Title

Evaluating the networking characteristics of the Cray XC-40 Intel Knights Landing-based Cori supercomputer at NERSC

Journal

Concurrency and Computation Practice and Experience, 30(1)

ISSN

1532-0626

Authors

Doerfler, Douglas
Austin, Brian
Cook, Brandon
et al.

Publication Date

2018-01-10

DOI

10.1002/cpe.4297

Peer reviewed

Evaluating the Networking Characteristics of the Cray XC-40 Intel Knights Landing Based Cori Supercomputer at NERSC

Douglas Doerfler, Brian Austin, Brandon Cook, and Jack Deslippe
Lawrence Berkeley National Laboratory
Berkeley, CA
Email: {dwdoerf, baustin, bgcook, jrdeslippe}@lbl.gov

Krishna Kandalla, Peter Mendygral
Cray Inc
St. Paul, MN
Email: {kkandalla, pjm}@cray.com

***Abstract*—There are many potential issues associated with deploying the Intel Xeon Phi™ (code named Knights Landing (KNL)) manycore processor in a large-scale supercomputer. One in particular is the ability to fully utilize the high-speed communications network, given the serial performance of a Xeon Phi™ core is a fraction of a Xeon® core. In this paper we take a look at the tradeoffs associated with allocating enough cores to fully utilize the Aries high-speed network versus cores dedicated to computation, e.g. the tradeoff between MPI and OpenMP. In addition, we evaluate new features of Cray MPI in support of KNL, such as inter-node optimizations and support for KNL's high-speed memory (MCDRAM). We also evaluate one-sided programming models such as Unified Parallel C. We quantify the impact of the above tradeoffs and features using a suite of NERSC applications.**

I. Introduction

With the introduction of the Intel Knights Landing (KNL) as a standalone processor for a supercomputer compute node, application developers have been refactoring their code in order to make best use of KNL's manycore architecture. In many cases this has been taking an MPI-only code and introducing threads, most commonly with OpenMP. Codes that thread well can potentially dedicate only a few cores per node for MPI processes and dedicate the remaining cores to threads. The advantage of reducing the number of MPI processes per node can translate into higher computational efficiencies. For example, it can lead to decreasing surface to volume ratios, which minimizes communication to computation time in boundary exchanges. This in turn leads to increased message sizes with potentially higher network bandwidth and efficiency. In addition, reducing the number of MPI ranks can lead to a reduction of the number of iterations required for convergence in some solvers, for example an algebraic multilevel preconditioner. [1] However, there is also a need to have a sufficient number of processes doing communications in order to fully utilize the resources of the high-speed network interface (NIC). Historically, the later has not been an issue with multicore architectures and MPI-only codes as there is a one-to-one mapping of MPI processes to processor cores. And for MPI+threads hybrid codes the traditional "heavy-weight" core, such as the Intel x86-64 Xeon®, are sufficiently high performant to fully utilize the NIC. However, the Xeon Phi™ core used in KNL is a fraction of the performance of a current generation Xeon® core.

The purpose of this paper is to gain a better understanding of how to fully utilize the high-speed interconnect on a KNL based node and compare that to a traditional Xeon® based node. In addition, we will investigate features of Cray MPI that have been developed to improve performance on KNL such as optimizations for inter-node communication and support for high-speed memory (MCDRAM). We will also look at one-sided programming models, as we believe these will become more prevalent and indicative of the types of networking features of future programming models.

In our analysis we will use microbenchmarks to investigate performance of specific communication primitives and then use real applications to demonstrate the impact at scale on the Cray XC based Cori and Edison supercomputers at NERSC.

II. Related Work

Barrett et al. compared the MPI message rate performance on a variety of simple and complex cores and found substantial benefit from out of order processing. [2] Barrett's paper focuses on MPI benchmark performance to illuminate the architectural features that contribute to message rate performance. In contrast, we examine the scaling behavior of real applications with complex communication patterns and (possibly threaded) phases of computational work interleaved with communication.

Shan et al. modified the communication functions in the MILC and IMPACT-T applications to use UPC in place of MPI and measured substantial speedups due to the use of lightweight one-sided messaging. [3] Their work was performance on a Cray XE-6 and did not compare the communication performance of different processor cores.

III. Target Platforms

The Edison and Cori supercomputers are both sited at the U.S. Department of Energy's Office of Science National Energy Research Scientific Computing Center (NERSC). [4],

[5] Edison, deployed in 2013, is based on the Cray XC-30 architecture featuring the Aries high-speed interconnect and uses Intel Ivy Bridge processors. Cori is NERSC's latest platform and is based on the Cray XC-40 architecture, again featuring Aries. Although Cori contains both Intel Haswell and KNL nodes, for this study we only look at the KNL based partition.

The KNL has many different internal cluster modes and NUMA configurations for its MCDRAM high-speed memory. Currently, Cori is configured primarily in the *quad* mode, and the MCDRAM is configured as a *cache* for the DDR memory. Unless otherwise stated, the results of this study use the *quad, cache* configuration.

We use Edison as a comparison platform because it is larger (in node count) than the Cori Haswell partition and allows for large scale comparisons. It is also the platform in which many of the NERSC applications will be transitioning from. Detailed platform information can be found at the respective NERSC web sites. [5]

IV. Microbenchmarks

A. MPI Microbenchmarks

1) Point-to-Point, single rank pair: We first look at the fundamental MPI characteristics of latency, bandwidth and message rate. We use OSU Micro-Benchmarks from The Ohio State University [6] to measure ping-pong latency and bandwidth, in addition to multi-pair "streaming" bandwidth between two nodes.

Ping-pong latency (*osu_latency*) for small message sizes for Cori was observed to be ∼2.6x that of Edison, 3.1 vs. 1.2 $\mu seconds$. This ratio is maintained till the message size reaches 8 KiB, when a protocol change is observed for larger message sizes. We tested *quad, flat* mode on Cori but found no difference in latency when compared *quad, cache*.

Bandwidth results are shown in Figure 1 for a ping-pong test (derived from *osu_latency*, and labeled pp), a streaming uni-directional test (*osu_bw*, bw), and a streaming bi-directional test (*osu_bibw*, bibw). The Edison results are consistent with that reported by Cray for Aries [7], with a maximum sustained bandwidth of 9.8 GB/s for ping-pong, 9.9 GB/s for uni-directional, and 15.6 GB/s for bi-directional. At a message size of 1 MiB, the observed bandwidth of Edison is ∼1.3x higher for all three tests. It's an interesting observation that on Cori peak sustained bandwidth of the interconnect is never achieved using a single core per node. We found no difference in bandwidth when comparing *quad, cache* and *quad, flat* KNL modes.

2) Point-to-Point, multi-rank pair: To better understand how many KNL cores it takes to fully saturate bandwidth between two nodes, we used the multi-pair bandwidth test (*osu_mbw_mr*) and increased the number of pairs until we are fully utilizing the cores on the respective node types. These results are shown in Figure 2. Using this method, in addition to being able to observe the respective bandwidth characteristics, we can also observe the effect of increasing message rates achieved for a given message size. As rank-pairs per node (RPN) increases, the bandwidth curve shifts to the left because the interconnect is being better utilized with a higher message rate for a given message size and hence achieving a higher effective bandwidth.

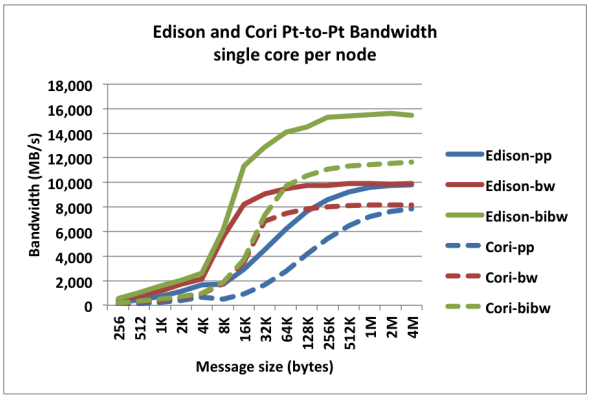


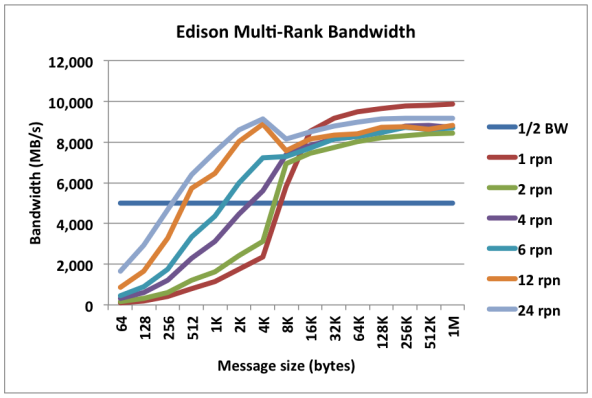
Figure 1. Ping-pong (pp), Uni-directional (bw) and bi-directional (bibw) bandwidth for Edison and Cori

For a single rank-pair (core-pair), Cori achieves 1/2 of the maximum sustained bandwidth with a message size between 16 KiB and 32 KiB. As rank-pairs are added 1/2 bandwidth is achieved with a message size of 128 B using 64 rank-pairs. For Edison, the required message sizes are 8 KiB using one rank-pair and 256 B using 24 rank-pairs. Although Cori achieves 1/2 bandwidth at approximately the same message size as Edison when both use all available cores, at low rank-pair counts Cori needs 2 to 4 times the number of cores to achieve a similar bandwidth as Edison. For example, using a message size of 4 KiB Edison reaches 1/2 bandwidth at 4 RPN while Cori requires at least 8 RPN. Again, We found no difference in bandwidth when comparing *quad, cache* and *quad, flat* KNL modes.

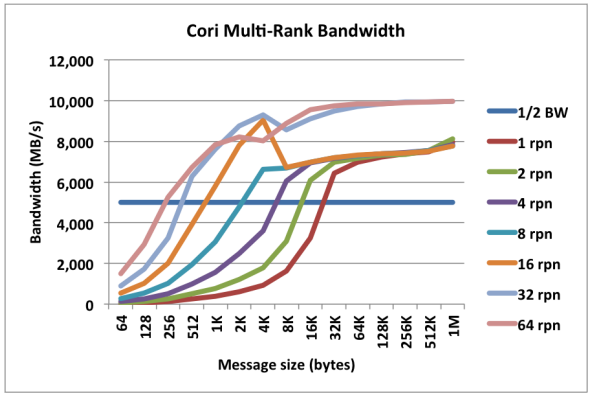
We also observed that Cori was unable to achieve full bandwidth using large messages below 16 RPN. Cray analysis concludes that the latency from the PCIe interface to memory on KNL is higher than it is on Intel Xeon® processors. More MPI processes on a node are required on KNL to hide the higher latency and obtain peak bandwidth, as shown in Figure 2. It is possible to improve observed bandwidth for rank pair-based tests with 16 or fewer ranks per node by lowering the value of the MPICH_GNI_NDREG_MAXSIZE environment variable (we set it to 65536, where the default is 4 MB), which is the threshold for switching to the PUT-based RDMA protocol. This may improve performance for some applications running on KNL, but can lead to performance reductions as well. Users can experiment with this setting to see if a lower threshold is beneficial for their application.

Table I
NERSC CORI AND EDISON HIGH-LEVEL ARCHITECTURAL FEATURES

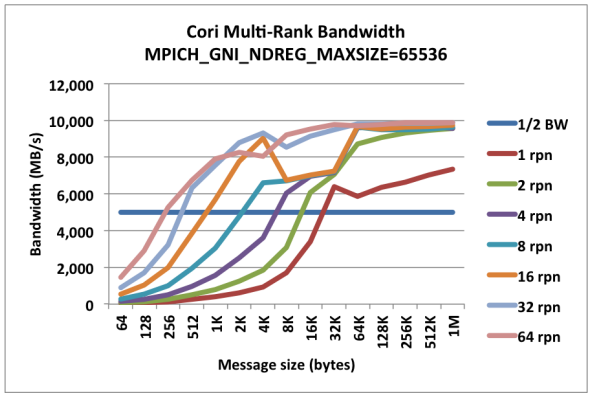
	Architecture	Interconnect	Node Processor	Number of Nodes	Compile Environment
Edison	Cray XC30	Cray Aries	2x IVB, 12-core, E5-2695V2	5,200	Intel 17.0.1.132
Cori	Cray XC40	Cray Aries	1x KNL, 68 core, 7250	9,304	Intel 17.0.2.174



(a) Edison



(b) Cori



(c) Cori with MPICH_GNI_NDREG_MAXSIZE=65536

Figure 2. Multiple rank-pair bandwidth, as ranks pairs are added, message rate increases and as does effective bandwidth

3) Multi-node, multi-rank pair: We also analyzed the more complex messaging patterns found in the Sandia MPI Micro-Benchmark (SMB) message rate benchmark [8]. This benchmark differs from OSU benchmark in that it communications between multiple node-pairs simultaneously, modeling the pattern found in many stencil based codes. We chose to use the nominal number of 6 peers per rank and varied the message size similar to the OSU tests. The SMB benchmark also runs several tests, but we chose to focus on the *pair-based* test as we feel it most nearly resembles what happens in production codes. Results are shown in Figure 3.

What we found was a significant drop in performance with message sizes above 8 KiB. We identified that when using SMB the message injection rate is sufficient to trigger translation lookaside buffer (TLB) thrashing in the Aries interconnect. Cray's recommendation in this situation is to use huge pages. Using the craype-hugepages2M module, which defaults the use of 2 MiB pages for the application, we reran the test and found that this indeed did improve performance substantially. Although we only show results for Cori, we also observed similar performance with Edison. But the effect of the many-core architecture of a KNL node is more dramatic. In addition, as was observed with the OSU multi-rank benchmark, Cori requires $\sim$2x the number of cores to reach the same effective bandwidth as Edison.

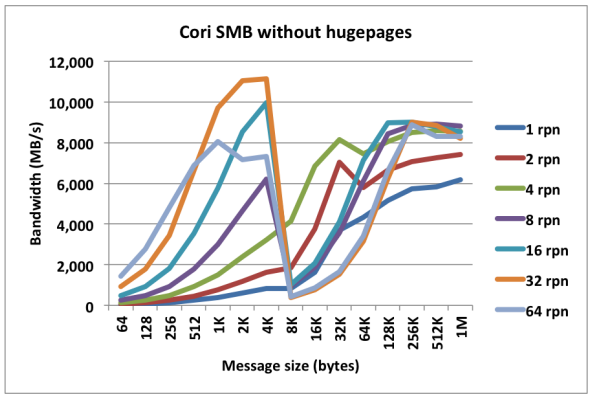
We will refer back to the observations from this section when we do an analogous study with applications and observe how many MPI ranks it takes to achieve the best performance.

B. UPC Microbenchmarks

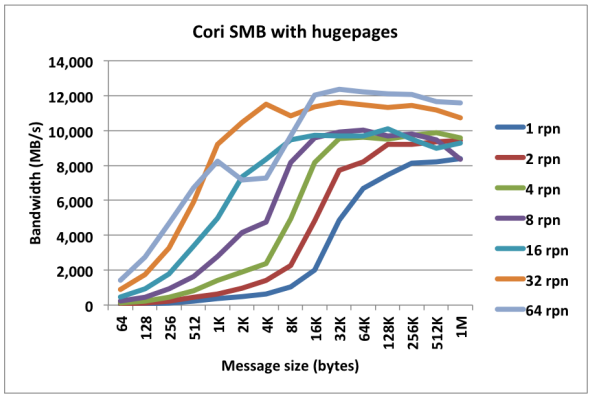
1) Point-to-Point, multi-rank pair: The OSU microbenchmarks were used to measure the UPC latency for multiple pairs of threads distributed evenly across two nodes. We modified the *osu_upc_memput* and *osu_upc_memget* benchmarks so that their communication patterns matched the MPI multi-latency tests. Specifically, each thread on $node-0$ puts to (or gets from) its peer on $node-1$.

On Cori, the small message latency for a single pair of threads was 2.17 μs (put) and 2.46 μs (get). Consistent with its more performant cores, the UPC latency on Edison was faster than Cori: 1.10 μs (put) and 1.43 μs (get). For the same reason, lightweight communication protocols have greater benefit on Cori: Edison's MPI latency is only 1.1$\times$ faster than its UPC-put latency, but there is a difference of 1.4$\times$ on Cori.

UPC uni-directional bandwidth results were inferred from the latency measurements reported by the *osu_upc_memput*

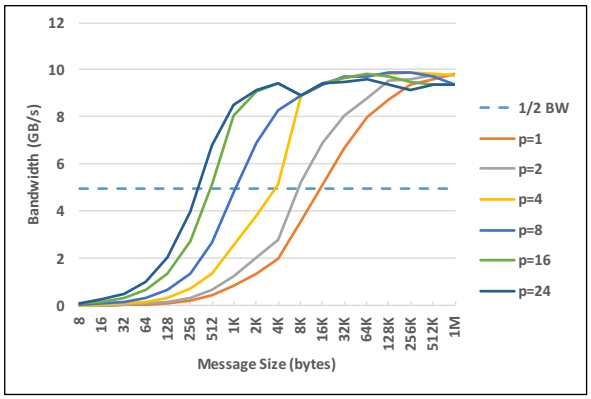


(a) Without huge pages

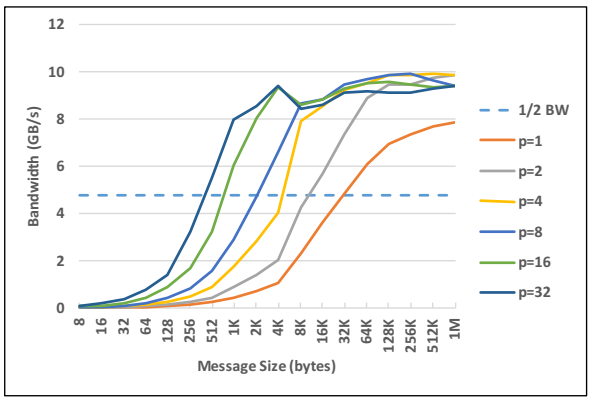


(b) With huge pages

Figure 3. Sandia MPI Micro-benchmark *msgrate* results on Cori. Using 2 MiB pages improves performance substantially for message sizes $>$ 8 KiB.



(a) Edison



(b) Cori

Figure 4. UPC uni-directional "put" bandwidth."Get" bandwidths (not shown) are similar, but are 10% lower for all message sizes and pair counts.

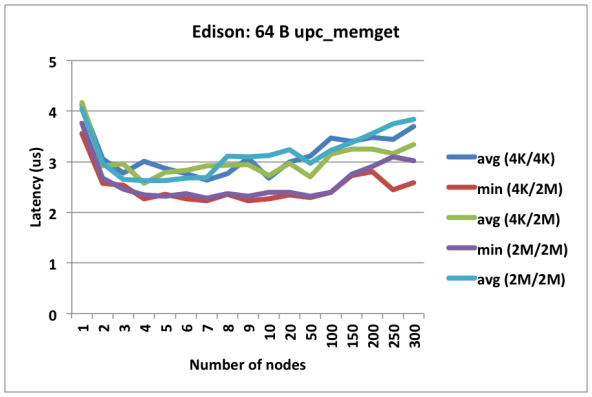
benchmark and are shown in Figure 4. The maximum sustained bandwidth on Cori and Edison is 9.9 GB/s. On Edison, this limit can be reached using only one UPC thread per node, but a single Cori thread can put only 7.9 GB/s and two or more threads must be used to attain peak bandwidth. To acheive half-bandwidth using only one thread-pair, UPC messages must be at least 32 KiB (Cori) or 16 KiB (Edison). The half-bandwidth threshold generally shifts to smaller messages as more threads are used on each node.

2) Meraculous microbenchmarks: We also examine the behavior of three microbenchmarks which correspond to kernels in Meraculous. Each kernel is iterated 20,000 times and the total time to complete 20,000 iterations is used to compute the average latency and bandwidth of the UPC thread. The min, max and average of all threads in a given run is reported. The benchmarks were all run with a fixed total memory footprint of 4GB and using 1 UPC thread/process per physical core. Measurements were done with the bupc-GB2.24.2 and bupc-GB_4K2.24.2 which utilize Cray Aries remote atomics instead of the usual Active Messages implementation. The bupc-GB2.24.2 module uses 2M pages for the UPC shared-heap data and 4K pages for application data and bupc-GB_4K2.24.2 uses 4K pages everywhere.

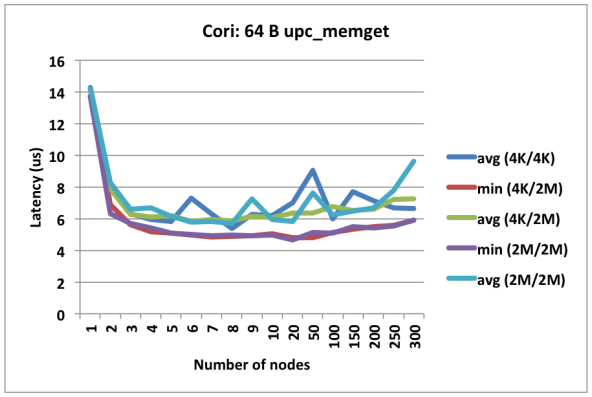
The first benchmark ("construct") is a proxy for the construction of the distributed hash table representation of the de Bruijn graph. Each UPC thread does an *atomic_fetch_and_add* operation to reserve space in a random other thread's heap, followed by a *upc_memput* operation, Figure 5. In this test the size of the data sent by the *upc_memput* is varied. The second microbenchmark is a proxy for retrieving elements from the table, where an iteration for a given size consists of a *upc_memget* operation from a random location in a random thread's shared buffer, Figure 6. The final benchmark is a proxy for the traversal of the graph with a *upc_memget* operation of a random entry from a random thread is followed by an *atomic_compare_and_swap* operation, Figure 7.

Within a single node all benchmarks quickly reach peak bandwidth with 2048 byte messages with *upc_memput* operations showing better performance than *upc_memget* operations. Further increases in message size result in consistent high sustained throughput. This behavior is largely a function of local shared memory characteristics and is not the focus of this work. The probability that a upc_memget is

of an element from a UPC thread which is on the same node is $P = 1/N$ where N is the number of nodes independent of the number of cores per node. This means that with more cores per node the effects of local interference on Cori are higher than Edison at low node counts. At higher node counts the birthday paradox plays a role since the total footprint of the shared memory locations is constant the probability of two UPC threads selecting the same target increases as the node count increases. In the case of small gets this can be a benefit since the footprint is smaller and element will likely be already in cache, however with atomic operations the possibility of conflicts is increased. In particular for the construct benchmark as the message size is increased this means more UPC threads will be competing for bandwidth, another problem which is enhanced by higher core counts. For the traversal benchmark this is less of an effect due to the small message size.



(a) Edison

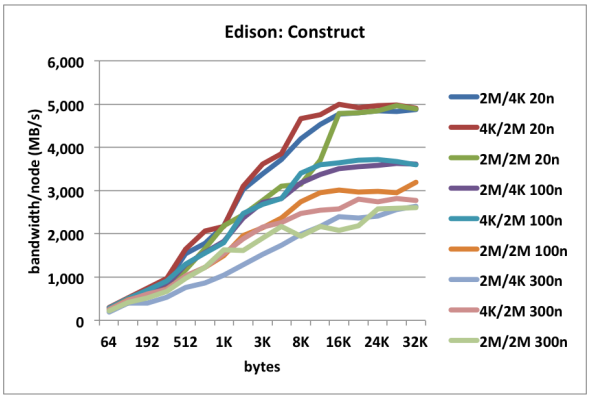


(b) Cori

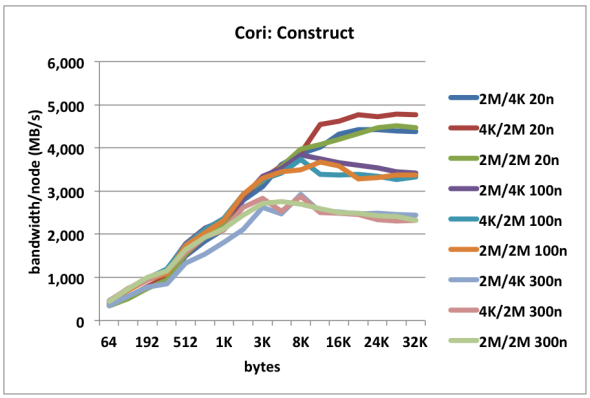
Figure 5. 64 byte upc_memget into 4GB random data distributed evenly among nodes

V. Applications

In this section we look at the trade off between MPI and OpenMP at the node level and the strong scaling characteristics of each application. The number of cores utilized per node is fixed (64 for Cori, 24 for Edison), but



(a) Edison



(b) Cori

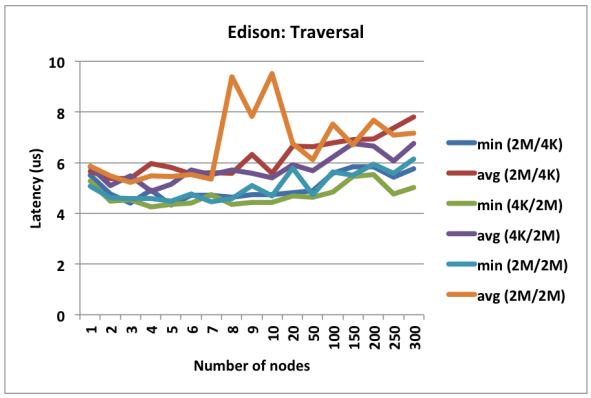
Figure 6. remote atomic fetch and add followed by upc_memput into 4GB random data distributed evenly among nodes

the cores are appropriated between MPI tasks / OpenMP threads in decreasing / increasing order such that the left most data point in the following figures is MPI-only across all cores, and the right most data point is one MPI rank per socket with the remaining cores dedicated to OpenMP. This method also provides a view of how well an application scales in MPI and OpenMP. The left side of the curve is MPI dominant, and the right side is OpenMP dominant, so at the extremes we can get an indications of how well each scales by looking at the respective trends.

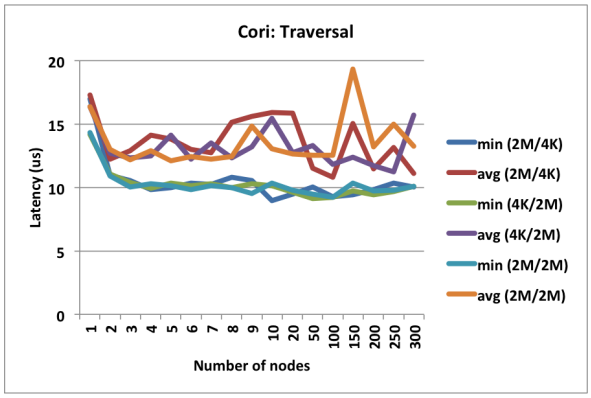
We do not take advantage of the available core hyper-threads, primarily so as not to introduce another variable to consider in the OpenMP scaling analysis. This may be a slight disadvantage in absolute performance for the KNL architecture, as many codes do benefit from an additional one or two hyper-thread per core.

We also look at strong scaling. In general, as you strong scale out the computation / communication ratio decreases and the interconnect performance becomes more dominant. Viewing the strong scaling differences between Edison and Cori will give us another indication of the impact of a the KNL many-core architecture.

To better understand the messaging characteristics of each



(a) Edison



(b) Cori

Figure 7. 64 bytes upc_memget from 4GB random data distributed evenly among nodes followed by remote atomic compare and swap

application, we also use the Integrated Performance Monitoring (IPM) tool [9] to do some basic MPI profiling to better understand the amount of time spent in communication and the message sizes.

A. MILC

The MILC code is a widely used, computationally intense application designed to compute the interactions of quarks and gluons as described by the theory of quantum chromodynamics (QCD). The computational grid is a four-dimensional space-time grid (x, y, z, t) with quark fields, defined as three-dimensional complex vectors, at the grid points and gluon variables, defined as 3x3 unitary matrices, defined at the 'links' between grid points [10]. The most computationally intense part is the conjugate gradient solver which determines how the motion of the quarks is affected by the gluons [10]. The four dimensional lattice is decomposed so that the sub-grid assigned to each MPI task has the minimum possible surface-to-volume ratio. The code has fine-grain parallelism implemented with OpenMP directives, mostly on loops over all grid points in the lattice [11]. Communications in the MILC code are largely dominated by point-to-point transfers associated with the 4D halo exchanges and global reductions associated with the conjugate gradient solver.

For this study, we primarily focus on the CG solver performance and in particular the high-speed communications characteristics. We use a "warmed up" and equilibrated lattice of size 128^4 as supplied by the APEX benchmark suite [12] to ensure a realistic and computationally challenging input problem. We collected data using a strong scaling methodology at 256, 512 and 1024 nodes. The MILC code automatically decomposes the problem for the given node count. At 256 nodes, this equates to a lattice size of 32^4 per node. Although this may be a somewhat smaller lattice decomposition per node than would be used in production, it's desirable for this study in that the lower surface to volume ratio stresses the network and lets us better study the network effects. For 512 nodes, the lattice size per node is $16x32x32x32$ and at 1728 nodes it's $16x16x32x32$, these two cases further stressing the network.

Results are shown in Figure 8. Although the Edison and Cori curves show a similar trend, Edison's performance doesn't degrade as quickly as OpenMP parallelism increases (MPI parallelism decreases), which is an indication of its higher per core performance on non-OpenMP regions and its ability to fully drive the interconnect with fewer MPI ranks (cores) as was shown in Section IV-A. For Edison, the best performance is obtained using 16 RPN (MPI-only) at all scales.

Cori also performs best using MPI-only at 256 and 512 nodes with performance dropping off when OpenMP is employed. At 1024 nodes, time spent in MPI becomes more dominant and the performance is relatively flat out to 8 MPI ranks but then tails off when $<$ 8 ranks (cores) are used. Beyond 4 RPN the performance drops off dramatically. The decrease in performance is due to either a lack of OpenMP parallelism, or a lower effective MPI performance due to fewer cores driving the interconnect. Although it's difficult to quantify the contribution of these two issues, one can see that as scale increases, and hence the fraction of time spent in communication increases, the relative performance degradation seen at the right hand side of the curve is less which allows us to imply that a lack of OpenMP performance at the smaller scales is most likely the dominant contributer.

IPM profiling showed that message sizes vary depending on the scale of the the run and the number of ranks per node. At 256 nodes and 64 RPN, message sizes for point-to-point MPI calls were 8 KiB. Each doubling of the number of MPI ranks per node increases the message size by an equal amount such that at 1 RPN the message sizes are on the order of 512 KiB. The SMB message rate benchmark analyzed in Section IV-A is indicative of the communication pattern used by MILC (i.e. a multi-dimensional stencil boundary exchange) and in that analysis of Cori we need at least 4 MPI ranks to drive the interconnect at a reasonable effective bandwidth using small message sizes. This same

characteristic can be seen in the MILC results.

As seen above, both machines scale very well using MPI-only and remain relatively flat out 8 to 12 threads. Selecting these particular combinations we plotted the strong scaling characteristics in Figure 9. At 256 nodes, computation time is dominant but as scale increases MILC becomes more communication bound and we see the performance of both platforms approach a similar performance. In addition, it can be seen that Edison and Cori scale relatively the same, showing that the many-core Cori/KNL architecture is able to fully utilized the network, as long as a sufficient number of MPI ranks per node are utilized.

In Section IV-A3 it was demonstrated that using huge pages can have a significant impact on MPI performance when using 8 or more RPN. Using 2 MiB pages with MILC on Cori showed a performance improvement of 10% when using 64 RPN at 16 nodes. We ran a test problem at 432 nodes and saw an improvement of 44% at 64 RPN and continuing to 8% at 8 RPN, demonstrating the sensitivity of MILC to message rate characteristics at large scale. For this reason, all the tests in this study were run using the Cray 2 MiB huge pages module (*i.e. module load craype-hugepages2M*).

In summary, Cori shows a significant overall performance improvement compared to Edison at the three scales evaluated when using at least 4 to 8 RPN. Comparing best times for each platform at a given scale, Cori demonstrates a speedup of $1.30\times$, $1.25\times$ and $1.26\times$ at 256, 512 and 1024 nodes respectively. It has also been observed that at least 4 RPN are necessary to get the best performance on Cori and as scale decreases and problem size per node increases at least 16 RPN is required due to the change in message sizes and message rate (more ranks per node) and hence number of RPN to achieve a high effective interconnect bandwidth. MPI-only performance is very good on Cori, while perhaps not surprising as MILC was initially developed as MPI-only, it does show that MPI-only is an effective method on the KNL many-core node architecture.

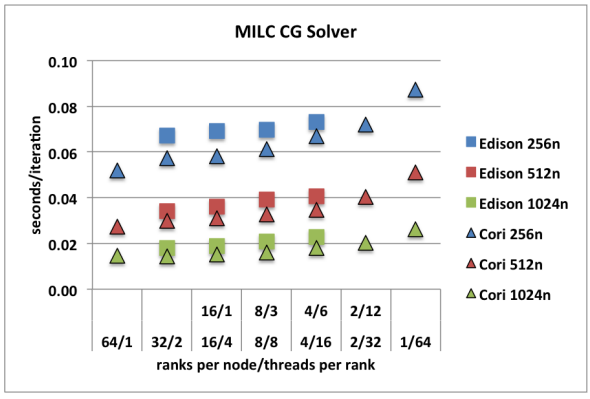


Figure 8. MILC MPI/OpenMP trade off

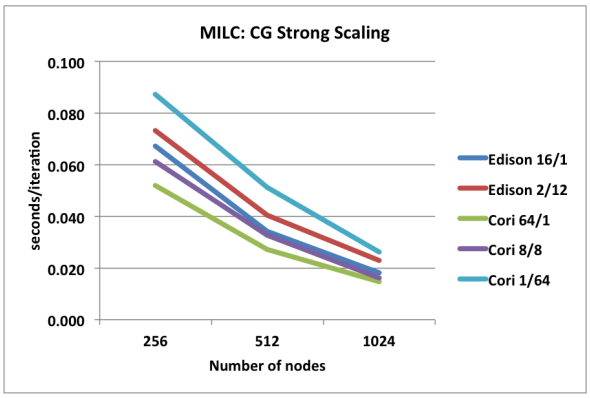


Figure 9. MILC strong scaling characteristics for selected MPI/OpenMP combinations

B. Berkeley GW

BerkeleyGW [13] is a material science application that is dominated by dense linear algebra (including distributed matrix multiplication, inversion, diagonalization, and contraction) and fast fourier transforms (FFT). In this way, it is an ideal proxy for a large number of material science and chemistry applications used at NERSC including top DFT applications like VASP and Quantum ESPRESSO. BerkeleyGW itself is used on top of density functional theory (DFT) applications like Quantum ESPRESSO and PARATEC to compute excited state properties of materials (e.g. band gaps, absorption spectroscopy etc.), though its computational cost and scale typically greatly outstrips that of the DFT codes.

We focus in this section on the scaling of the main bottleneck in BerkeleyGW runs, the calculation of the the electronic polarizability. It involves a distributed matrix-matrix multiply of N^2xN matrices. So, an $O(N^4)$ operation where N scales with the number of atoms in the system. The parallel ZGEMM is hand coded because various preprocessing steps need to be done on the matrices and communication is done via an MPI_REDUCE statement on the final NxN matrix - distributed in block-cyclic form for later BLACS/PBLAS/ScaLAPACK operations. In the current implementation, there is a reduce done for each rank, such that the number of reduce statements increases and the size of the reduce decreases with increasing MPI ranks. The total matrix size is 17GiB. So, the average message size is 17GiB / (number of MPI Ranks).

Figure 11 shows the strong scaling of the code on Edison and Cori. Considering that the code is doing predominantly a distributed ZGEMM, the performance on Cori significantly outperforms Edison at low node-counts. At larger system sizes, however, the gap is reduced as the network plays a bigger role in the calculation, but Cori is still able to drive the network and not give up its performance advantage.

It is additionally interesting to note that, like some of the

other applications discussed here, using fewer MPI ranks per node than the number of cores gives optimal performance - as illustrated in figure Figure 10. Additionally, on Cori, one can see that as you go to larger node counts the optimal ratio of MPI ranks to OpenMP threads decreases - meaning more OpenMP threads and fewer MPI ranks is preferred. This is related to to the reduced number of messages of larger sizes generated when running with threads. It is also possibly a feature in the MPI_REDUCE implementation on Aries. Doing the on-node reduce explicitly via OpenMP outperforms the implicit on-node reduce in the MPI_REDUCE command.

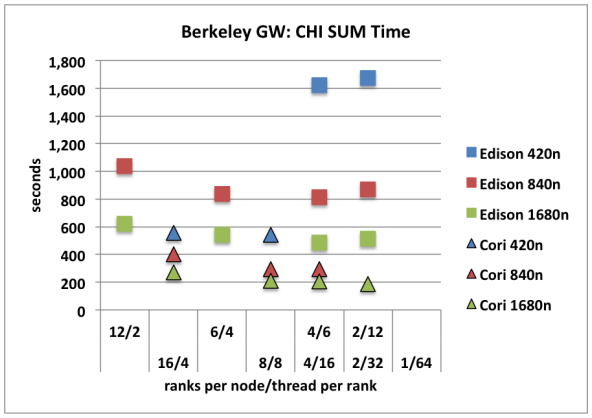


Figure 10. BerkeleyGW MPI/OpenMP trade off

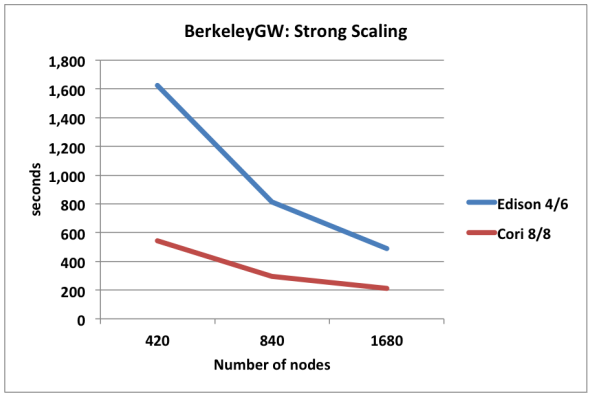
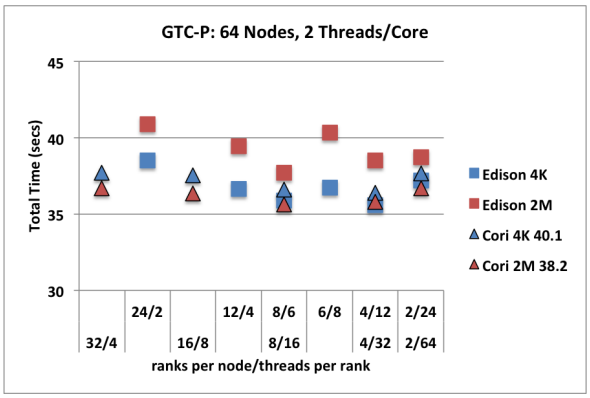


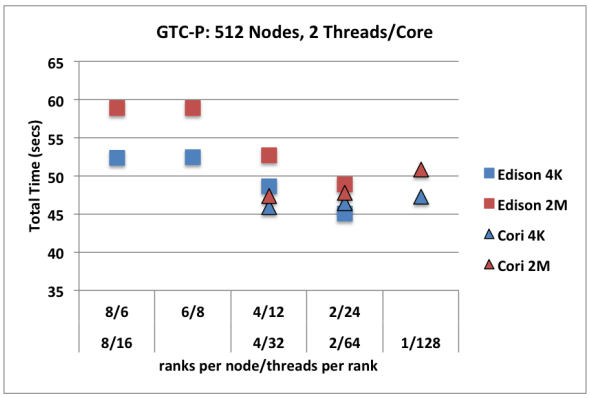
Figure 11. BerkeleyGW strong scaling charateristcs using the best MPI/OpenMP combinations

C. GTC-P

The Gyrokinetic Toroidal Code of Princeton (GTC-P) simulates Tokomak Fusion devices using a particle-in-cell algorithm. The MPI decomposition of GTC-P first slices the spatial domain (and associated grid points and particles) into 64 toroidal slices, and futher subdivides these domains into concentric radial domains. An additional level of on-node parallelism is acheived by threading over particles. The computational profile of GTC-P is dominated by a "charge" phase, that deposits charge from the particles to the grid, and a "push" phase, that interpolates the electric field from the grid to the particles and updates the particle positions based on that field. Both of these phases are characterized by irregular memory access patterns that arise from indirect access of the grid points. After the particles have been pushed, the "shift" phase moves particles to the appropriate spatial domain (process) and is responsible for most MPI communication. Figure Figure 12 shows the trade off between MPI process and OpenMP threads. At 64 nodes, Cori's performance has a broad maximum between 4 and 8 MPI ranks per node and 32-16 OpenMP threads per rank. With 128 threads per process, GTC-P performance is only ***X%*** less than the optimal configuration, reflecting the excellent thread scaling of GTC-P. The communication time (not shown) decreases uniform ally as thread counts increase because fewer particles cross the boundaries between spatial domains when the domains are larger. For all of these configurations, both Edison and Cori perform marginally (5%) better with 4k pages than with 2M pages.



(a) 64 Nodes



(b) 512 Nodes

Figure 12. GTC-P MPI/OpenMP trade off

Figure Figure 13 shows the strong scaling behavior of GTC-P. Edison used 8 ranks per node and 6 threads per rank,

while Cori used 8 ranks per node and 16 threads per rank. At lower concurrencies, Edison is approximately 75% faster than Cori, but this advantage diminishes at larger scales and the two systems are nearly on par at 512 nodes. In this strong scaling study, 2M pages led to about ≈15% better performance than 4k pages,.

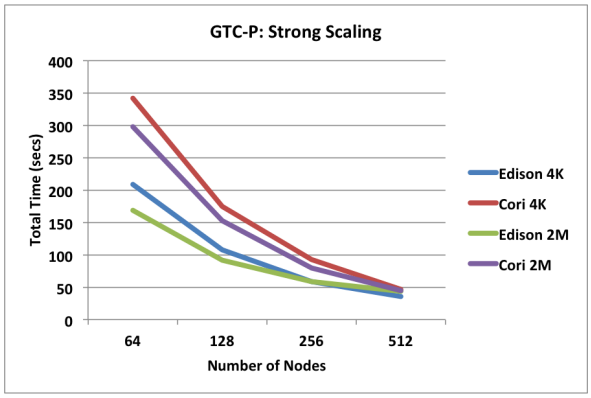


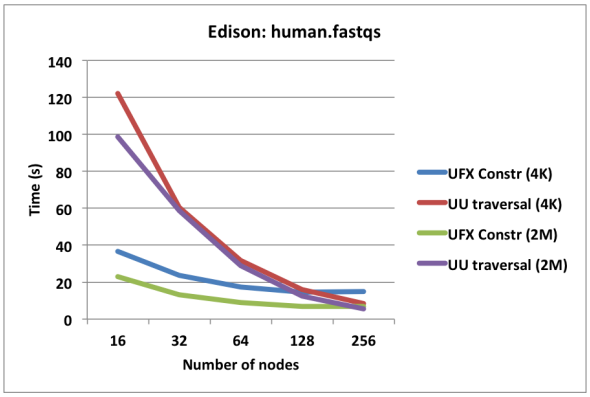
Figure 13. GTC-P strong scaling charateristcs using the best MPI/OpenMP combinations

D. Hipmer/Meraculous

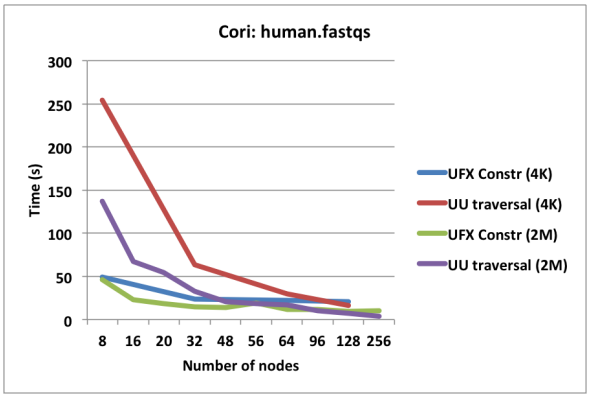
The APEX application Meraculous represents a portion of the de novo genome assembly pipeline in HipMer [14]. Fine-grained random access is a typical feature in this communication heavy pipeline which implements a variety of graph al Figure 14 shows strong scaling studies of the APEX version of the Meraculous application run with the 107 GiB *human.fastqs.ufx.bin.trim.min3* dataset. All runs were conducted with Berkeley UPC 2.24.2 builds which utilized the Aries hardware remote atomics instead of the default Berkeley UPC Active Messages implementation. Cray UPC was not used because the benchmark as written relies on specific Berkeley UPC functionality. The curves labeled as "4K" use 4 KiB pages for both UPC shared segments and application memory allocations. Curves labeled "2M" utilized 2 MiB pages for UPC shared segments and 4 KiB pages for application buffers.

On both Edison and Cori the traversal of the UU graph shows near ideal scaling with the number of nodes. The traversal microbenchmark proxy for the operations performed in this phase shows that the latency of the fixed size get followed by a remote atomic operation is nearly independent of the number of nodes in the calculation. On Cori KNL the effect of huge pages is very strong on this latency sensitive kernel due to the increased number of UPC threads per node putting additional pressure on the translation lookaside buffer (TLB) resources of both the CPU and the Aries network interface. As the application is scaled and the memory footprint per node is reduced the TLB pressure is reduced and the probability of a last level cache (LLC) hit is increased. However, the latency increases due to the potential for more network hops in the Dragonfly topology (potentially outside of a single Aries group) and the greater potential for interference from a busy production system.

The use of huge pages also increases the performance of the construction of the UFX hash table kernel on both Edison and Cori. However, the scalability of this kernel shows strong degradation on Cori vs. Edison. The reason for this is the default setting for the APEX application is 100 for the number of aggregate k-mers before reserving remote space and copying them over. With a fixed number of k-mers and nodes but more nodes per core, more memory operations are required and stride of those arrays is distributed further with some data structures proportional to the number of UPC threads which is $68/24 \approx 2.8\times$ higher on Cori. The higher UPC thread count also results in increased contention for reservation of space in destination buffers on Cori relative to Edison. Additionally as the concurrency is increased the probability of multiple UPC threads accessing the same location in a fixed size table increases. This increased probability combined with larger message sizes leads to an increased contention for bandwidth.



(a) Edison



(b) Cori

Figure 14. Strong scaling with 24 UPC threads (Edison) and 64 UPC threads (Cori) per node using the human.fastqs dataset.

VI. Conclusion

The primary premise of the paper was to investigate and quantify the use of the Intel Xeon Phi™ Knights Landing processor based node on the Cray XC-40 Cori supercomputer at NERSC. The method involved comparing Cori's KNL based partition to NERSC's Edison supercomputer which uses Intel Xeon® Ivy Bridge processors, and the same Cray XC-40 architecture, in particular the Aries high-speed interconnect. Thus, the difference in performance traits between the two platforms can for the most part be attributed to the different processor architectures. We looked at micro-benchmarks and real applications used at NERSC. NERSC applications primarily use the MPI with OpenMP programming paradigm, and hence we looked at the trade off of partitioning a node's cores between MPI ranks and OpenMP threads. In addition, we also investigated the performance of one-sided communications by looking at several UPC benchmarks and one application.

It was expected that in order to get maximum performance with Cori, we would need to use multiple MPI ranks in order to fully utilize the Aries high speed interconnect. But we didn't know to what degree. Using MPI micro-benchmarks, it was shown that with Cori you need anywhere from $2\times$ to $4\times$ the number of MPI ranks (cores) per node in order to achieve the same performance, i.e. effective bandwidth at a given message size, as Edison. We also identified some deficiencies in performance when using the default Cray MPICH configuration parameters. For example, when using a large number of MPI ranks on a node in a stencil based communication pattern, there is the chance you can run into an issue with the Aries interconnect thrashing its TLB when moving data from the network to/from on-node memory. This was mitigated by using 2 MiB huge pages. We demonstrated that using huge pages with the MILC application showed anywhere from a 10% to 40% improvement in performance. In addition, we found that you need to use 32 or more cores of Cori in order to fully drive the high-speed interconnect, but a change in a Cray MPICH environment variable allows lower core counts to get full bandwidth. But changing environment variables to improve upon a micro-benchmark may not necessarily translate into improved real application performance, and in many cases can hurt. Although in our analysis, it neither helped nor hurt performance our applications.

The MPI application performance analysis shadowed what we observed with the micro-benchmarks. Once you have a sufficient number of MPI ranks per node to adequately drive the interconnect, the overall performance of the application depends on its MPI and OpenMP scalability. If a code threads well, e.g. GTC-P, then we found a sweet spot in the MPI vs. OpenMP trade off that gave best performance. For codes that don't thread scale well, e.g. MILC, the best performance was found to be MPI-only. However, it should be noted that without huge pages MPI-only performance can be limited.

In our strong scaling study, we found that at the lower node counts Cori provided the best overall performance, but as the applications were scaled out to a larger node count communication becomes dominant and that advantage was narrowed. Which is actually a positive finding for Cori, as we showed that at extreme scales the Xeon Phi™ architecture doesn't give up performance.

Our investigation of UPC performance was solely evaluating the Berkeley UPC implementation, because it contains certain non-standard features required by the application Meraculous. With micro-benchmarks, we again demonstrated the advantage of using 2 MiB pages, for both the application and with Berkeley UPC internally. Latency micro-benchmarks showed that Cori does show a $2-3\times$ disadvantage to Edison. However due to excelent scaling of the primary traversal stage the overall performance of Meraculous is roughly equal at intermediate concurrancies due to having ~ 2.8 more cores. Achieved bandwidths were similar, but again Edison did demonstrate a little advantage for large message sizes. With Meraculous, at low node counts Edison achieves substantially higher performance, but as the problem scales out to more nodes, the performance is similar.

VII. Future Work

This study is a summary of investigations and findings of a small subset of the NERSC application base. The NERSC exascale readiness program (NESAP) is actively working with over 20 code teams to ensure they have success on the Cori platform. [15] The lessons learned in this study will form a basis of knowledge that will be used by the broader NESAP applications in ensuring high performance for Cori, and more importantly, as an on ramp for the set of next-generation platforms that will be encountered in the near future.

Acknowledgment

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] M. M. Wolf, M. A. Heroux, and E. G. Boman, "Factors impacting performance of multithreaded sparse triangular solve," in *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, ser. VECPAR'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 32–44. [Online]. Available: http://dl.acm.org/citation.cfm?id=1964238.1964246

[2] B. W. Barrett, R. Brightwell, R. Grant, S. D. Hammond, and K. S. Hemmert, "An evaluation of mpi message rate on hybrid-core processors," *International Journal of High Performance Computing Applications*, vol. 28, no. 4, pp. 415–424, 2014. [Online]. Available: http://hpc.sagepub.com/content/28/4/415.abstract

[3] H. Shan, B. Austin, N. Wright, E. Strohmaier, J. Shalf, and K. Yelick, "Accelerating applications at scale using one-sided communication," in *6th Conference on Partitioned Global Address Programming Models*, Santa Barbara, CA, October 10 2012.

[4] "National energy research scientific computing center." [Online]. Available: https://www.nersc.gov

[5] "Cori and edison supercomputing systems." [Online]. Available: https://www.nersc.gov/systems/

[6] "Mvapich: Mpi over infiniband, omni-path, ethernet/iwarp, and roce," version 5.3.1. [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/

[7] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc series network," vol. WP-Aries01-1112. [Online]. Available: http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf

[8] "Sandia mpi micro-benchmark suite (smb)," version 1.0-1. [Online]. Available: http://www.cs.sandia.gov/smb/

[9] "Integrated performance monitoring for high performance computing." [Online]. Available: https://github.com/nerscadmin/IPM

[10] B. Bauer, S. Gottlieb, and T. Hoefler, "Performance modeling and comparative analysis of the MILC Lattice QCD application su3_rmd," in *Proc. CCGRID2012: IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2012.

[11] S. Gottlieb and S. Tamhankar, "Benchmarking MILC with OpenMP and MPI," *Nucl.Phys.Proc.Suppl.*, vol. 94, pp. 841–845, 2001.

[12] "Benchmark distribution & run rules," crossroads and NERSC-9 procurement benchmarks. [Online]. Available: http://www.nersc.gov/research-and-development/apex/apex-benchmarks

[13] [Online]. Available: http://www.berkeleygw.org

[14] E. Georganas, A. Buluç, J. Chapman, S. Hofmeyr, C. Aluru, R. Egan, L. Oliker, D. Rokhsar, and K. Yelick, "Hipmer: an extreme-scale de novo genome assembler," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 14.

[15] "Nesap." [Online]. Available: http://www.nersc.gov/users/computational-systems/cori/nesap/