Buffer Allocation

The user buffers can be allocated specifically to MCDRAM by using the *memkind* library [13], which is the heap allocator developed by Intel for Xeon Phi processors. Before allocating buffers to the high-bandwidth memory, the MEMKIND_HBW_NODES environment variable must be set to the memory node ID assigned to MCDRAM. In our experimental system, the memory node ID of MCDRAM was 1, while that of DDR4 was 0. The library provides interfaces, such as hbw_posix_memalign() and hbw_malloc() that allocate a buffer to the high-bandwidth memory specified by MEMKIND_HBW_NODES. We modified the OSU-micro-benchmarks to selectively allocate the user buffers to high-bandwidth memory.

The MPI implementations provide shared buffers internally for intra-node communication in *Eager* mode as described in Section II-B. MVAPICH2 initializes the shared memory areas for small messages by allocating a virtual file in /dev/shm. This directory is a temporary file system (i.e., *tmpfs*) that resides in DDR4. Thus, we also modified MVAPICH2 to facilitate mounting of MCDRAM as a *tmpfs* into the /dev/shm/mcdram directory and create a new virtual file for shared memory.

We measured the performance of intra-node point-to-point and collective communication on an Intel KNL 7250 machine comprising 68 cores and 16 GB MCDRAM. The MPI implementation used was MVAPICH2 version 2.2 and the operating system installed was Linux (kernel version 3.10.0).

B. Point-to-Point Communication

We used OSU-micro-benchmarks to measure the bandwidth and latency of intra-node point-to-point communication. We used a non-cached memory buffer for each communication iteration in micro-benchmarks. The experiments were conducted for 2, 34, and 68 processes. In the experiments, we ran processes of the same pair (i.e., connection) on the same tile. We compared three cases in each graph. The first case allocates all communication buffers to DDR4 (*Default*). In the second case, the shared buffers for *Eager* mode are allocated to MCDRAM, while the user buffers are in DDR4 (*MCDRAM+DDR4*). We also consider the case in which all communication buffers reside in MCDRAM (*MCDRAM-only*).

Fig. 3 shows the measurement results of point-to-point communication bandwidth for small and medium messages, where the messages are copied to/from the shared memory because the threshold to switch from *Eager* to *Rendezvous* was 64 kB. Fig. 3(c) shows that *MCDRAM-only* improved the bandwidth up to 372% for medium messages compared with *Default*. However, as the number of processes (i.e., number of point-to-point connections) is reduced, the improvement rate is also reduced as shown in Fig. 3(b) and 3(a) because MCDRAM is beneficial in terms of bandwidth. Thus, buffer allocation to MCDRAM in the presence of a large number of connections has advantages. Moreover, it is notable that both user buffer and shared memory need to be in MCDRAM to yield the best performance; otherwise, the performance gain is very limited, as shown by *MCDRAM+DDR4*. Fig. 4 shows the point-to-point communication latency for small and medium messages. As the discussion above indicates, the latency for small messages does not benefit from MCDRAM, because the latency tests involving small and medium messages do not generate bandwidth-sensitive communication load.

Fig. 5 shows the point-to-point communication bandwidth for large messages. In these experiments, the shared memory is not involved in communication, so we do not consider the *MCDRAM+DDR4* case. Fig. 5(c) shows that *MCDRAM-only* improves the bandwidth up to 366% compared with *Default*. We found that the improvement rate is reduced according to the number of processes. Fig. 6 shows the point-to-point communication latency for large messages. Unlike Fig. 4, Fig. 6 shows that *MCDRAM-only* can reduce the latency up to 73% compared with *Default* because the latency tests are sensitive to bandwidth under increased message size.

C. Collective Communication

We measured the latency of all-to-all collective commun-

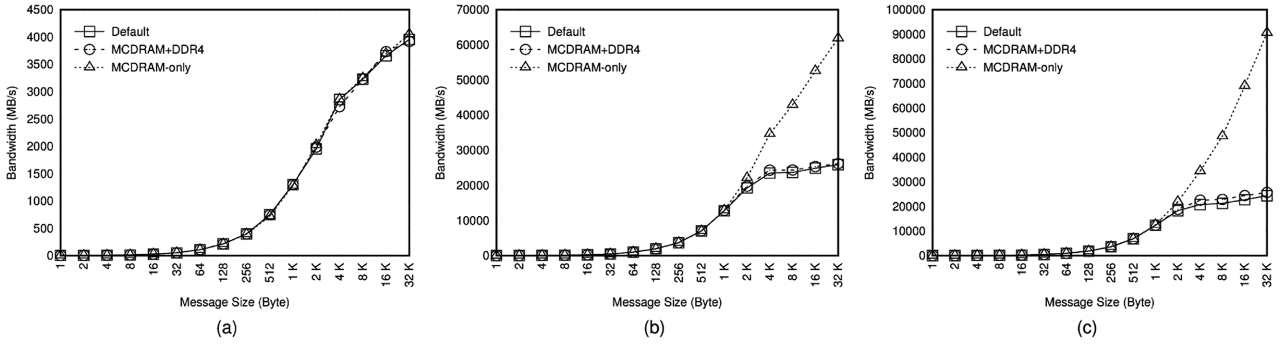

Fig. 3. Point-to-point communication bandwidth for small and medium messages. (a) Two-processes, (b) 34-processes, and (c) 68-processes.

ication, which is the popular but most time-consuming collective operation. Fig. 7 shows the measurement results of all-to-all latency. Fig. 7(c) shows that *MCDRAM-only* reduces the latency up to 74% compared with *Default*. Such performance improvement is decreased for a smaller number of processes, because MCDRAM facilitates bandwidth, and the large number of processes induces a bandwidth-sensitive communication load.

D. Implications

The *numactl* command [14] is the easiest approach to utilize high-bandwidth memory in the KNL processor and provide a transparent run-time environment for applications. However, the memory areas, the location of which does not affect the performance significantly, also consume high-bandwidth memory. To save high-bandwidth memory while ensuring comparable performance with the *numactl* case, it is highly desirable to selectively locate only the performance-sensitive buffers into the high-bandwidth memory.

In our previous study [15], we modified MPI_Alloc_mem() so that it can decide the location of the buffer (i.e., MCDRAM or DDR4) based on the buffer size and

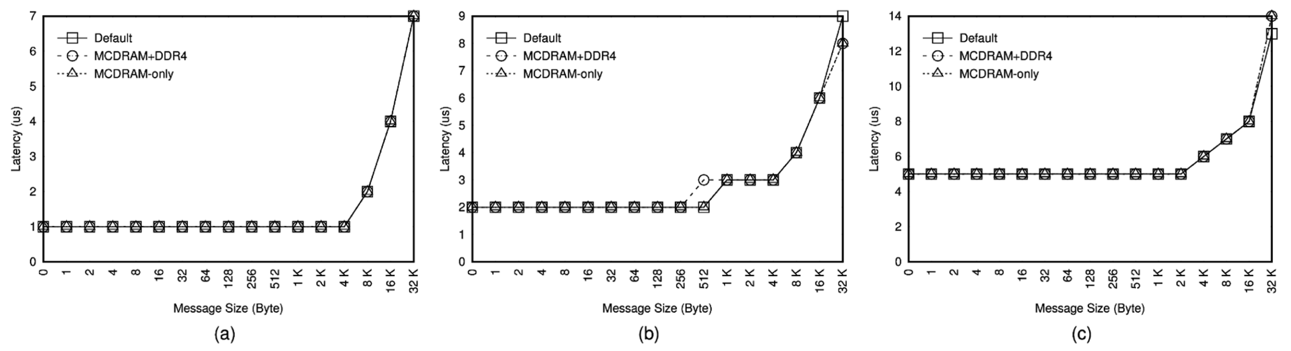

Fig. 4. Point-to-point communication latency for small and medium messages. (a) Two-processes, (b) 34-processes, and (c) 68-processes.

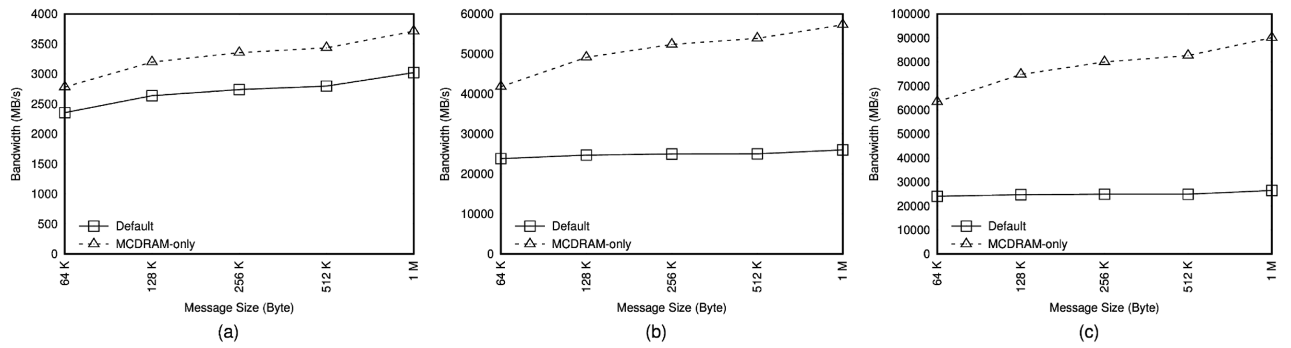

Fig. 5. Point-to-point communication bandwidth for large messages. (a) Two-processes, (b) 34-processes, and (c) 68-processes.

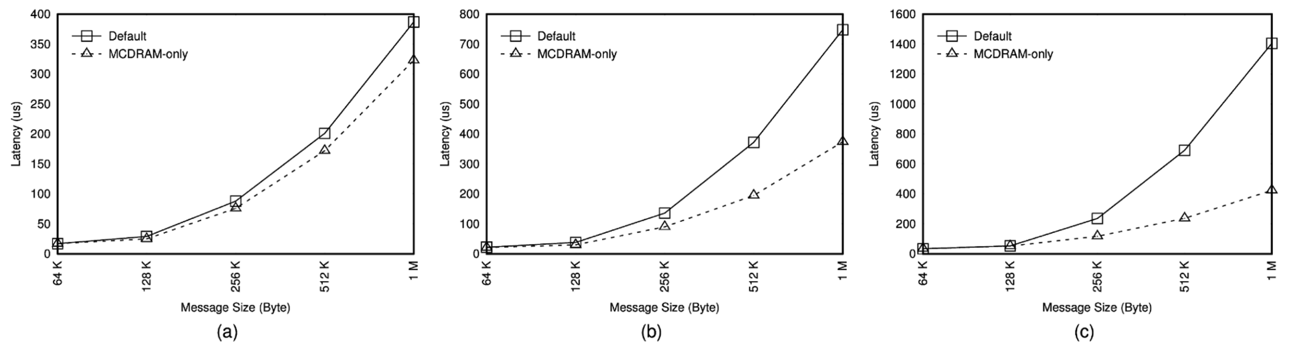

Fig. 6. Point-to-point communication latency for large messages. (a) Two-processes, (b) 34-processes, and (c) 68-processes.

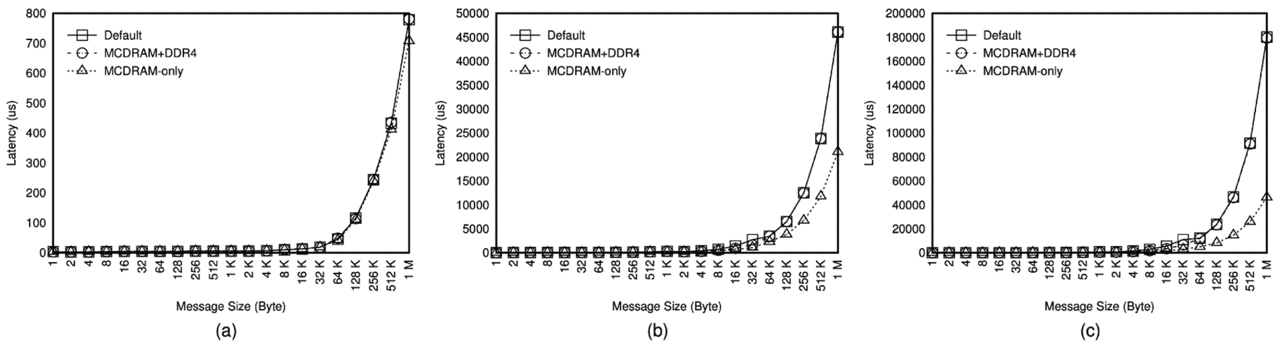

Fig. 7. Collective communication latency (all-to-all). (a) Two-processes, (b) 34-processes, and (c) 68-processes.

the number of processes. However, it is difficult to decide the optimal thresholds for the buffer size and the number of processes, because these can significantly vary according to the processor architectures and application workloads. Thus, it is desirable to have the application decide where to allocate buffers. The second argument of MPI_Alloc_mem() is provided to specify the requirements for the buffer to be allocated. In the latest specification, however, only the MPI_INFO_NULL value is valid for this argument. With the emergence of new memory hierarchy, standardization of the values is required to ensure that MPI applications specify the buffer requirements (e.g., performance or location) and benefit from the similar policies across different MPI implementations.

IV. IMPACT OF PROCESSOR INTERCONNECT

In this section, we analyze the impact of processor interconnect on MPI communication by varying the core affinity of MPI processes. We measured the performance of inter- and intra-node communication on KNL and Haswell processors.

A. Core Affinity

The core affinity defines mapping between a set of cores and a given task, which can be either a hardware event handler or a process. Since modern processor interconnects described in Section II-A exhibit asymmetric overheads to access memory and I/O devices, the core affinity affects MPI communication performance. In Linux, we can specify the cores that handle hardware events from a device by editing the smp_affinity file of the device in the /proc/irq directory. This file includes a bit vector, each bit of which is associated with a core and defines whether or not the core is allowed to handle the events from the device. In fact, this file defines the cores that receive interrupts from the device. In our experiments, we did not change the core affinity for the event handlers of 40-GigE and Omni-Path.

The core affinity of user-level processes can be changed by the sched_setaffinity() system call, which modifies the cpus_allowed field in the process control block. This field is also a bit vector that represents the cores on which the process can be scheduled. The /proc/<pid>/status file displays the value of this bit vector. We inserted a few lines of code into the OSU-microbenchmarks to set the core affinity of MPI processes.

We measured the performance of inter- and intra-node point-to-point communication on a pair of Intel Haswell 2687W machines connected through 40-GigE and a pair of Intel KNL 7250 machines connected through Omni-Path. The MPI implementation used was MVAPICH2 version 2.2 and the operating system installed was Linux (kernel version 3.10.0).

B. Inter-Node Communication

We measured the bandwidth of point-to-point communication of single connection with different core affinities of MPI processes. Fig. 8 shows the results of measurement using the Haswell machines. We compared four core affinity cases. The cases denoted as *Same NUMA Node* show the performance when the hardware event handler of network device (i.e., 40-GigE) and the communication process run on the same NUMA node. We reported the performance of the best and worst affinity combinations within the same NUMA node. The *Different NUMA Node* cases show the performance when the hardware event handler and the communication process run on different NUMA nodes. We find that *Same NUMA Node* outperforms *Different NUMA Node* as the message size increases. It is noteworthy that the event handler of 40-GigE ran on the NUMA node, to which the network device was attached (i.e., NUMA Node 1 in Fig. 2). Thus, the distance from the user buffer to the network device was closer in the *Same NUMA Node* case. Its performance was also enhanced by the better locality. The event handler of the network device not only performs TCP/IP processing but copies messages between user and kernel buffers as described in Section II-B. Thus, in the *Same NUMA Node*

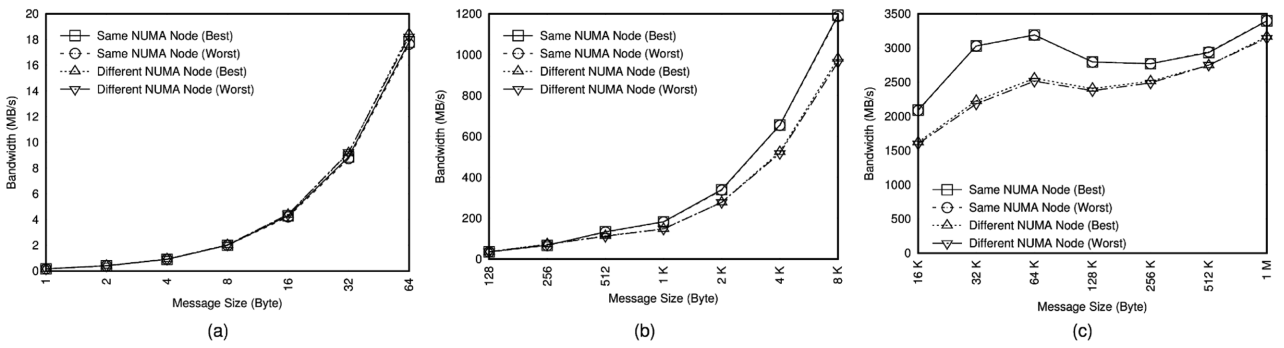

Fig. 8. Inter-node communication bandwidth on Haswell machines connected with 40-GigE. (a) Small, (b) medium, and (c) large messages.

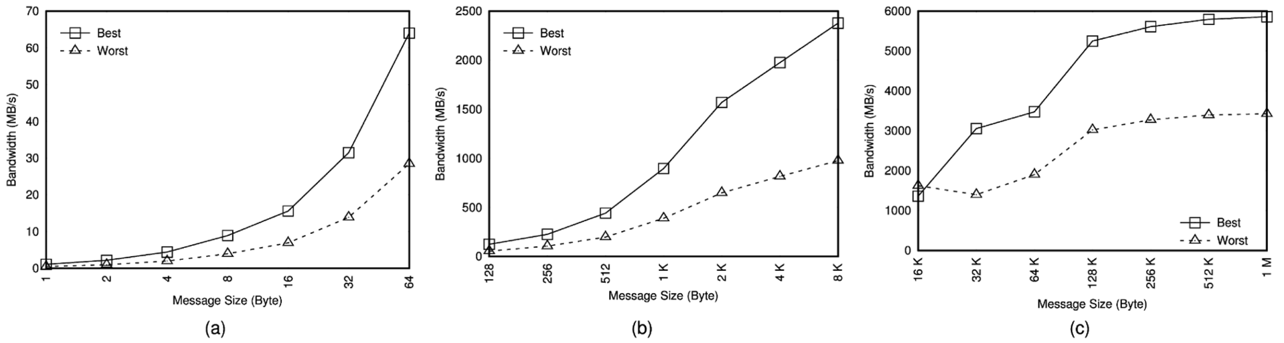

Fig. 9. Inter-node communication bandwidth on KNL machines connected with Omni-Path. (a) Small, (b) medium, and (c) large messages.

cases, the data accesses from event handler and communication process occur within the same NUMA node, which results in better cache efficiency. The best and worst cases in the same category do not show prominent differences, because the overhead caused by different core affinities within a NUMA node is not significant compared to the TCP/IP processing overheads.

Fig. 9 shows the measurement results on the KNL machines. The threshold to switch from *Eager* to *Rendezvous* was 16 kB and the network buffers were allocated in DDR4. Since the machines comprise a single processor package, we changed the core affinity of MPI processes only within the package. As shown in the figure, the best case reported 143% higher bandwidth than the worst case because the distance to the network device varied significantly on the 2D mesh interconnect.

C. Intra-Node Communication

We also measured the bandwidth of intra-node point-to-point communication for different core affinities. Fig. 10 shows the measurement results of single connection with the Haswell machine. The threshold to switch to *Rendezvous* was 32 kB. In these graphs, the *Same NUMA Node* cases show the bandwidth measured when the sending and receiving processes ran on the same NUMA node, while the communicating processes ran on different NUMA nodes in the other cases. Unlike the inter-node communication in Section IV-B, prominent differences in performance exist between the best and the worst cases in the same category, due to the low-overhead intra-node communication channels, which are not based on TCP/IP as described in Section II-B. Thus, the impact of processor interconnects on intra-node communication is increased compared with the inter-node communication cases. With respect to medium message sizes shown in Fig. 10(b), the *Same NUMA Node* cases yield better performance than the *Different NUMA Node* cases. However, for small and large message sizes in Fig. 10(a) and 10(c), providing the core affinity of communicating processes to the same NUMA node does not always guarantee better bandwidth. We suppose that running processes on the same NUMA node occasionally exhibits low performance on Haswell machine when small messages are involved in intra-node communication. In Fig. 10(a), user messages are small whereas in Fig. 10(c), the small control messages for *Rendezvous* protocol are exchanged.

Fig. 11 shows the intra-node communication bandwidth of different core affinities on the KNL machine. In these experiments, we ran 68 processes (i.e., 34 connections of

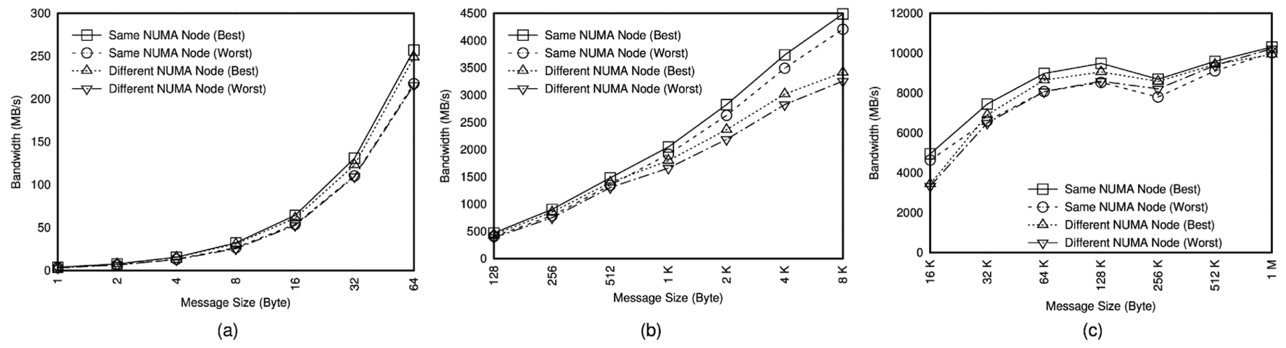

Fig. 10. Intra-node communication bandwidth on Haswell machine. (a) Small, (b) medium, and (c) large messages.

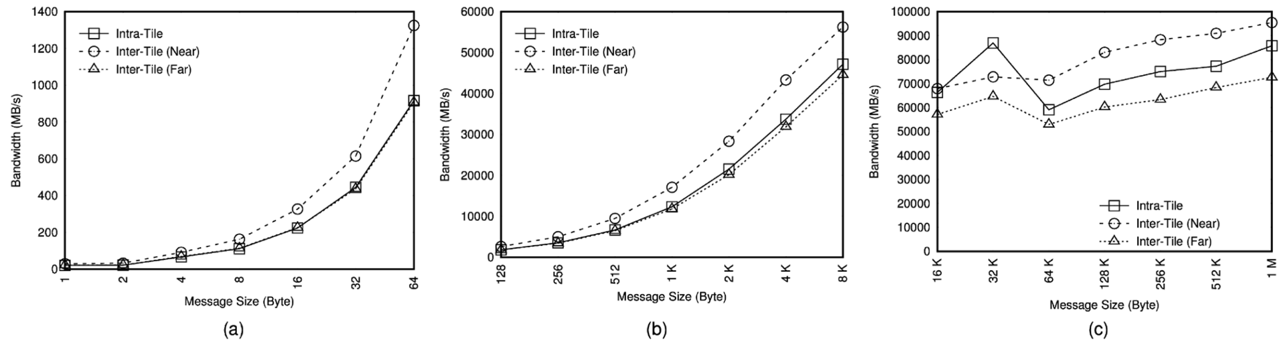

Fig. 11. Intra-node communication bandwidth on KNL machine. (a) Small, (b) medium, and (c) large messages.

point-to-point communication) and allocated communication buffers to MCDRAM. We reported the bandwidth of three different affinity policies. The *Intra-Tile* case ran the processes paired on the same tile. The *Inter-Tile* cases facilitated the processes in the same pair on two different tiles, which are either close to each other or far apart. As shown in Fig. 11, *Inter-Tile (Near)* shows better performance than the others for most of the message sizes. It suggests that running the sender and the corresponding receiver on the cores that share the LLC may not promise the highest communication bandwidth.

D. Implications

As the number of cores increases, the scalability of processor interconnects becomes a very important issue. In this section, we showed that the contemporary interconnects affect the communication performance depending on the core affinity. Although many studies focused on inter-NUMA-node process scheduling, our experimental results showed that the communication performance could be changed significantly depending on the location of the communicating process within the same NUMA node. The results emphasize the importance of intra-NUMA-node process scheduling with respect to communication performance. For example, SyMMer [16] determines the core affinity of MPI processes dynamically at the run time. Such tools need to be extended to allow for adaptive decision of the core affinity of MPI processes even within a NUMA node considering the characteristics of the processor interconnect.

V. RELATED WORK

Performance analyses of high-bandwidth memory, such as MCDRAM, have been studied comprehensively [2-4]. Li et al. [2] and Peng et al. [3] analyzed the impact of high-bandwidth memory on HPC applications. Xing et al. [4] also optimized the performance of a graphic analysis application over the many-core system.

The impacts of core affinity on MPI applications were analyzed in various studies [16-19]. SyMMer [16] proposed a library to determine the core affinity of MPI processes. Ganapathi et al. [17] analyzed the impact of the distance between the NUMA node and the network interface card. LAMA [18] provided the environmental variables that allowed users to set core affinity for MPI applications. TreeMatch [19] was used to propose an algorithm that abstracted the hardware architecture into a tree format and mapped the MPI processes.

However, the current studies have analyzed the impact of high-bandwidth memory and core affinity at the application level. In this paper, we measured and analyzed the impact of emerging many-core architectures in terms of MPI programming model and run-time supports. Furthermore, we suggested several ways to optimize the MPI implementations on many-core systems.

VI. CONCLUSIONS

In this paper, we analyzed the impact of high-bandwidth memory and processor interconnects on the performance of MPI communication. We showed the potential of the high-bandwidth memory for improved performance of MPI intra-node communication and suggested more specific usages of the second argument in MPI_Alloc_mem(). In addition, we presented the impact of processor interconnects by changing the core affinity of MPI processes. The experimental results suggested that the core affinity within a processor package was also critical in many-core processors. We believe that our analyses provide directions for future optimization of MPI implementations in emerging many-core architectures. In the future, we intend to investigate the scalability of many-core systems with respect to MPI communication.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2017R1A2B4012759).

REFERENCES

1. Intel, "Intel Xeon Phi processor product brief," http://www.intel.com/xeonphi/.
2. A. Li, W. Liu, M. R. B. Kristensen, B. Vinter, H. Wang, K. Hou, A. Marquez, and S. L. Song, "Exploring and analyzing the real impact of modern on-package memory on HPC scientific kernels," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, 2017.
3. I. B. Peng, R. Gioiosa, G. Kestor, J. S. Vetter, P. Cicotti, E. Laure, and S. Markidis, "Characterizing the performance benefit of hybrid memory system for HPC applications," *Parallel Computing*, vol. 76, pp. 57-69, 2018.
4. Y. Xing, Z. Chen, N. Xiao, F. Liu, and Y. Lu, "Graph analytics on manycore memory systems," *IEEE Access*, vol. 6, pp. 51429-51439, 2018.
5. MPI Forum, "Message Passing Interface," https://www.mpi-forum.org/.
6. Network-Based Computing Laboratory, "MVAPICH2," http://mvapich.cse.ohio-state.edu/.
7. L. Chai, A. Hartono, and D. K. Panda, "Designing high performance and scalable MPI intra-node communication support for clusters," in *Proceedings of 2006 IEEE International Conference on Cluster Computing*, Barcelona, Spain, 2006, pp. 1-10.
8. L. Chai, P. Lai, H. W. Jin, and D. K. Panda, "Designing an efficient kernel-level and user-level hybrid approach for MPI intra-node communication on multi-core systems," in *Proceedings of the 37th International Conference on Parallel Processing*, Portland, OR, 2008, pp. 222-229.
9. D. Buntinas, B. Goglin, D. Goodell, G. Mercier, and S. Moreaud, "Cache-efficient, intranode, large-message MPI communication with MPICH2-nemesis," in *Proceedings of the International Conference on Parallel Processing*, Vienna, Austria, 2009, pp. 462-469.
10. H. W. Jin, S. Sur, L. Chai, and D. K. Panda, "Lightweight kernel-level primitives for high-performance MPI intra-node communication over multi-core systems," in *Proceedings of 2007 IEEE International Conference on Cluster Computing*, Austin, TX, 2007, pp. 446-451.
11. B. Goglin, and M. Stephanie, "KNEM: a generic and scalable kernel-assisted intra-node MPI communication framework," *Journal of Parallel and Distributed Computing*, vol. 73, no. 2, pp. 176-188, 2013.
12. J. Vienne, "Benefits of cross memory attach for MPI libraries on HPC clusters," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, Atlanta, GA, 2014.
13. Intel, "Memkind library," http://memkind.github.io/.
14. A. Kleen, "A NUMA API for Linux," Novell Inc., Technical Whitepaper, 2005, http://developer.amd.com/wordpress/media/2012/10/LibNUMA-WP-fv1.pdf.
15. J. Y. Cho, H. W. Jin, and D. Nam, "Enhanced memory management for scalable MPI intra-node communication on many-core processor," in *Proceedings of the 24th European MPI Users' Group Meeting*, Chicago, IL, 2017.
16. T. Scogland, P. Balaji, W. Feng, and G. Narayanaswamy, "Asymmetric interactions in symmetric multi-core systems: analysis, enhancements and evaluation," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Austin, TX, 2008, pp. 1-12.
17. R. B. Ganapathi, A. Gopalakrishnan, and R. W. McGuire, "MPI process and network device affinitization for optimal HPC application performance," in *Proceedings of 2017 IEEE 25th Annual Symposium on High-Performance Interconnects*, Santa Clara, CA, 2017, pp. 80-86.
18. J. Hursey, and J. M. Squyres, "Advancing application process affinity experimentation: open MPI's LAMA-based affinity interface," in *Proceedings of the 20th European MPI Users' Group Meeting*, Madrid, Spain, 2013, pp. 163-168.
19. E. Jeannot and G. Mercier, "Near-optimal placement of MPI processes on hierarchical NUMA architectures," in *Euro-Par 2010 Parallel Processing*. Heidelberg: Springer, 2010, pp. 199-210.

Joong-Yeon Cho

Joong-Yeon Cho received the B.S. degree in computer science and engineering from Konkuk University, Seoul, Korea, in 2012 and the M.S. degree in computer, information & communications engineering from Konkuk University, Seoul, Korea, in 2014. He is currently pursuing the Ph.D. degree in computer, information & communications engineering at Konkuk University, Seoul, Korea (e-mail: jycho@konkuk.ac.kr). His research interests include operating systems, cloud computing and high-performance computing.

Hyun-Wook Jin

Hyun-Wook Jin received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Korea University, Korea, in 1997, 1999, and 2003, respectively. From 2003 to 2006, he was a Research Associate with the Department of Computer Science and Engineering, The Ohio State University, USA. Since 2006, he has been with the Department of Computer Science and Engineering, Konkuk University, Seoul, Korea, where he is currently a Full Professor (e-mail: jinh@konkuk.ac.kr). His research interests include operating systems, parallel programming models, cloud computing, and real-time systems.

Dukyun Nam

Dukyun Nam is a senior researcher in the National Institute of Supercomputing and Networking at Korea Institute of Science and Technology Information (KISTI), Daejeon, Republic of Korea (e-mail:dynam@kisti.re.kr). He received the B.S. degree in computer science and engineering from Pohang University of Science and Technology (POSTECH), Korea, in 1999, M.S. and Ph.D. degrees in information and communication engineering from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 2001 and 2006, respectively. His research interests are in high performance and distributed computing, low power computing, system software in HPC, etc.