An In-Depth Performance Analysis of Many-Integrated Core for Communication Efficient Heterogeneous Computing

Jie Zhang, Myoungsoo Jung

An In-depth Performance Analysis of Many-Integrated Core for Communication Efficient Heterogeneous Computing

Jie Zhang and Myoungsoo Jung

Computer Architecture and Memory Systems Laboratory,
School of Integrated Technology,
Yonsei University
jie@camelab.org and mj@camelab.org

Abstract. Many-integrated core (MIC) architecture combines dozens of reduced x86 cores onto a single chip to offer high degrees of parallelism. The parallel user applications executed across many cores that exist in one or more MICs require a series of work related to data sharing and synchronization with the host. In this work, we build a real CPU+MIC heterogeneous cluster and analyze its performance behaviors by examining different communication methods such as message passing method and remote direct memory accesses. Our evaluation results and in-depth studies reveal that i) aggregating small messages can improve network bandwidth without violating latency restrictions, ii) while MICs can execute hundreds of hardware cores, the highest network throughput is achieved when only $4 \sim 6$ point-to-point connections are established for data communication, iii) data communication over multiple point-to-point connections between host and MICs introduce severe load unbalancing, which require to be optimized for future heterogeneous computing.

Keywords: Manycore, Communication, Accelerator, Parallel Programming, High Performance Computing, Heterogeneous Computing

1 Introduction

In the past few years, many-core integrated (MIC) architecture has be employed for accelerating the heterogeneous computing [5]. Typically, a MIC architecture chip has dozens of reduced x86 cores, which can provide massive parallelism. Parallel user applications, which are executed across different cores of MIC(s), can share data or collaborate to compute with each other by using user-level communication systems, such as massage passing interface (MPI) [2] or open computing language (OpenCL) [6]. For the modern MIC designs, these user-level communication systems in practice are built upon a symmetric communication interface, referred to as *SCIF*, which is a kernel component of manycore platform software stack. Even though there are many prior studies that perform the user-level optimizations for efficient manycore communication [4], little attention has been paid to the low-level communication methods for MICs.

In this work, we build a real MIC-accelerated heterogeneous cluster that has eight main processor cores and 244 physical MIC cores (61 cores per MIC device) and characterize the performance behaviors on the heterogeneous cluster, which are observed by the low-level communication methods. Specifically, we evaluate the latency and bandwidth of the cluster using two different strategies: i) *massing passing* and ii) *remote memory access* (RMA). While the massage passing establish a pair of message queues for exchanging short, latency-sensitive messages, RMA enables one process to remotely access the memory of target process it connected to. In this paper, we explore a full design space of MICs with those two the message passing and RMA communication methods by taking into account a wide spectrum of system parameters, including different datapath configurations, various data page and message sizes, as well as different number of threads, ports and connection channels.

2 Background

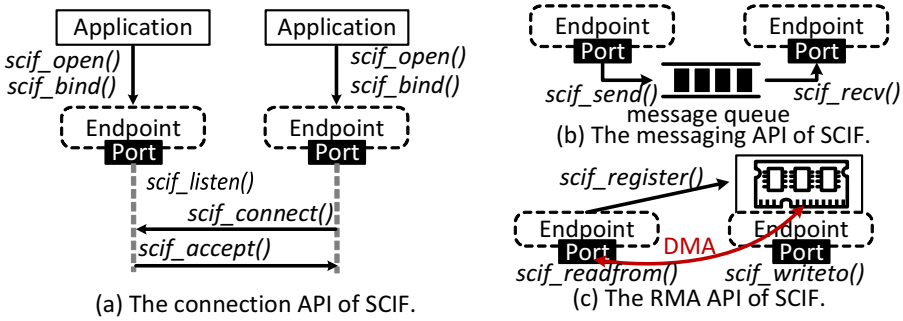
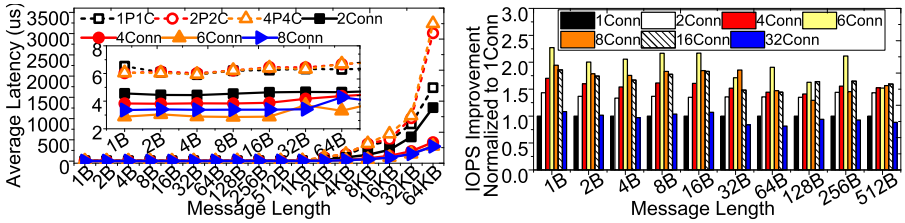


Fig. 1: Communication APIs of SCIF interface.

In this section, we briefly explain how MIC devices communicate with the host through the user-level APIs supported by the software stack. In heterogeneous cluster, each separate multiprocessor (i.e., host and MICs) is regarded as a node. The software stack of MIC architectures provides a common transport interface, which is referred to as *symmetric communication interface* (SCIF) [1], to establish a point-to-point communication link to connect a pair of processes on either different nodes or in the same node. The APIs supported by the SCIF user mode library can be categorized as a set of connection APIs, messaging APIs, and remote memory access (RMA) APIs. Figure 1 demonstrates how two processes can connect with each other over SCIF APIs. As shown in Figure 1a, the connection APIs provide a socket-like hand-shaking procedure (e.g., `scif_open()`, `scif_bind()`, `scif_listen()`, `scif_connect()`, and `scif_accept()`) to set up connections [7]. The messaging APIs support a two-sided communication between connected processes by implementing message queues (c.f. Figure 1b). Messages can be sent and received via the commands `scif_send()` and `scif_recv()`. On the other hand, the RMA APIs are responsible for transferring a large bulk of data. As shown in Figure 1c, it first maps a specific memory region of a local process to the address space of target process through a memory registration

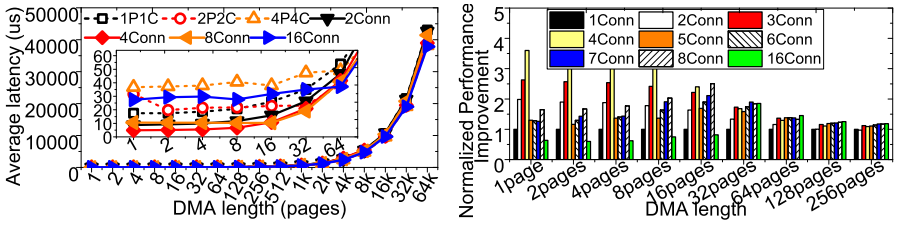
API, `scif_register()`. It then leverages read/write APIs, `scif_readfrom()` and `scif_writeto()`, to access remote data.

3 Empirical Evaluation and Analysis



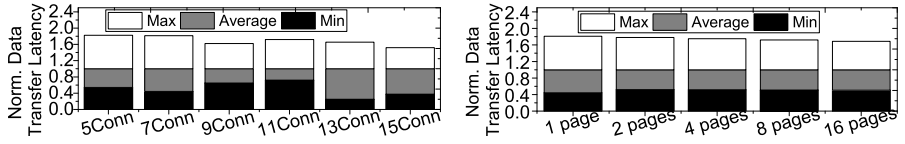
(a) Average latency. (b) IOPS improvement.

Fig. 2: Performance improvement of message passing with multiple connection channels.



(a) Average latency. (b) Performance normalized to 1Conn.

Fig. 3: Performance improvements of RMA with connection channels.



(a) Latency of various connection channels. (b) Latency of various DMA lengths.

Fig. 4: Max, min, and average communication latencies of each thread under various connection channel and DMA length configurations.

Message-based Communication Method. Figure 2 illustrates the performance of MIC message passing mechanism by employing different number of producer/consumer threads within single connection channel or establishing multiple connection channels. Specifically, "XPYC" indicates X producer threads and Y consumer threads are employed for a single connection channel, while "XConn" means X connection channels are established with a single producer thread and a single consumer thread per connection channel. From the results, we can conclude that, it improves communication throughput without introducing extra latency penalty, if one can aggregate multiple small-size messages to 512B (***Finding 1***). Multiple producer and consumer threads are unable to improve the communication throughput of single point-to-point connection channel (***Finding 2***). One can also observe that properly establishing multiple point-to-point connection channels can significantly improve the performance (***Finding 3***).

RMA-based Communication Method. Figure 3 shows the performance of RMA by employing multiple producers-consumers or setting up multiple channel

connections. From this figure, we conclude that it is better to leverage message-based approach to transfer data whose size is larger than 512B, compared to messaging based approach, even though the minimum data access granularity of the RMA-based approach is 4KB (***Finding 4***). As shown in Figure 3, more producer/consumer threads unfortunately cannot help improve the performance (***Finding 5***). In addition, more connection channels can improve the performance, but four connection channels are sufficient to achieve the best performance (***Finding 6***). To figure out the reason behind the poor performance imposed by establishing many connection channels, we analyze the busy time of each thread which performs data transfer over an individual connection channel. Figure 4 shows the maximum, minimum, and average communication latency of each thread with different number of connection channels and different transfer data size. Based on the results, we can conclude that the communication over SCIF can introduce long tail latency, which degrades the performance (***Finding 7***).

4 Conclusion

In this work, we evaluated and analyzed the performance of inter-node communications across CPU cores and multiple MICs. Our evaluation results reveal that the performance of current inter-node communication methods is sub-optimized owing to the low throughput of small requests and the long tail latency. We then provide seven system-level findings with an in-depth performance analysis.

5 Acknowledgement

This research is mainly supported by NRF 2016R1C1B2015312. This work is also supported in part by IITP-2017-2017-0-01015, NRF-2015M3C4A7065645, DOE DE-AC02-05CH 11231 and MemRay grant (2015-11-1731). The corresponding author is M. Jung.

References

1. I. M. I. Core, "Symmetric communications interface (scif) user guide," *Intel Std., Rev. 0.8*, 2012.
2. W. Gropp *et al.*, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.
3. Intel, "Intel Xeon Phi 7120A https://ark.intel.com/products/80555," 2014.
4. S. Potluri *et al.*, "Efficient inter-node mpi communication using gpudirect rdma for infiniband clusters with nvidia gpus," in *ICPP*. IEEE, 2013, pp. 80–89.
5. E. Saule *et al.*, "Performance evaluation of sparse matrix multiplication kernels on intel xeon phi," in *PPAM*. Springer, 2013.
6. J. E. Stone *et al.*, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, 2010.
7. M. Xue and C. Zhu, "The socket programming and software design for communication based on client/server," in *PACCS*. IEEE, 2009, pp. 775–777.