Understanding the IO Performance Gap Between Cori KNL and Haswell

Jialin Liu[1], Quincey Koziol[1], Houjun Tang[2], François Tessier[3], Wahid Bhimji[1], Brandon Cook[1], Brian Austin[1], Suren Byna[2], Bhupender Thakur[1], Glenn Lockwood[1], Jack Deslippe[1], Prabhat[1].

***Abstract*— The Cori system at NERSC has two compute partitions with different CPU architectures: a 2,004 node Haswell partition and a 9,688 node KNL partition, which ranked as the 5th most powerful and fastest supercomputer on the November 2016 Top 500 list. The compute partitions share a common storage configuration, and understanding the IO performance gap between them is important, impacting not only to NERSC/LBNL users and other national labs, but also to the relevant hardware vendors and software developers. In this paper, we have analyzed performance of single core and single node IO comprehensively on the Haswell and KNL partitions, and have discovered the major bottlenecks, which include CPU frequencies and memory copy performance. We have also extended our performance tests to multi-node IO and revealed the IO cost difference caused by network latency, buffer size, and communication cost. Overall, we have developed a strong understanding of the IO gap between Haswell and KNL nodes and the lessons learned from this exploration will guide us in designing optimal IO solutions in many-core era.**

I. Introduction

Cori [1], a Cray XC40 system, is an integrated system with Intel Haswell and Xeon Phi Knights Landing (KNL) processors that provides a flexible and heterogeneous computing environment to users and enables efficient hybrid programming for faster scientific discovery. Each KNL node has 68 physical cores, each supporting 4 hyperthreads, for a total of 272 threads per node and each Haswell node has 32 physical cores, each supporting 2 hyperthreads, for a total 64 of threads per node.

Optimizing applications for many-core architectures such as KNL often involves significant effort to take full advantage of the larger number of weaker cores, high-bandwidth on-package memory, and wider vector units. NERSC's NESAP program [6] selected 20 scientific applications and focused on optimizing the computation and communication aspects of each for the Cori system, but we focus on I/O performance in this work.

Figure 1 shows the configuration of the Lustre file system shared by the KNL and Haswell partitions on Cori [2]. There are 130 LNET routers that connect the compute nodes with storage nodes, with all users randomly distributed among the five metadata servers: 1 major server and 4 additional servers. Each OSS controls one OST, and the OSTs are configured with GridRAID, similar to RAID6 (8+2), but can restore failure 3.5 times faster than traditional RAID6. Each OST consists of 41 disks, and can deliver 240TB capacity. The

[1]National Energy Research Scientific Computing Center, jalnliu@lbl.gov
[2]Lawrence Berkeley National Laboratory
[3]Argonne National Laboratory

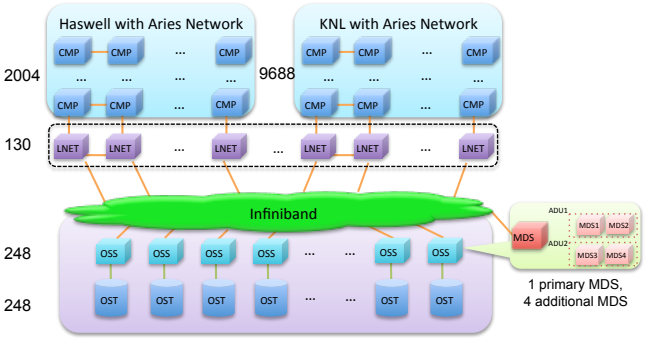

Fig. 1. Cori IO System

Infiniband network connects the MDS, ADUs and OSSs to the LNET routers on the system.

On the compute side, 2004 Haswell nodes share the 130 LNET servers with the 9688 KNL nodes. Thus, from the storage point of view, there is no difference between the two partitions and implies that any potential IO difference mainly comes from the compute node architecture.

During our initial performance comparisons, we noted various I/O performance discrepancies between the two partitions. For example, in our 32 node HDF5 parallel IO test, Haswell cores were 1.68X faster than KNL. In order to understand this gap, we have started from the single core case, and conducted comprehensive tests in different scenarios, including multi-core/node cases:

- Single core test with buffer/sync/direct IO
- Scaling test with varying CPU frequencies
- Single core test with different MCDRAM modes
- Scaling test with multiple cores on a single node
- Inter/Intra node point to point tests
- Inter-nodes MPI collective and independent IO test

Through these tests and related data analysis, we are able to understand the IO gap by isolating the different software/hardware components. Section II focuses on IO performance from a single core. Section III extends these experiments to multi-core/node, while Section IV concludes this paper and provide guidelines to efficiently perform IO on the targeted architectures.

II. Single Core IO Performance

A. Benchmark and Experimental Setup

For single core IO performance tests, we used the Unix command *dd*. This command creates and fills a file according to parameters, such as the block size per operation. For

example, $dd\ if = input\ of = output$, in which a POSIX read is conducted first, then followed by a POSIX write. This read/write is repeated a number of $count$ times. Each read/write operation is of size bs bytes. We used $count = 1000$ and $bs = 1MB$, which generates a 1GB file.

We also used a simple HDF5 benchmark written by ourselves [3]. This HDF5 code performs a simple HDF5 write function call using MPI independent IO to write a 2D array. The file produced as output is also 1 GB in size.

The first POSIX benchmark based on *dd* was chosen to avoid as much as possible the impact of the HPC IO software stack (MPI, HDF5, etc) and to highlight the hardware difference between Haswell and KNL. With *dd*'s benchmark as a baseline, the second HDF5 experiment was chosen as comparison, and a practical indicator for typical application performance.

These experiments were run on Cori and also confirmed on the KNL partition on Theta, the new Cray XC40 supercomputer at Argonne National Laboratory [5].

B. IO Path on Cori

In our first test we vary the CPU frequency and measure IO performance on both Haswell and KNL cores. As shown in Figure 3, we requested different CPU scaling options on both partitions with SLURM, e.g., srun –cpu-freq=1300000, and performed a simple Unix *dd* test with 1 MB block size and 10000 count. This *dd* test first generates a 1 MB buffer full of zeros in memory then writes onto disk repeatedly, producing a 10GB file.

IO performance decreases as the CPU frequencies decrease, but KNL IO performance is much slower than Haswell, even when their CPU frequencies are set to the same value (e.g. 1.4GHz, 1.3GHz, and 1.2GHz). We continued to profile the CPU statistics with perf, and the results are explored further in section II-C. We also noted that the workload in this test is relatively small, and can fit in the page cache very well, which means minimal disk IO is involved in this test and indicates memory and page cache could be factors that cause the IO difference between Haswell and KNL. In order to understand the page cache impact, we conducted comprehensive IO benchmarking based on different IO modes and different memory modes.

Figure 2 shows the different IO paths on KNL cores for different MCDRAM modes and different IO modes (e.g., buffered IO or synchronous IO). As mentioned before, the MCDRAM on KNL can be configured in multiple modes. In cache mode, the MCDRAM will be used as a direct mapped last level cache for DRAM; in flat mode, the MCDRAM will be configured as additional addressable main memory exposed as separate NUMA domain. As shown in Figure 2, the location of the user space memory on KNL node depends on the configuration of MCDRAM:

- **Cache** Mode: user space is in dram
- **Flat** Mode with '$numactl$ **-m** 1': user space is in MCDRAM
- **Flat** Mode with '$numactl$ **-p** 1': user space is allocated in MCDRAM preferably

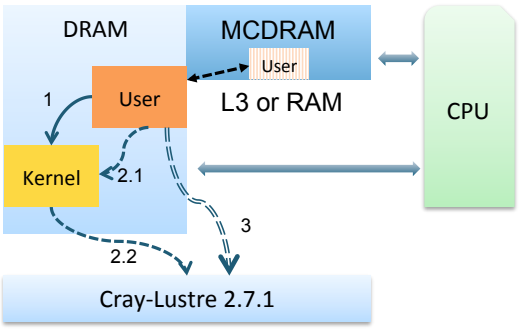

Fig. 2. IO Path on KNL, 1/2/3 is buffered/synchronous/direct IO; User data is located in either MCDRAM or DRAM and depends on MCDRAM mode and NUMA configuration

The IO path is different under different IO modes such as buffered, synchronous, and direct IO. In this paper, when referring *IO path*, we refer to the dominant costly IO operation in the workflow. For example, the IO path of buffered IO (e.g., write) with MCDRAM in cache mode includes a costly synchronized $memcpy$ from user space to kernel space, and less costly asynchronized $flush$ from kernel space to file system (assuming the buffer is not full). Therefore, for the three IO modes, we have their dominant IO cost path as:

- **Buffered** IO: $memcpy$ from user space to kernel space
- **Synchronous** IO: $memcpy$ from user space to kernel space, and $flush$ from kernel space to disk
- **Direct** IO: $flush$ from user space to disk

C. Buffered IO

Generally, applications benefit from page buffering during IO. For example, during writes data is written to page buffers, which resides in kernel space in memory, and then the page buffers are periodically flushed to disk asynchronously. The write function call immediately returns once the data is in page buffer, without waiting for the longer disk IO to complete.

As explained in the previous section, we varied the CPU frequencies on the Haswell core from 2.301GHz (default, turbo mode) to 2.3, 2.2, 2.1, 2.0, 1.9, 1.8, 1.7, 1.6, 1.5, 1.4 and 1.3GHz. And for the KNL core, we varied from 1.401 (default, turbo mode) to 1.4, 1.3, 1.2, 1.1, 1.0GHz. These frequencies are determined by the core's available scaling options, i.e., $scaling\_available\_frequencies$ in $/sys/devices/system/cpu/cpu0/cpufreq/$. We should also note that there is no guarantee for having a constant cpu frequency. Setting the $--cpu-freq$ in SLURM is only putting a limit on the maximum cpu frequencies. The actual cpu frequency in each test varied between a range, and only an averaged frequency number over this period is reported in the $perf$ profiling result. As shown in Figure 3, we found that the IO bandwidth ratio of KNL to Haswell is 29%(turbo mode) and 43% (Non-turbo mode). Similar testing was also conducted using the HDF5 benchmark, Figure 4.

On both Haswell and KNL cores the IO bandwidth scaled proportionally with CPU frequencies. We run linear regres-

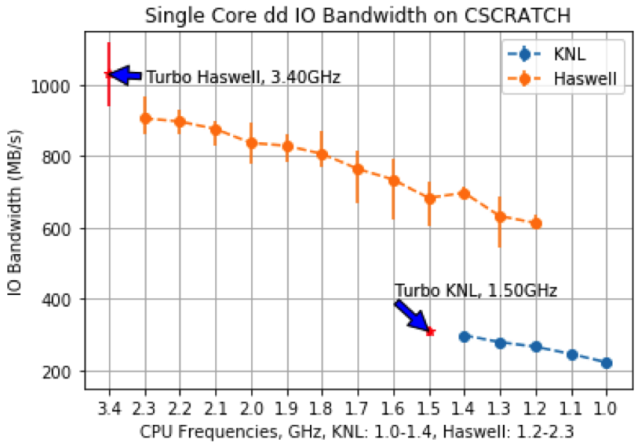

Fig. 3. Buffered IO Performance with Varying CPU Frequencies, dd

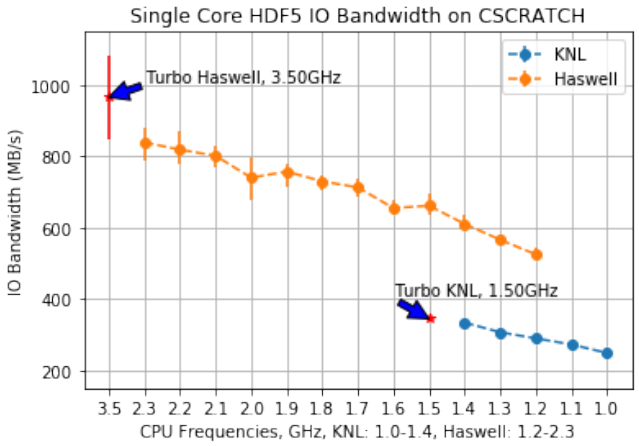

Fig. 4. Buffered IO Performance with Varying CPU Frequencies, HDF5

sion analysis and plot the relationship between IO and CPU frequencies in Figures 5, 6, 7, 8.

Turbo mode typically involves larger variance of CPU frequencies, thus, we see a worse fitting in those plots. The regression results for non-turbo mode are summarized in Table I. The r^2 value of 0.95 indicates a strong correlation between KNL core CPU frequencies and its IO bandwidth, while the relationship is less significant on Haswell cores. The non-zero intercepts, 286MB/s on Haswell and 41MB/s on KNL, indicate that when the CPU is idle, the IO bandwidth is not zero, which probably is caused by page buffer IO or DMA. Overall, when all the data fits into page buffers, we have observed a strong correlation between CPU frequencies and IO bandwidth. When page buffering is disabled and real storage IO is involved, we expect that the impact from CPU

TABLE I
REGRESSION RESULTS (NON TURBO MODE)

	Haswell	KNL
r^2	0.79	0.95
intercept	286.11	41.28

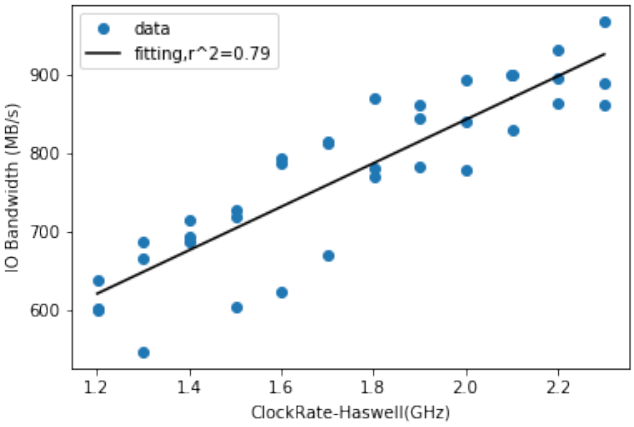

Fig. 5. Haswell IO Bandwidth vs. CPU Frequencies, Page Cache On

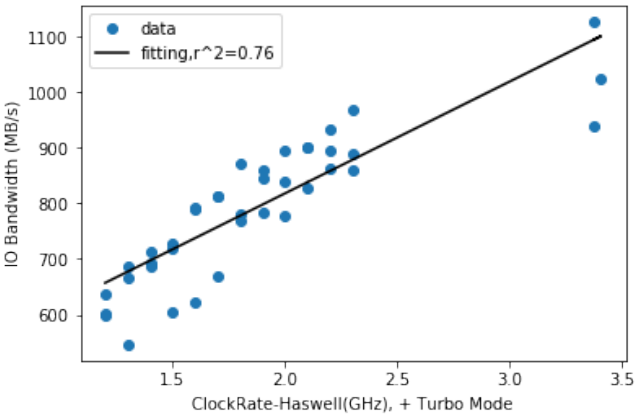

Fig. 6. Haswell IO Bandwidth vs. CPU Frequencies (with Turbo Mode), Page Cache On

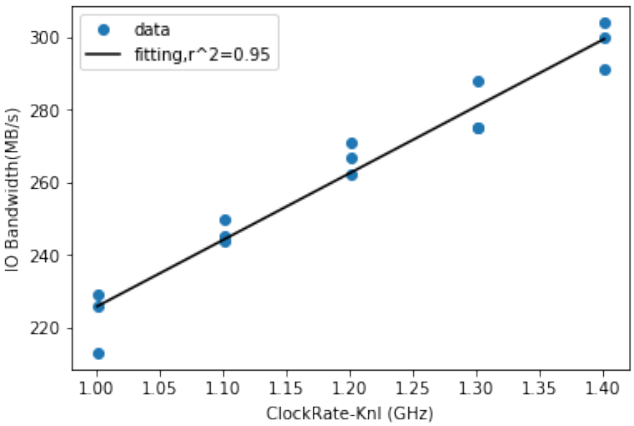

Fig. 7. KNL IO Bandwidth vs. CPU Frequencies, Page Cache On

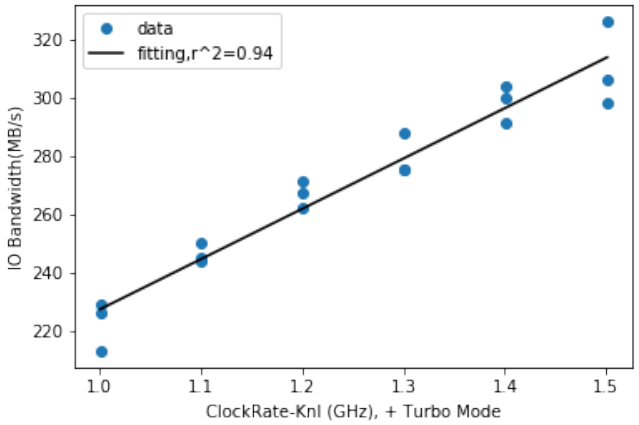

Fig. 8. KNL IO Bandwidth vs. CPU Frequencies (with Turbo Mode), Page Cache On

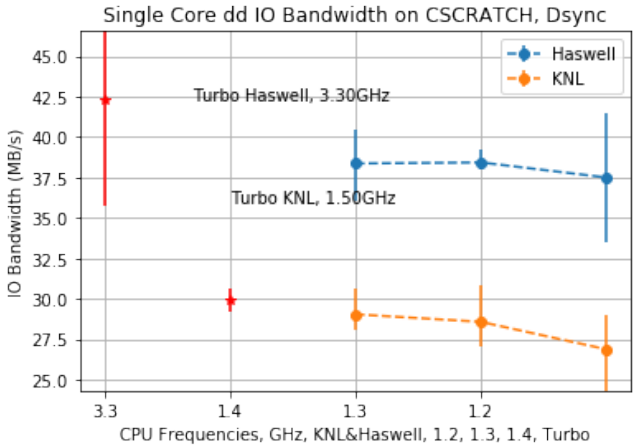

Fig. 9. IO Bandwidth vs. CPU Frequencies (with Turbo Mode), Synchronous IO

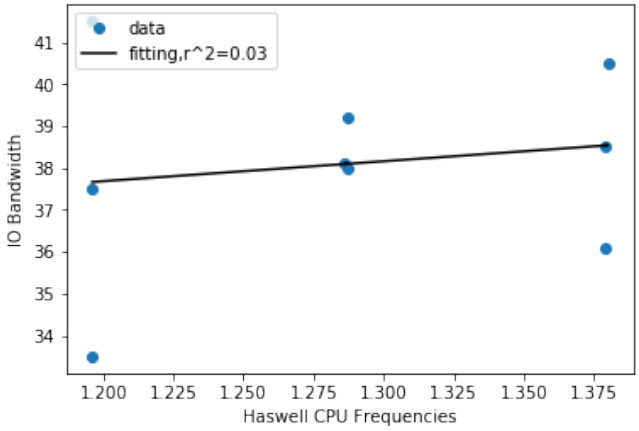

Fig. 10. Haswell IO Bandwidth vs. CPU Frequencies, Synchronous IO

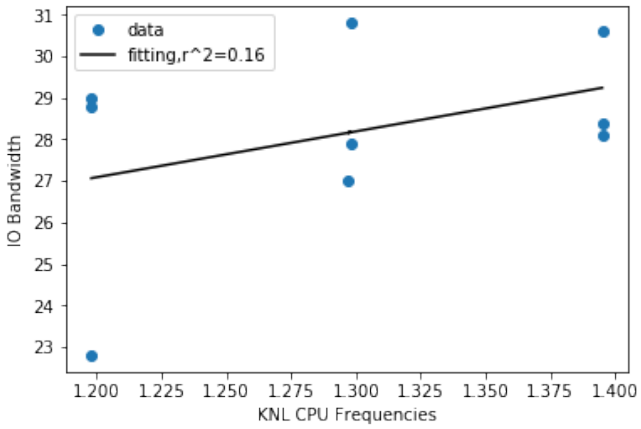

Fig. 11. KNL IO Bandwidth vs. CPU Frequencies, Synchronous IO

architecture will diminish, which we explore in the following section.

D. Synchronous IO

We then turned on the 'dsync' option for *dd* to force data to be flushed to disk during each write. By forcing the data to be flushed to disk, we observed a dramatic bandwidth drop, with a larger variance (see the error bar), in Figure 9. The IO bandwidth gap between KNL and Haswell, however, is reduced, i.e., $\frac{KNL\ Bandwidth}{Haswell\ Bandwidth} = 67\%$ compared to 43% for buffered IO, as shown earlier in Section II-C.

This means the CPU impact diminishes when disk IO is actually involved, and the major IO path is now dominated by storage performance (i.e. with Lustre). We confirmed this through the regression analysis: in Figure 10 and 11, we can see that r^2 is only 0.03 on Haswell cores and 0.16 on KNL cores. This lower r^2 means there is merely a correlation between CPU frequencies and IO bandwidth at this point.

On KNL, we also set the MCDRAM to different modes, i.e., L3 cache and addressable memory space, and expected better performance in latter case, as illustrated in the IO path in Figure 2. Using MCDRAM as the only (m=1) or the first priority (p=1) addressable memory space should offer faster memory allocation and memory copy. However, we observed slight performance degradation, as shown in Table II, which is probably caused cross-memory (DRAM and MCDRAM) copy and cross-memory buffer management. To bypass this memory overhead, we continued with direct IO, where page buffer copies are skipped, and a write function call will directly dump data from user space to disk.

E. Direct IO

The IO bandwidth ratio of KNL to haswell at the same CPU frequencies, is now increased to 96%. This shows a very close IO performance between the two type of nodes.

TABLE II
BANDWIDTH OF SYNC IO

	Haswell	Knl-dram	Knl-mcdram
stdev	0.55	0.46	0.22
average	45.21	31.23	30.20
median	45.35	31.47	31.13
$\frac{knl\ bw}{haswell\ bw}$		69%	67%

The comparison isolated the CPU frequencies by setting the same frequencies, and avoided the memory copy cost by bypassing the kernel space page buffer. This test demonstrated a fair performance comparison, which minimized IO gap and revealed the previous unknown issue.

TABLE III
BANDWIDTH OF DIRECT IO

	Haswell	KNL-DRAM	KNL-MCDRAM
stdev	1.21	1.02	1.15
average	48.89	43.83	46.78
median	49.60	43.93	46.40
$\frac{KNL\ bw}{Haswell\ bw}$		90%	**96%**

F. Summary of Single Core Tests

In this single core test section, we have walked through different IO modes, e.g., buffered/synchronous/direct, and revealed the IO difference between Haswell and KNL cores. In summary, on single core performance, page buffer helps a lot in many cases, e.g., write, and multiple read. But we also found that **KNL IO is closer to Haswell when less software layers are involved.** With the aid of page buffering, both chips' IO performance are scaling proportionally with their CPU frequencies separately when IO can fit into the buffer. In summary, we plot the the performance ratio under different IO modes in Figure 12.

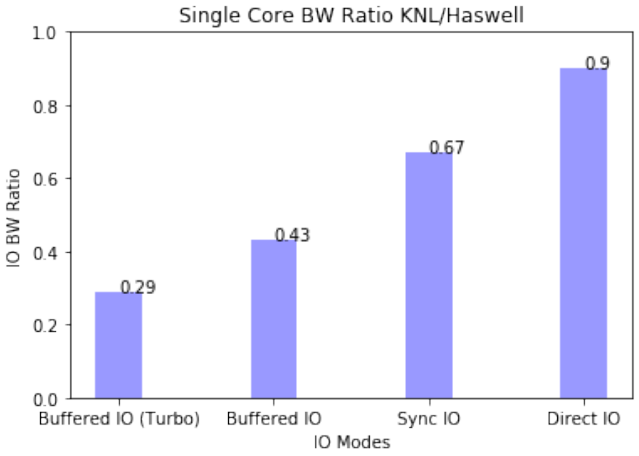

Fig. 12. Performance Ratio between KNL and Haswell with different IO mode

We could see clearly that the KNL is closer to Haswell on the right side of the plot. The KNL performs badly on the left side. (Except the left most 'Buffered IO(Turbo)', which is measured at turbo cpu frequencies, other test are compared at same cpu frequencies. For example, the third bar is the synchronous IO performance ratio between KNL and Haswell at same cpu frequencies, i.e., 1.2, 1.3, 1.4 GHz). **Note that the absolute bandwidth dropped from 1000 MB/s to 40 MB/s from left to right.** A different view is shown in Figure 13 what we learned from this probably indicates that **buffer management is slower on KNL**, which will be our future work to investigate. We also found that this lesson is meaningful as we could potentially leverage the less costly direct IO during read operation (so far we only test the write operation), where the data can be moved directly from disk to user buffer, while bypassing the kernel buffer. We verified this in the next section.

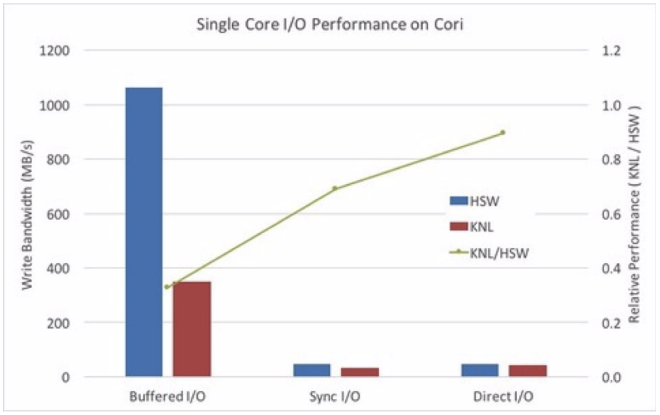

Fig. 13. Performance Ratio between KNL and Haswell with different IO mode, (a different view)

III. MULTI-CORE/MULTI-NODE PERFORMANCE

Having a better understanding of single-core IO performance, in this section, we explore multi-core/node scenarios. Certainly, leveraging many cores on the KNL partition is the way to reach higher bandwidth. We started with a single-node IOR test and scaled from 1 to 68 processes. We also conducted comparisons of point-to-point communication and collective IO performance between Haswell and KNL cores.

A. Single-Node IOR Test

We have enabled synchronous and direct IO separately in this IOR test. This test is file per process, using POSIX API. The IO is 256 segments with a 4MB block and transfer size, so each process is writing 1GB data. In the case of 68 processes, it is 68 files and 1G per file.

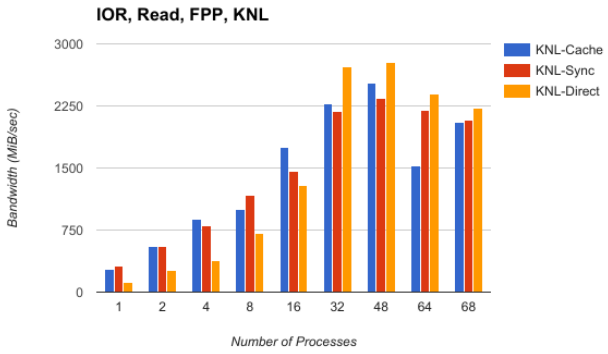

Fig. 14. IOR File Per Process Read on KNL, Weak Scaling on Single Node

From Figure 14 and 15, Overall, the maximum single node aggregated read bandwidth on KNL is 2.78G, and 3.42G

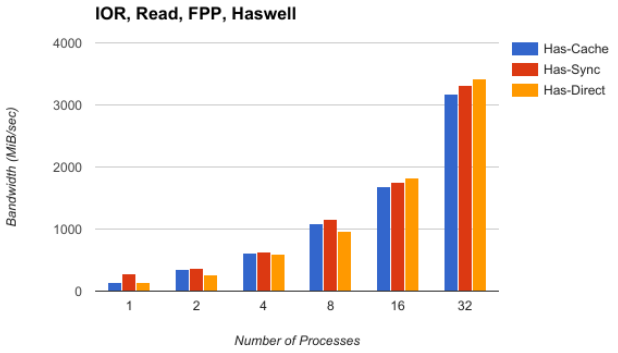

Fig. 15. IOR File Per Process Read on Haswell, Weak Scaling on Single Node

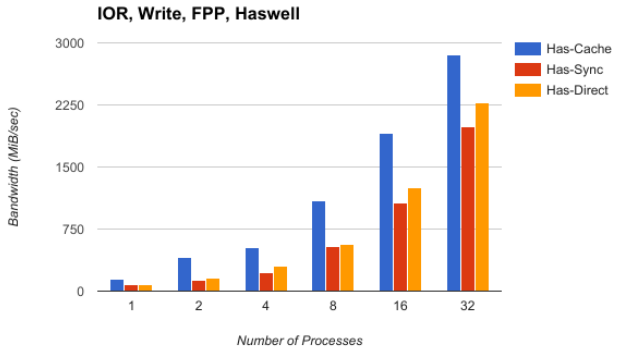

Fig. 17. IOR File Per Process Write on Haswell, Weak Scaling on Single Node

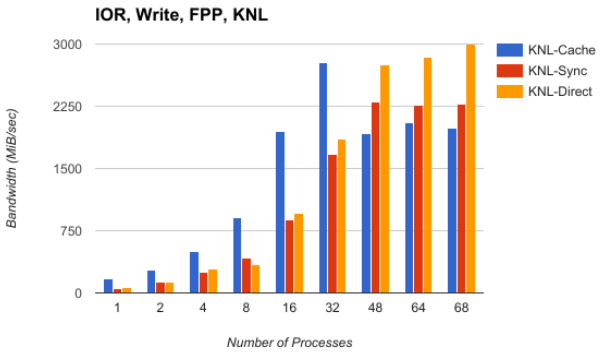

Fig. 16. IOR File Per Process Write on KNL, Weak Scaling on Single Node

on Haswell, both with direct IO. Their performance ratio is 81.2% in direct IO, 70.3% in synchronous IO and 79.6% in buffered IO. Specifically in this file per process IOR read test, KNL achieved maximum bandwidth with 48, i.e., 70% of a single node's total cores, by writing 48 1G files toally with direct IO. Haswell got its maximum bandwidth at 32 processes, 100% of a single node cores, by writing 32 1GB file totally in direct IO. In both cases, direct IO starts to outperform buffered and synchronous I/O at certain number of processes (or certain size of I/O), e.g., 32 processes, 32 GB on KNL and 16 processes, 16 GB on Haswell. Direct IO skips read-ahead, while buffered/synchronous IO leverages that, which is probably the reason that at certain point, **read-ahead benefit diminishes**.

In Figure 16 and 17, we have found that page buffering certainly helps during the write. But KNL is scalable to 68 cores with direct IO, and **outperforms Haswell's maximum write bandwidth.**

B. Point-to-Point Communication

To add some context to the I/O results, we carried out experiments showing the bandwidth achieved for MPI point-to-point communication between nodes and within a node. We targeted two supercomputers at Argonne National Laboratory: Theta and Cooley. Theta, like Cori, is a Cray XC40 supercomputer with more than three thousand Intel KNL 7250 nodes, connected through an Aries dragonfly network with a theoretical bandwidth of 14 GBps between two nodes if these nodes are in the same cabinet. Cooley is an analysis and visualization cluster of 126 nodes, each hosting two 2.4 GHz Intel Haswell E5-2620 processors (with 6 cores each). On Cooley, nodes are interconnected with a CLOS network and a single Infiniband FDR link between two nodes can achieve up to 7 GBps.

1) Bandwidth: To evaluate point-to-point communication performance on both KNL-based and Haswell-based architectures, we ran a *Ping-Pong* test from the Intel MPI Benchmarks suite (IMB) [4] that performs a simple transfer from one rank to another. The data transfer size varies as a power of two between 0 and 64 MB. For these experiments, we mapped our ranks on two different nodes (inter-node communication) and on the same node (intra-node communication). For this second case, we forced the process binding on both platforms to assign processes on two different processors sharing the main memory. It has to be noted that we saw no impact when allocating memory in the HBM on KNL nodes.

Figure 18 presents these results. First, we remark that contrary to Cooley the peak network bandwidth obtained on Theta is far from the theoretical peak performance. Second, on both architectures, we notice a large performance decrease for intra-node communication beyond a certain amount of data sent. Several factors can explain this behavior, such as a change strategy within the MPI implementation.

2) Latency: We also isolated our results with an empty message (0 bytes) to provide a rough estimate of the latency between nodes and within a node. Table IV shows these results. For intra-node communication this latency is almost three times higher on the KNL-based platform. We observe the same behavior for inter-node communication with a latency on Cooley 60% lower than on Theta.

C. MPI IO Performance

To further compare the IO performance between Haswell and KNL with multiple nodes, we developed an MPI-based

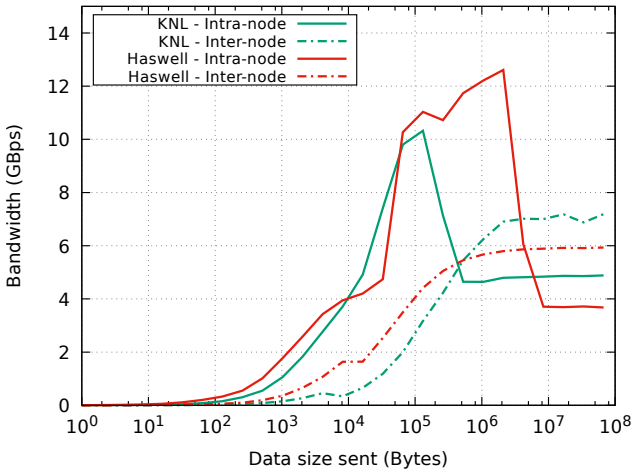

Fig. 18. Bandwidth comparison for KNL-based and Haswell-based nodes for a ping-pong benchmark with intra-node and inter-node communication

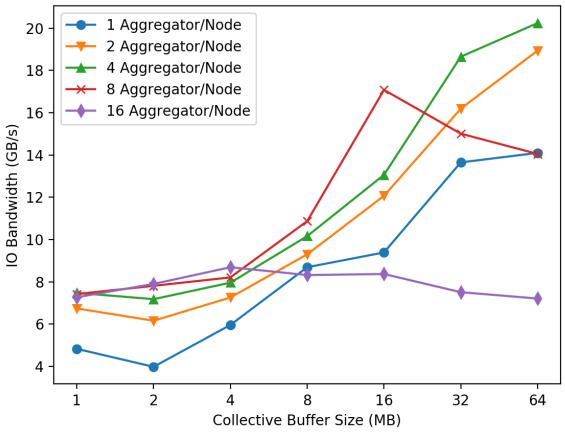

Fig. 19. Haswell IO Bandwidth with different collective buffer size and aggregators per node.

TABLE IV
LATENCY IN μs BASED ON THE SENDING OF A 0 B MESSAGE

	Haswell	KNL
Inter-node	1.97	4.92
Intra-node	0.23	0.63

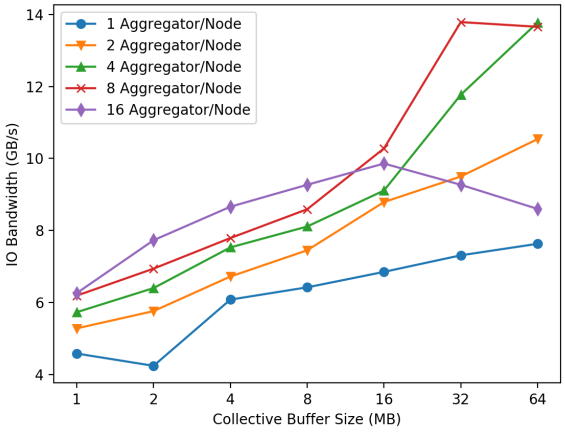

Fig. 20. KNL IO Bandwidth with different collective buffer size and aggregators per node.

micro-benchmark. The benchmark writes a $\approx 486GB$ dataset to the Lustre file system using MPI collective IO, with varying collective buffer sizes and numbers of aggregators. KNL nodes have 68 cores per node, as opposed to 32 for Haswell nodes. For a fair comparison in our experiments, we used 32 MPI ranks per node on both types of nodes, and each with 32 nodes, resulting in a total number of 1024 MPI processes in both cases. Additionally, we set the stripe count to 32 and stripe size equal to the collective buffer size for both.

Figure 19 and 20 show the IO performance with collective buffer size ranging from $1MB$ to $64MB$ and number of IO aggregators per node from 1 to 16. Generally, the IO bandwidth increases with collective buffer size, with a few exceptions for 8 and 16 aggregators per node cases. Thus **it is recommended to use larger collective buffer sizes, as well as Lustre's stripe size**.

The IO bandwidth also increases with the number of aggregators, but only up to 8 aggregators per node. When there are more aggregators, we see a performance decrease as the stripe size gets larger. As a result, **setting** 4 **aggregators per node on Haswell nodes and** 8 **aggregators per node on KNL nodes are recommended to have the best write performance**.

IV. Conclusion and Future Work

In this work, we have studied the IO performance gap between Haswell and KNL nodes. Our testing identified bottlenecks from CPU frequencies, page buffer (memory copying), and network latency. When we set KNL and Haswell to have same frequencies, we still see a large performance gap, until we disable the page buffer. With direct IO, we see a nearly 96% performance ratio of KNL to Haswell on single core tests. We further verified this with IOR and scaled to full ranks on a single node. Page buffering certainly helps in many cases, but in case of reading larger data (larger than page buffer) for just one time, direct read operations, which bypasses the kernel buffer, outperform page buffered and synchronous reads. Page buffer management is slower on KNL is indicated from the comparison of buffered IO bandwidth ration and direct IO bandwidth ratio. Direct read also scales much better than buffered IO. In case of inter-node IO, we have benchmarked the point-to-point communication latency, and have observed 4.92 μsec latency on KNL and 1.97 μsec latency on Haswell nodes. The MPI-IO tests suggest that 2-4 aggregators and 4-8 aggregators per node on Haswell and KNL separately will yield the best write performance. Overall, this study has provided a clear picture of the IO performance between Haswell and KNL cores, and in the future, we would like to perform further

testing to explore heterogeneous IO optimization for many core system.

REFERENCES

[1] Cori at NERSC, http://www.nersc.gov/users/computational-systems/cori/.

[2] Cori File System, http://www.nersc.gov/users/storage-and-file-systems/file-systems/ngfdrawings/.

[3] HDF5 Benchmarks at NERSC, https://github.com/valiantljk/h5spark/tree/master/mpiio.

[4] Intel Mpi Benchmarks, https://software.intel.com/en-us/articles/intel-mpi-benchmarks.

[5] Theta at ANL, https://www.alcf.anl.gov/theta.

[6] T. Barnes, B. Cook, J. Deslippe, D. Doerfler, B. Friesen, Y. He, T. Kurth, T. Koskela, M. Lobet, T. Malas, L. Oliker, A. Ovsyannikov, A. Sarje, J. L. Vay, H. Vincenti, S. Williams, P. Carrier, N. Wichmann, M. Wagner, P. Kent, C. Kerr, and J. Dennis. Evaluating and optimizing the nersc workload on knights landing. In *2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 43–53, Nov 2016.