Early Evaluation of the Cray XC40 Xeon Phi System 'Theta' at Argonne

Scott Parker, Vitali Morozov, Sudheer Chunduri, Kevin Harms, Chris Knight, and Kalyan Kumaran
Argonne National Laboratory, Argonne, IL, USA
{sparker, morozov, chunduri, harms, knightc, kumaran}@alcf.anl.gov

***Abstract*—The Argonne Leadership Computing Facility (ALCF) has recently deployed a nearly 10 PF Cray XC40 system named Theta. Theta is nearly identical in performance capacity to the ALCF's current IBM Blue Gene/Q system Mira and represents the first step in a path from Mira to a much larger next generation Intel-Cray system named Aurora to be deployed in 2018. Theta serves as both a production resource for scientific computation and a platform for transitioning applications to run efficiently on the Intel Xeon Phi architecture and the Dragonfly topology based network.**

This paper presents an initial evaluation of Theta. The Theta system architecture will be described along with the results of benchmarks characterizing the performance at the processor core, memory, and network component levels. In addition, benchmarks are used to evaluate the performance of important runtime components such as MPI and OpenMP. Finally, performance results for the scientific applications are described.

I. Introduction

The Argonne Leadership Computing Facility (ALCF) has recently installed a nearly 10 PF Cray XC40 system named Theta. The installation of Theta marks the start of a transition from ALCF's current production resource Mira [3], a 10 PF peak IBM Blue Gene/Q, to an announced 180 PF Intel-Cray system named Aurora to be delivered in 2018. Aurora is to be based on the upcoming 3rd generation Intel Xeon Phi processor while Theta is based on the recently released 2nd generation Xeon Phi processor code named Knights Landing (KNL) [13]. Therefore, in addition to serving as a production scientific computing platform, Theta is intended to facilitate the transition between Mira and Aurora.

This paper presents an initial evaluation of Theta using multiple benchmarks and important science and engineering applications. Benchmarks are utilized to evaluate the performance of major system subcomponents including processor core, memory, and network along with the power consumption. The performance overhead aspects of the OpenMP and MPI programming models is also evaluated. These benchmark results can be used to establish the baseline performance expectations and to guide the optimization and implementation choices for the applications. The ability of the KNL to achieve high floating point performance is demonstrated using a DGEMM kernel. The capabilities of a KNL memory are evaluated using STREAM and latency benchmarks. The performance differences in the memory between the flat and cache memory modes on KNL is also presented. MPI micro-benchmarks are used to characterize the Cray Aries network performance. Finally, performance and scalability results for several ALCF scientific applications are discussed.

The remainder of the paper is organized as follows, Section II describes the Theta system architecture. Section III discusses the performance of KNL's microarchitecture, memory, computation and communication subsystems using a set of micro-benchmarks and Section IV covers the performance of several key scientific applications.

II. Theta/XC40 Architecture

The Cray XC40 is the latest in the Cray line of supercomputers. The ALCF Theta XC40 system has a nearly 10 PF peak with key hardware specifications shown in Table I.

TABLE I
Theta – Cray XC40 Architecture

Nodes	3,624
Processor core	KNL (64-bit)
Speed	1100 - 1500 MHz
# of cores	64
# of HW threads	4
# of nodes/rack	192
Peak per node	2662 GFlops
L1 cache	32 KB D + 32 KB I
L2 cache (shared)	1 MB
High-bandwidth memory	16 GB
Main Memory	192 GB
NVRAM per node	128 GB SSD
Power efficiency	4688 MF/watt [7]
Interconnect	Cray Aries Dragonfly
Cooling	Liquid cooling

Theta continues the ALCF's architectural direction of highly scalable homogeneous many core systems. The primary components of this architecture are the second generation Intel Xeon Phi Knights Landing many core processor and the Cray Aries interconnect. The full system consists of 20 racks which contain 3,624 compute nodes with an aggregate 231,936 cores, 57 TB of high-bandwidth IPM, and 680 TB of DRAM. Compute nodes are connected into a 3-tier Aries dragonfly network. A single Theta cabinet contains 192 compute nodes and 48 Aries chips. Each Aries chip can connect up to 4 nodes and incorporates 48-port Aries router which can direct communication traffic over electrical or optical links [4]. Additionally, each node is equipped with 128 GB solid state drive (SSD) resulting in total of 453 TB of solid state storage on the system.

Each Theta compute node consists of a KNL processor, 192 GB of DDR4 memory, 128 GB SSD, and a PCI-E connected Aries NIC. The KNL processor is architected to have up to 72 compute cores with multiple versions available containing either 64 or 68 cores as shown in Figure 1. Theta contains the 64 core 7230 KNL variant. On the KNL chip the 64 cores are organized into 32 tiles, with 2 cores per tile, connected

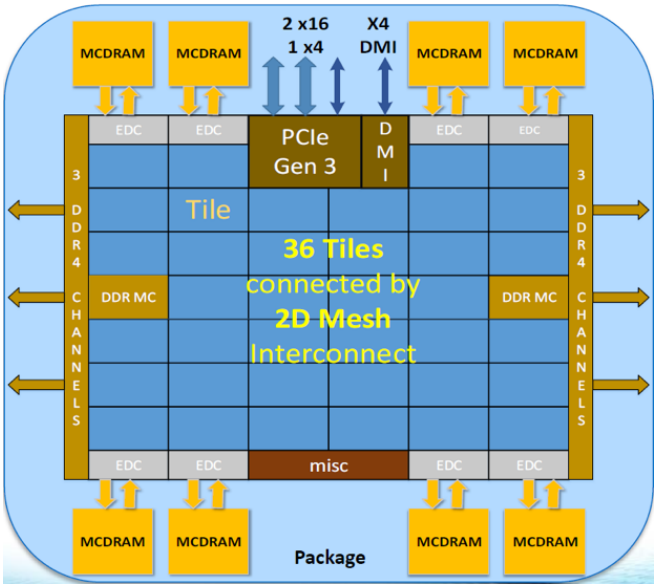

Fig. 1. KNL Processor (Credit Intel)

by a mesh network and with 16 GB of in-package multi-channel DRAM (MCDRAM) memory. The core is based on the 64-bit Silvermont microarchitecture [13] with 6 independent out-of-order pipelines, two of which perform floating point operations. Each floating point unit can execute both scalar and vector floating point instructions including earlier Intel vector extensions in addition to the new AVX-512 instructions. The peak instruction throughput of the KNL architecture is two instructions per clock cycle, and these instructions may be taken from the same hardware thread. Each core has a private 32 KB L1 instruction cache and 32 KB L1 data cache. Other key features include:

1) Simultaneous Multi-Threading (SMT) via four hardware threads
2) Two new independent 512-bit wide floating point units, one unit per floating point pipeline that allow for eight double precision operations per cycle per unit.
3) A new vector instruction extension AVX-512 that leverages 512-bit wide vector registers with arithmetic operations, conflict detection, gather/scatter, and special mathematical operations.
4) Dynamic frequency scaling independently per tile. The fixed clock "reference" frequency is 1.3 GHz on 7230 chips. Each tile may run at a lower "AVX frequency" of 1.1 GHz or a higher "Turbo frequency" of 1.4-1.5 GHz depending on the mix of instructions it executes.

Two cores form a tile and share a 1 MB L2 cache. The tiles are connected by the Network-on-Chip with mesh topology. With the KNL, Intel has introduced on chip in-package high-bandwidth memory (IPM) comprised of 16 GB of DRAM integrated into the same package with the KNL processor. In addition to on-chip memory, two DDR4 memory controllers and 6 DDR4 memory channels are available and allow for up to 384 GB of off-socket memory. Theta contains 192 GB of DDR4 memory per node. The two memories can be configured in multiple ways as shown in Figure 2:

- Cache mode - the IPM memory acts as a large direct-mapped last-level cache for the DDR4 memory
- Flat mode - both IPM and DDR4 memories are directly addressable and appear as two distinct NUMA domains
- Hybrid mode - one half or one quarter of the IPM configured in cache or flat mode with the remaining fraction in the opposite mode

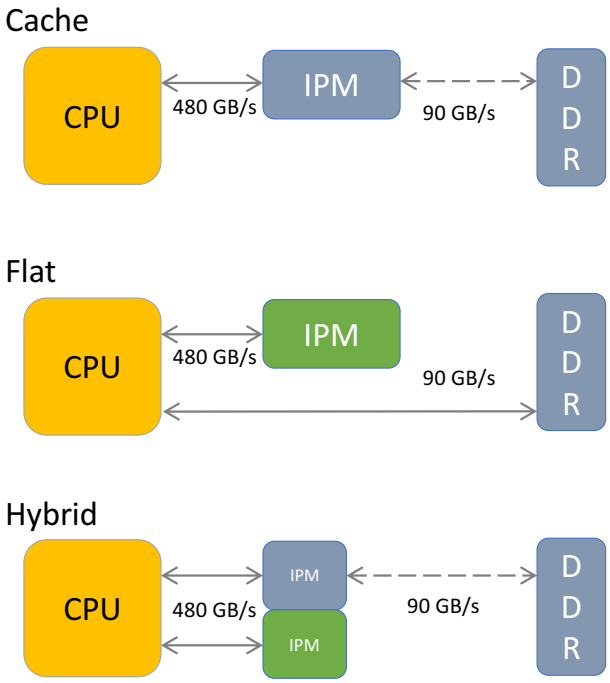

Fig. 2. Memory Modes

This configurable, multi-component memory system presents novel programming options for developers.

III. System Characterization

This section will present the results of micro-benchmarks used to characterize the performance of the significant hardware and software components of Theta. The processor core is evaluated to test the achievable floating point rate. Memory performance is evaluated for both the in-package memory and the off-socket DDR4 memory and the impact of configuring the memory in flat and cache mode is quantified. MPI and network performance is evaluated using MPI benchmarks. OpenMP performance is also evaluated. Finally, the power efficiency of the KNL processor for both floating point and memory transfer operations is determined.

A. Computation Characteristics

As show in Figure 1 the KNL processors on Theta contain 64 cores organized into tiles containing 2 cores and a shared 1 MB L2 cache. Tiles are connected by an on die mesh network which also connect the tiles with the IPM and DDR memory along with off die IO. Each individual core consists of 6 functional units: 2 integer units, 2 vector floating point, and 2 memory units. Despite containing 6 functional units, instruction issue is limited to two instructions per cycle due to instruction fetch

and decode. Instructions may generally issue and execute out-of-order, with the restriction that memory operations issue in order but may complete out of order.

The KNL features dynamic frequency scaling on each tile. The reference frequency is defined based on the TDP of the socket and set at 1.3 GHz on the Theta system. Depending on the mix of instructions, each tile may independently vary this frequency to a lower level, the AVX frequency 1.1 GHz, or to a higher level, the Turbo frequency 1.4 GHz. Based on the "reference" frequency, the theoretical peak of a KNL core is 41.6 GFlops and 2.6 TFlops for a Theta node. However, a stream of AVX-512 instructions cannot be executed at this frequency for a long period of time due to power limitations. Therefore, a more realistic estimate for peak flop rate should account for "AVX frequency", which is 1.1 GHz on Theta, which gives 35.2 GFlops per core, or 2.25 TFlops per node.

We evaluate the performance of Theta's computational capability using *dgemm*, a matrix multiplication kernel. This benchmark achieves over 1.9 TFlops on a Theta node, or 86% of peak performance, for a relatively small matrix size as shown in Figure 3. As seen from `dgemm` benchmark, which is compute intensive, running more than one thread per core does not improve the performance. Utilizing more than one hyper-thread is not beneficial for compute bound kernels since one thread can issues the core limit of two instructions per cycle. While not the case for the *dgemm* kernel, using more than one hyper-thread can in some cases reduce performance due to threads sharing resource such as L1 and L2 caches and instruction re-order buffers.

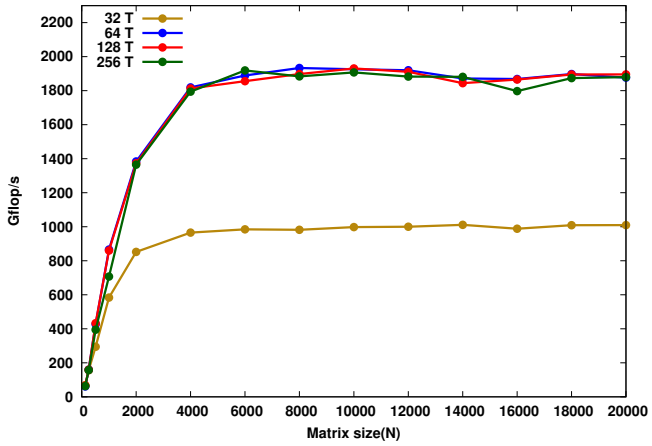

Fig. 3. MKL DGEMM performance (in GFlops)

A number of factors have been observed which limit the performance of the KNL core. While the KNL is capable of two wide instruction issue, it is limited to fetching 16 bytes from the instruction cache per cycle. An AVX-512 vector instruction with a memory operand with non-compressed displacement can be up to 12 bytes long. This may lead to significant degradation of the issue rate, effectively reducing the instruction throughput to only one instruction per clock cycle. Another important consideration is apparent power limits on the throughput of vector instructions for periods of time longer a millisecond. Testing with AVX-512 dense instruction sequences has shown that a KNL core does not continuously execute vector instructions with peak throughput of two instructions per cycle (IPC) and instead throttles throughput to about 1.8 IPC.

It has been observed that system noise from processing OS serviced interrupts can influence the results of fine grained benchmarking. To a degree this may be mitigated by using the Cray's core specialization feature, which localizes the interrupt processing to specific cores. However, while this can reduce run to run variability it may result in a net loss of application performance if one or more cores are exclusively dedicated to handling OS operations. Another source of performance variability has been observed when two cores compete for the shared L2 cache. When running identical workloads one of the cores has been observed to consistently achieve a higher L2 hit rate in some circumstances and therefore achieve better performance. As consequence of this behavior, the two cores running the same copy of a workload may exhibit noticeable performance differences.

The performance of shared memory primitives is crucial for effective thread scaling across the cores on a KNL. The scaling of OpenMP primitives *Barrier*, *Reduction* and *PARALLEL FOR* is shown in Figure 4 as the number of threads scales from 1 to 256 using the EPCC OpenMP benchmarks [5]. The cost of the OpenMP primitives is seen to scale in relation to square root of the thread count as in equation $a * sqrt(T) + b$ where T denotes the number of threads. The dotted lines in Figure 4 show a curve fitted using this equation to the experimentally obtained values plotted using solid lines. The lack of a shared last level cache on KNL may be impacting the cost of OpenMP synchronization as the cost of the OpenMP operations relate to the cost of memory access, with OpenMP operations taking between 130 and 25,000 clock cycles.

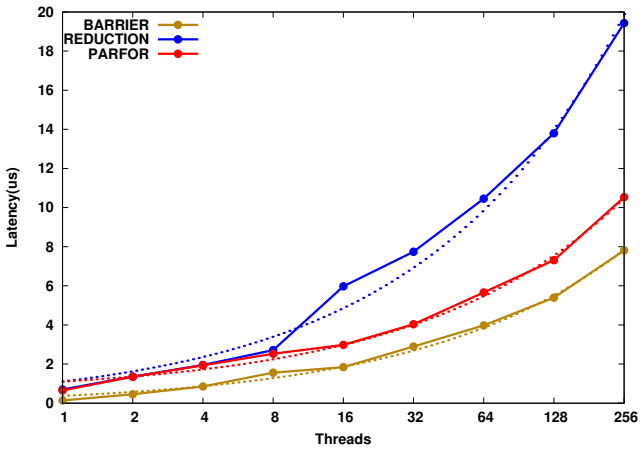

Fig. 4. OpenMP primitives performance scaling and the Curve Fit (dotted lines)

The cost of an OpenMP reduction operation may be compared with that of an MPI Allreduce with all ranks present on the same node. As shown in Figure 5, the cost of the reduction is similar for small and large degrees of on node parallelism however the MPI reduction significantly outperforms the OpenMP version in the middle of the range. Using large numbers of MPI ranks per node is therefore not a disadvantage in terms of the cost of reduction operations.

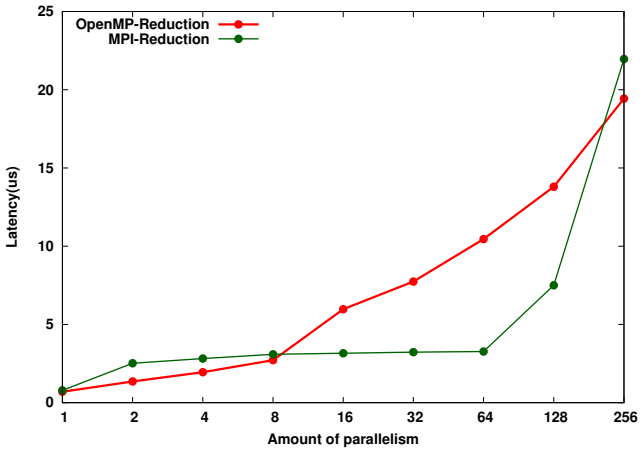

Fig. 5. OpenMP vs. MPI reduction on KNL

B. Memory Subsystem Characteristics

The KNL processor has introduced a novel two component memory architecture consisting of 16 GB of in-package memory (IPM) along with standard off-socket DDR4 memory. As is shown in Figure 1, the IPM memory is arranged in 8 stacks of 2 GB and the chip contains two DDR memory controllers providing 6 channels of DDR4 2133 or 2400 memory. The configuration on Theta provides 192 GB of DDR4 2400 memory on each node in addition to the IPM. This new memory arrangement provides unique benefits and challenges for applications.

This memory subsystem may be configured in one of several ways at boot time as shown in Figure 2. The IPM may be configured as a direct-mapped cache for the DDR, in which case the IPM acts similar to a large L3 cache. This configuration is referred to as "Cache Mode". Alternatively, the IPM and DDR may be configured as seperately accessible NUMA domains, which is referred to as "Flat Mode". Finally, a "Hybrid" mode exists which partitions the IPM into both cache and flat portions, with configuration ratios of 25-75% or 50-50%. In addition, affinity between memory and cores and the distributed cache directory may be configured in a variety of modes including AlltoAll, Quad, and SNC-4 modes. To date the impact of these affinity modes have not been explored in depth, but initial evaluations have not shown significant performance differences. Application performance can, however, be significantly different in Flat and Cache modes.

Memory latency has been measured with a latency benchmark that measures the latency for contention-free accesses to each level of cache hierarchy and the main memory. As seen in Table II, the first-level cache is effectively implemented to have 4-cycle latency between an issue and ready to use. The second-level cache has higher latency at 15 cycles. Both IPM and DRAM memory pools are engineered with DRAM memory and latency is similar at 140 ns for DDR and 161 ns for IPM. However, it is noted that the IPM latency is slightly higher then the DRAM latency, which is attributed to the increased number of channels and associated overhead. Latency-sensitive applications may not therefore readily benefit from placing objects into the IPM.

TABLE II
BEST MEASURED CACHE AND MEMORY LATENCY AND BANDWIDTH.

L1 cache latency	4 cycles	3.1 ns
L2 cache latency	15 cycles	12.2 ns
IPM latency	210 cycles	161 ns
DRAM latency	183 cycles	140 ns
L1 cache bandwidth	44 B/cycle	28.4 GB/s
L2 cache bandwidth	52 B/cycle	30.9 GB/s
IPM flat bandwidth, reg stores	266 B/cycle	346 GB/s
IPM flat bandwidth, stream stores	373 B/cycle	485 GB/s
DRAM flat bandwidth, reg stores	51 B/cycle	66 GB/s
DRAM flat bandwidth, stream stores	68 B/cycle	88 GB/s
IPM cache bandwidth, reg stores	265 B/cycle	344 GB/s
IPM cache bandwidth, stream stores	271 B/cycle	352 GB/s
DRAM cache bandwidth, reg stores	52 B/cycle	67 GB/s
DRAM cache bandwidth, stream stores	45 B/cycle	59 GB/s

Memory bandwidth is characterized using the STREAM triad benchmark [10]. Measurement of memory bandwidth on the KNL is complicated by the two-level nature of the memory architecture. The peak memory bandwidth observed with the STREAM triad benchmark was dependent on the memory mode (flat or cache), the memory being addressed (IPM or DDR), and additionally upon the type of store instruction utilized. The KNL processor implements both conventional cacheable store instructions and the Intel non-temporal streaming store instructions for which stored data is not cached at certain levels of the cache hierarchy. For the KNL streaming stores are not cached in the L1 or L2 caches, but may be cached in IPM when configured as a cache. The potential advantages for streaming stores are that they do not pollute the cache when the stored data is not reused and they avoid a read-for-ownership operation, which fetches cache lines being written to into cache from main memory to insure the entire line is up to date in cache. This operation consumes additional memory bandwidth and limits the bandwidth available for other memory operations.

Table II shows the STREAM triad results for a KNL node on Theta. Flat-mode memory bandwidth results were obtained using a 7.5 GB working set of data from both DDR and IPM using both conventional and non-temporal streaming stores. The peak bandwidth observed is 485 GB/s using IPM and 88 GB/s using DDR. Cache-mode bandwidth was tested using two working set sizes, 7.5 GB/s which is roughly half the IPM cache size, and 120 GB which is much larger than the IPM cache and therefore data comes primarily from the DRAM. Tests were again performed with both conventional and non-temporal streaming stores. In cache mode 352 GB/s was measured for the 7.5 GB working set case, however significant variation was observed across the runs performed on different nodes, with a low value of 225 GB/s observed. For the 120 GB working set case a peak of 67 GB/s was observed. This significantly reduced bandwidth is due to most memory requests for this case being resolved in the DRAM memory as the working set size is much greater than the 16 GB cache size resulting in bandwidth scaled to the DRAM performance. No significant variation across nodes was observed for this case as virtually all memory requests miss in the IPM cache.

Flat mode can be seen to deliver significantly higher band-

width than cache mode. This is a consequence of an additional read from the IPM memory which is required in cache mode in order to check if the address of a line being written is present in the IPM cache. This additional read reduces the available memory bandwidth in cache mode for STREAM triad by approximately 25%. Similar bandwidth rates are observed when conventional stores are used in flat mode, which also necessitates an additional read, in this case a read-for-ownership, which reduces the effective bandwidth by a similar amount. In cache mode there is no advantage to using non-temporal streaming stores and these stores can be observed to be detrimental for working sets larger than the IPM cache as can be seen by the reduction in bandwidth from 67 to 59 GB/s when non-temporal streaming stores are used.

The performance variation of 225-352 GB/s observed in cache mode is attributable to cache misses in the direct mapped IPM cache. These misses occur because cache lines are placed into the cache using physical addresses and the Linux virtual to physical address mapping can become effectively randomly ordered at the page level which allows for cache conflicts on physical addresses even though the virtual address range of 7.5 GB should produce no conflicts. The randomness of the page ordering allows for a varied number of conflicts across different nodes, producing nodes with virtually no conflict, which achieve 352 GB/s, and nodes with higher numbers of conflict which can reduce the bandwidth down to approximately 225 GB/s. Intel and Cray have implemented a kernel module, Zone Sort, that provides sorting of the free memory page list and attempts to reduce the shuffling of the pages in the virtual to physical address mapping and thereby reduce the number of spurious cache conflicts. This sorting, however, does not fully prevent page shuffling as seen by the measured bandwidth ranging from 225 - 352 GB/s which occurs even with the sorting in place.

The above results utilize all of the cores on the KNL. Peak STREAM triad bandwidth from memory obtainable using only a single core is slightly over 14 GB/s, in this case the performance for a single core is limited due to the number of allowable outstanding loads and stores and the latency of memory operations. It therefore requires a little over half of the cores on a KNL node to saturate the 485 GB/s peak bandwidth of the IPM.

The bandwidth numbers reported were obtained using the Cray core specialization feature which relocates most OS activity to the higher numbered cores. For these measurements core specialization was used to move OS activity to the highest numbered core and 63 cores were used to run the STREAM benchmarks. An approximate 10% improvement in IPM bandwidth was achieved in flat mode when core specialization was used.

C. Communication Characteristics

Theta utilizes a Cray Aries interconnect arranged in a dragonfly topology. Four Theta nodes are connected to a single Aries chip via a PCI-E Gen3 interface. The Aries chip contains four NICs for the attached nodes and 40 bi-direction network ports operating in each direction at either 4.7 GB/s for optical links or 5.25 GB/s for electrical links. The performance of the Theta Aries network is evaluated in this section using OSU MPI benchmarks to quantify the achievable latency and bandwidth for both point-to-point and collective communication.

1) Messaging Rate (`MMPS`*):* The `MMPS` benchmark measures the aggregate number of messages sustainable between two nodes on the network. It measures the interconnect messaging rate, which is *the number of messages that can be communicated to and from a node within a unit of time*. This gives a measure of the capability of the underlying implementation, both software and hardware, to process incoming and outgoing messages. In this benchmark, a reference node communicates with a node that is located one hop away on the Dragonfly. Each MPI task on the reference node communicates with the corresponding MPI task on the neighbor node by sending and receiving messages using non-blocking communication.

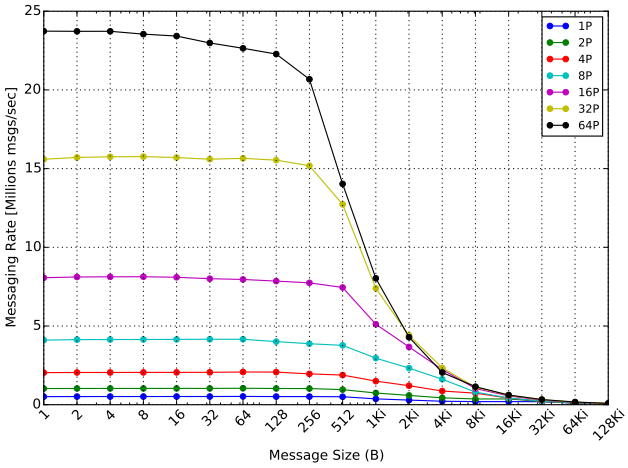

Fig. 6. Messaging Rate (Million messages per sec) on Theta

Figure 6 depicts the messaging rate between 2 KNL nodes on Theta that are 1 hop away. For smaller message sizes the link bandwidth does not significantly influence the measurements and thus the measurements with smaller messages indicate the peak injection rate possible. A single core is unable to achieve peak messaging rate, and the messaging rate increases linearly as we add more MPI ranks per node due to the increased number of communication streams, thereby leading to a higher messaging rate. With 64 processes per node, a maximum messaging rate of 23.7 mmps (million messages per second) for the 1-byte message transfers is achieved.

The peak sustained bandwidth observed using 64 processes per node was around 11.4 GB/s as shown in Figure 7. The maximum bandwidth achievable using one process per node reaches to only around 8 GB/s. As shown, using more MPI processes per node improves the aggregate off-node bandwidth up to around 16 ranks per node. For applications where the communication rate is critical for performance using multiple MPI ranks per node, or having multiple threads calling MPI, could improve performance.

2) Latency: Minimizing point-to-point communication latency is key for improved performance for many applications. Several OSU MPI benchmarks were used to evaluate the achievable bandwidth and latency between the nodes located

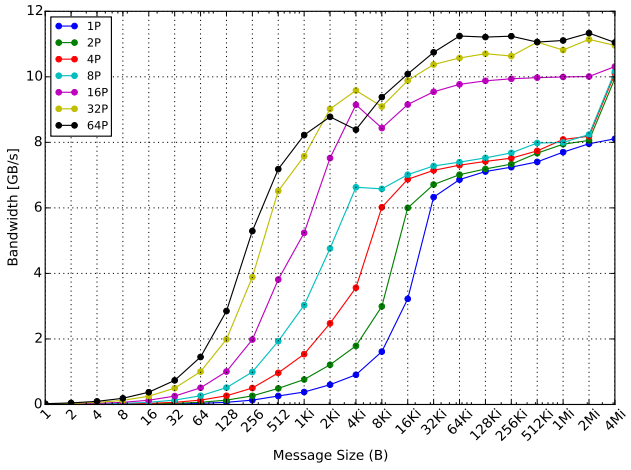

Fig. 7. Messaging Rate (in terms of Bandwidth) on Theta

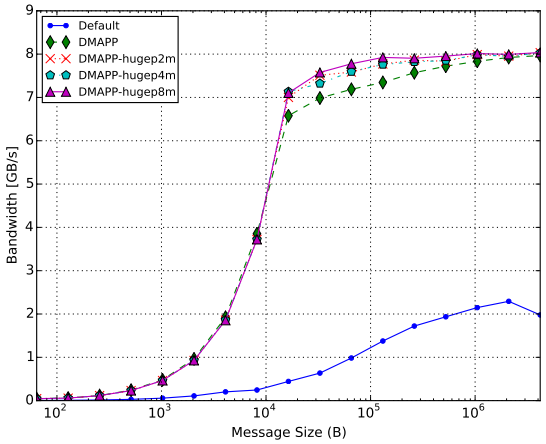

Fig. 9. RMA GET bandwidth on Theta

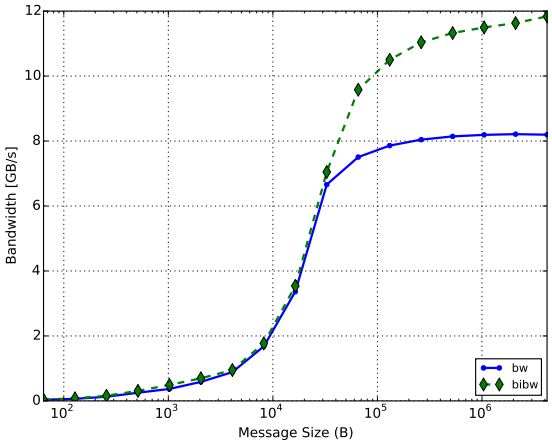

Fig. 8. Point-to-point unidirectional and bidirectional bandwidth on Theta

at one hop away within the dragonfly topology. A node on the Aries dragonfly network is connected to the router over a PCIe interface having a 16 GB/s peak bandwidth, however PCIe overheads limit the achievable bandwidth to significantly lower values. A peak unidirectional bandwidth of close to 8 GB/s and peak bi-directional bandwidth of close to 12 GB/s was observed for communication between the nodes as shown in Figure 8.

The latency for nodes 1 hop away was also measured and is shown in Table III. The latencies for sendrecv were measured using the ping-pong benchmark and one-sided put and get latencies were also measured. The 0-byte sendrecv latency is 3.07 us, whereas the 0-byte one-sided Get/Put has a latency of just 0.61 us. 1-byte message latencies ranged from 3.22 to 4.7 us. The significant increase in one sided latency from 0 to 1 byte messages is possibly related to zero byte messages not requiring any off node operations.

The Cray MPI implementation allows the use of one of two underlying communication layers. The default layer is Nemesis driver for Aries layered over uGNI [12]. Alternatively the Distributed Shared Memory Application API [9] (DMAPP) may be used for improved one-sided operations performance. Figure 9 shows the peak RMA Get bandwidth measured using the default configuration is around 2 GB/s. Using DMAPP, communication performance is significantly improved with a peak bandwidth of 8 GB/s for RMA Get. Similar improvements may be seen in Figure 10 for RMA Put bandwidth with a peak bi-directional bandwidth of around 11.6 GB/s. Using Huge pages was found to improve the bandwidth further, achieving the peak bandwidth even with moderate sized messages. Using huge pages helps avoid TLB thrashing on the Aries router and thus helps in achieving higher bandwidth.

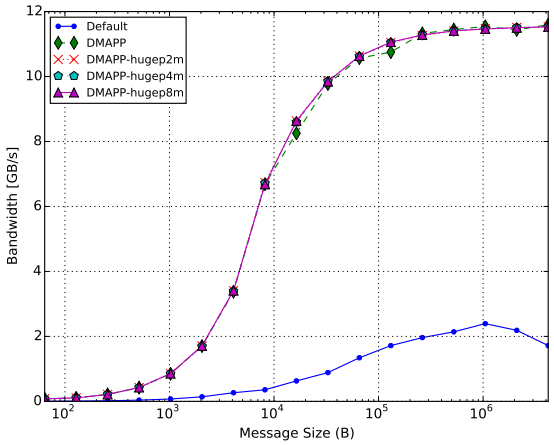

Fig. 10. RMA PUT bidirectional bandwidth on Theta

3) Performance of Collective Communication: The performance of three common MPI collective operations, *Broadcast*,

TABLE III
MPI MESSAGE LATENCY IN US

Benchmark	Zero Bytes message	One Byte message
Ping Pong	3.07	3.22
Put	0.61	2.90
Get	0.61	4.70

Gather, and *Allreduce*, which have distinct data flow dependencies [8] was evaluated.

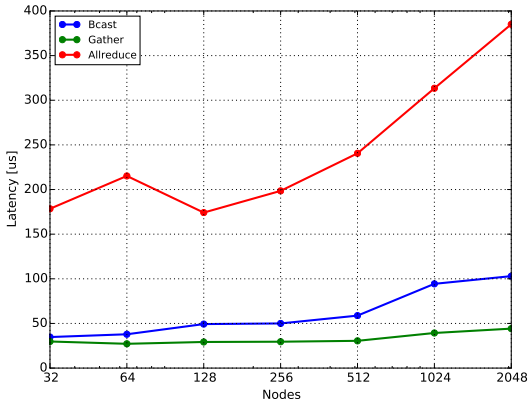

Fig. 11. Performance comparison of MPI Collectives (Broadcast, Gather and AllReduce) with message size 8KB.

Figure 11 depicts the latency (in microseconds) of collective operations for a message size of 8KB with increasing numbers of nodes, using only one MPI rank per node. As expected latencies increase with node count, though Allreduce shows a spike at 64 nodes and has the overall steepest latency increase. Gather is unexpectedly faster compared to Broadcast and Allreduce is significantly slower than either Gather or Bcast, which may be due to the additional cost of the floating point operations required for a reduction. The reason for the Allreduce spike at 64 nodes is unclear, but it should be noted that considerable variation in MPI performance was observed between measurements due to inter-job contention on the dragonfly topology.

In the dragonfly topology most network links may be utilized at any time by multiple applications running on the system resulting in variable amounts of bandwidth available on a given link for a given application. This variation adds complexity to the benchmarking process and requires multiple runs to minimize it's impact on the reported results. This network performance variability can also cause considerable variation in the performance of applications.

D. Power Characteristics

The Intel Xeon Phi CPU occupies five of the top 10 positions on the November 2016 Green500 list. Theta, specifically, is number 7 on this list with a reported MFLOPS/W of 4688.0. In this section the power efficiency of the KNL based compute node on Theta is examined using the power monitoring infrastructure provided by Cray. Cray provides two methods to access power data. The first method is via Cray Hardware Supervisory System (HSS) and the second is using Intel's RAPL. The power characteristics of the compute-intensive *dgemm* benchmark and the memory-intensive STREAM benchmark were examined using Cray's provided mechanism to measure the node's power and access the data using PAPI.

Dgemm: Table IV depicts the power characteristics for a single node of the *dgemm* benchmark as the number of OpenMP threads per MPI rank are varied. A total of 64 MPI ranks on the node was used, with one MPI rank per core and either 1, 2 or 4 threads per core. The node was configured for flat quadrant mode with the application being bound to IPM. The Intel MKL DGEMM library was executed with a matrix size of 1024×1024, which is large enough to exceed the L2 cache size, for 1000 iterations. The KNL does not offer hardware to count the exact number of floating point operations, so the number of FLOPs has been estimated as $2 \times n^3$ FLOPS. Table IV shows that the best performance is achieved with just a single thread per core, which has the best overall time-to-solution as well as lowest power.

TABLE IV
NODE POWER CHARACTERISTICS OF THE *dgemm* BENCHMARK FOR 1024×1024 MATRIX AND A NUMBER OF THREADS PER CORE (TPC).

TPC	Time sec.	Power Watts	Energy kJ	Flop Rate GFLOP/s	Power Eff. GF/Watt
1	110.0	284.6	31.3	1249.5	4.39
2	118.6	285.4	33.9	1158.7	4.06
4	140.3	295.0	41.4	979.3	3.32

STREAM: Table V contains the power consumption and efficiency for STREAM running on a single node in flat-quadrant mode. The IPM and DDR are tested separately. The STREAM benchmark is run with a single MPI rank using 64 threads with 1 thread per core. The "idle" power consumption of a node is measured by a benchmark executing a sleep loop which measures 48.4 W. For both the IPM and DDR4 tests, a total of 15 GB of memory was used and each test case was run for 100 iterations. It can be seen that IPM provides a considerable gain of 4.3x in the memory bandwidth power efficiency and an 1.2x increase in overall power consumption.

TABLE V
POWER CONSUMPTION OF THE STREAM BENCHMARK FOR A NODE. THE PER NODE BANDWIDTH IS MEASURED BY RUNNING ONE THREAD ON EACH CORE.

Stream	IPM			DDR4		
	GB/s	Power Watts	Efficiency GB/s/Watt	GB/s	Power Watts	Efficiency GB/s/Watt
Copy	420.4	265.8	1.58	82.9	216.1	0.38
Scale	426.6	267.2	1.60	82.9	220.1	0.38
Add	471.7	281.7	1.67	87.3	223.4	0.39
Triad	449.5	270.5	1.66	87.1	224.4	0.39

IV. Application Studies

This section will examine the performance of three applications, LAMMPS, MILC, and Nekbone on Theta. These three applications represent a key cross-section of core hours used on ALCF systems and cover a range of algorithmic characteristics (See Table VI) which are common to many other applications which will run on Theta.

TABLE VI
ARGONNE APPLICATIONS STRESS KEY ALGORITHMS

App	Struct. Grid	Unstr. Grid	Dense LA	Sparse LA	FFT	Particle N-Body	Monte Carlo
NEKbone	√	√	√				
LAMMPS	√				√	√	
MILC	√		√	√			√

A. LAMMPS

LAMMPS [11] is an open-source molecular dynamics code that supports many potentials and models to efficiently simulate particles in a myriad of systems (e.g. liquids, biomolecules, materials, and mesoscopic systems). LAMMPS is modular code written in portable C++ and supports parallelization with MPI, OpenMP, and Kokkos and supports running on both CPUs and GPUs. LAMMPS is principally parallelized using spatial decomposition schemes that partition a system into sub-volumes assigned to unique MPI ranks. Within each sub-volume, multi-threaded parallelization (e.g. OpenMP and Kokkos) can be used to complete operations common to classical molecular simulations: build Verlet neighbor lists, evaluate interparticle interactions, integrate Newton's equations of motion, etc... Modified versions of the LAMMPS rhodo benchmark, simulating the rhodopsin protein within a solvated lipid bilayer, was used in the following performance analyses. For all single-node and strong-scaling analyses, simulations sampling constant volume conditions were used along with the quad cluster mode. Intel compiler version 2017.2.174 and LAMMPS svn version 12990 from January 26, 2015 was used in this work. Additional performance improvements are available in LAMMPS for Intel hardware with installation of the USER-INTEL package to accelerate energy and neighbor list calculations.

Strong scaling results for a system with 32 million particles are shown in Figure 12 for runs up to capability scale on both Mira and Theta (cache-quad mode). In all cases, each core maps to a single MPI rank and multiple OpenMP threads are used: 2 and 4 OpenMP threads per MPI rank on Theta and Mira, respectively. For these strong scaling results, an important modification to the rhodo benchmark is evaluation of electrostatic interactions using a pairwise damped Coulomb potential (via pair_style "lj/charmm/coul/charmm") to mimic large-scale coarse-grained simulations as opposed to using the standard particle-particle particle-mesh (PPPM) algorithm based on 3D Fast-Fourier Transforms (FFT). On Mira, 16 nodes is used for baseline as there is insufficient memory available to run one MPI rank per core on fewer BG/Q nodes. The parallel efficiency of LAMMPS on 3,072 nodes of Theta is 50% with most of the scaling loss coming from nearest neighbor communication for particle exchanges. The pairwise computation and rebuilding of neighborlists both scale linearly for this system up to 3072 nodes. On a per-core basis, LAMMPS is 1.4x faster on Theta than Mira and 5.2x faster on a per-node basis (comparing red-square and black-circle curves in Figure 12). Several performance critical kernels in LAMMPS have been ported to Intel hardware, including KNL with AVX-512 instructions, and strong-scaling results for a similar model are shown in Figure 12 as the blue-triangle curve. These Intel-specific optimizations, which can not be executed on the BG/Q, yield an additional 2.2x speedup on Theta, thus bringing the total node-to-node speedup to 10.8x compared to a BG/Q node.

The effect of memory mode (cache vs. flat) was examined in single-node runs with LAMMPS for a system of 256,000 particles. With pairwise electrostatics evaluated again via pair_style "lj/charmm/coul/charmm", the run-times were

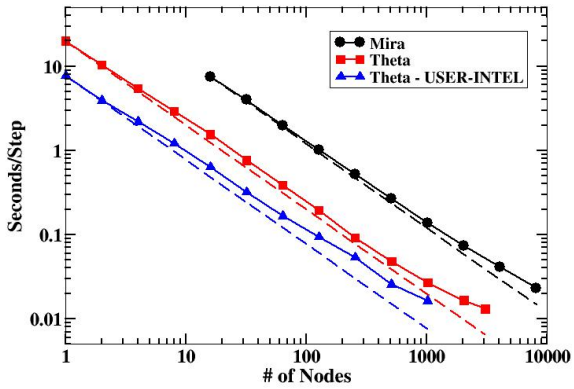

Fig. 12. Strong scaling of LAMMPS on BG/Q and KNL. In all cases, each core maps to a single MPI rank and multiple OpenMP threads are used as described in the text.

observed to be nearly identical in cache and flat modes, and likewise were similar when allocating memory explicitly from IPM or DRAM, with run-times being within 1% of one another. Measurements with Intel Vtune Amplifier profiler indicate that the maximum peaked memory bandwidth is ∼30 GB/s from predominantly the pairwise computation and neighborlist functions. With the USER-INTEL optimizations, the peaked memory bandwidth observed was improved to ∼60 GB/s. Since the KNL DRAM has been observed to deliver up to 88 GB/s (Table II), it can provide sufficient memory bandwidth for this LAMMPS simulation. However, in similar single-node runs where electrostatic interactions are now evaluated using the PPPM method, which involves 3D Fast Fourier Transforms, differences in run-times were observed with flat-dram mode running slowest with (∼27%) difference in run-times. In measurements with Vtune, a sustained peaked memory bandwidth of ∼80 GB/s was observed primarily coming from the FFT and particle-mesh operations in the PPPM algorithm. When run in cache mode or allocating explicitly to IPM in flat mode, the maximum peaked memory bandwidth observed was ∼170 GB/s. Similar memory bandwidth profiles for PPPM functions were observed with the Intel-optimized implementation. While performance measurements may be improved with more recent versions of LAMMPS and its USER-INTEL add-on package, the key observation here is that the IPM improves performance for those kernels limited by memory bandwidth. It is expected that, moving forward, optimal performance for large-scale LAMMPS simulations that don't fit entirely within IPM would be obtained by allocating most of the data structures to DRAM and explicitly allocating memory bandwidth critical data structures (e.g. 3D FFTs for PPPM) to IPM.

B. MILC

MILC [2] is the MIMD Lattice Computation code designed to address the fundamental understanding of high energy and nuclear physics, such as the study of the mass spectrum of strongly interacting particles, and the weak interactions of these particles. MILC simulates the fundamental theory of the strong nuclear force, Quantum Chromodynamics (QCD),

formulated on a four-dimensional space-time lattice. The MILC codebase contains many components, but for this study the *su3_rhmd_hisq* application is used, which implements the Rational Hybrid Monte Carlo algorithm. MILC is implemented in C and, for this analysis, was run using MPI only with 64 ranks per node. The version of MILC used did not contain the QPhiX optimizations.

Figure 13 shows the data from two weak scaling studies performed a few days apart. For these runs the input data set uses 24^4 lattice sites per node which requires a small amount of memory per core that easily fits within the IPM on the node. However, despite fitting entirely within the IPM the runs were performed in cache memory mode, along with the quadrant NUMA mode, because the majority of nodes on Theta were only available in this configuration. In Figure 13, the Y-axis units of MFLOP/s was computed by extracting the timing data for several kernels and using known operation counts determined by the MILC developers. Ideal weak scaling performance should result in an identical number of MFLOPS for each node count. Instead we see significant variance in performance. This is a consequence of the general state of the machine at the time the jobs were run. The first set of benchmark runs were done during a time when Theta was highly utilized, labelled 'pre-reboot'. The second set of runs were done immediately after a maintenance period in which the entire machine was rebooted. Two of the primary causes of variability in performance for the MILC code are the variability present in cache mode, described in Section III-B, and contention on the dragonfly network, described in Section III-C. The 'post-reboot' data shows an overall improved performance due to better overall performance of cache mode after a reboot and the reduced number of running jobs causing contention on the network.

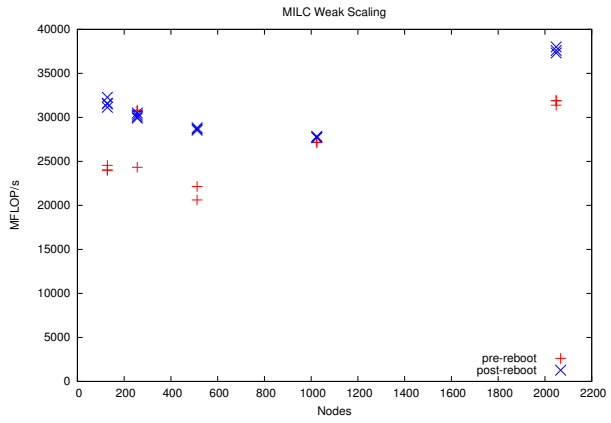

Fig. 13. MILC Weak Scaling

C. Nekbone

Nekbone [1] is a mini-app derived from the Nek5000 [6] CFD code which is a high order, incompressible Navier-Stokes CFD solver based on the spectral element method. It exposes the principal computational kernels of Nek5000 to reveal the essential elements of the algorithmic-architectural coupling that are pertinent to Nek5000. Nekbone solves a standard Poisson equation in a 3D box domain with a block spatial domain decomposition among MPI ranks. The volume within a rank is then partitioned into high-order quadrilateral spectral elements. The solution phase consists of conjugate gradient iterations that invoke the main computational kernel which performs operations in an element-by-element fashion. Overall, each iteration consists of invoking routines performing vector operations, matrix-matrix multiply operations, nearest-neighbor communication, and MPI_Allreduce operations. The code is written in Fortran and C, where C routines are used for the nearest neighbor communication and the rest of the routines are in Fortran. It uses hybrid parallelism implemented with MPI and OpenMP. Nekbone is highly scalable and can accommodate a wide range of problem sizes, specified by setting the number of spectral elements and the number of grid points within the elements.

Nekbone may be run using MPI ranks, OpenMP threads, or a combination of the two. OpenMP threading in Nekbone is coarse grained with only one parallel region spanning the entire solver. Compute load is distributed across threads in the same manner that it is done across ranks. In every iteration, a fixed set of elements to be updated are assigned to threads or ranks. Thus, the compute load and the amount of synchronization performed by OpenMP threads is nearly identical to that of MPI ranks. The number of elements per rank or thread may be load balanced by ensuring that the configured run contains a number of elements perfectly divisible by the number of ranks and threads. Table VII shows the Nekbone solver time for the same problem configuration run with varying numbers of ranks, threads, and hyper-threads on a Theta node. As expected, given the similar implementation of ranks and threads, there is no significant difference in performance across the ranks vs thread mix, except at high rank counts when multiple hyper-threads are used. The use of more than one hyper-thread can be seen to be detrimental to performance, which can be understood due to the nature and performance by the various Nekbone kernels which are discussed further below.

TABLE VII
NEKBONE THREADS AND RANKS

	1 H.T.		2 H.T.		4 H.T.	
Ranks	**thds**	**time**	**thds**	**time**	**thds**	**time**
1	64	3.07	128	3.65	256	4.72
2	32	3.00	64	3.66	128	4.70
4	16	3.02	32	3.61	64	4.57
8	8	3.02	16	3.63	32	4.59
16	4	3.05	8	3.67	16	4.58
32	2	3.04	4	3.66	8	4.61
64	1	3.14	2	3.75	4	4.69
128			1	4.10	2	5.08
256					1	14.9

To compare the performance of the Theta KNL processor to that a of common Haswell configuration (E5-2699v3, dual socket, 36 cores), Nekbone was configured for a single node run using only OpenMP and using all of the cores on the processor. For the same problem run on both nodes, the solve time per element was 0.38 ms on KNL and 1.22 ms on Haswell, showing a 3.2 times improvement in performance on KNL over Haswell. This speedup is reasonable considering the 1.7x improvement in peak FLOP rate and the 4x improvement in

memory bandwidth, when using IPM, on the KNL over the Haswell node.

The scaling of Nekbone on Theta has been evaluated in both weak and strong scaling contexts. Weak scaling results are shown in Figure 14. This figure shows the weak scaling performance for a number of problem sizes, all with 16 grid points per element and between 128 and 16,384 spectral elements per node. The runs were performed using 2 MPI ranks per node and 32 threads per rank with the node in flat mode and using the libxsmm library for small matrix multiplies. In general, the problems with more elements per node are seen to scale better, which is expected given the higher compute-communication ratio for these problems. However, even for the largest problem size considered, the parallel efficiency decreases appreciably as Nekbone is scaled to larger nodes on Theta. Nekbone, itself is highly scalable, as shown in Figure 15, which shows near perfect weak scaling on the ALCF BG/Q system Mira up to 48k nodes when using 512 elements per node. Loss of parallel efficiency with weak scaling on Theta is attributable to either increased cost for point-to-point communication due to network contention or the increased cost of MPI_Allreduce operations as the rank count increases, since the workload per node remains otherwise same. Future work will examine the use of explicit rank mapping across the dragonfly network to optimize rank placement and minimize point-to-point communication contention.

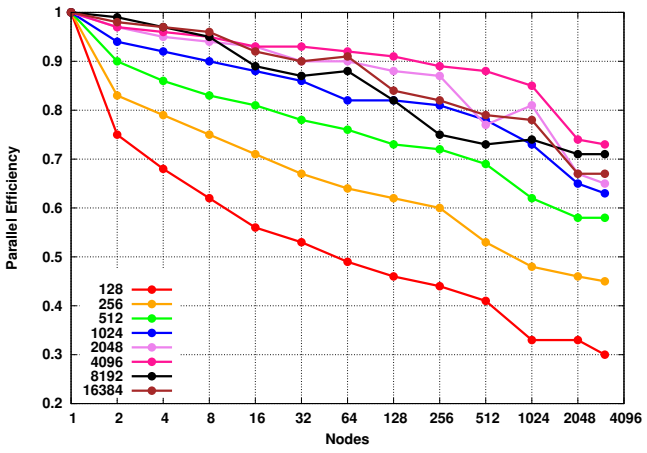

Fig. 14. Nekbone Theta weak scaling

Strong scaling results for Nekbone on Theta are shown in Figure 16 for a range of different problems sizes containing between 131,072 and 1,048,576 total elements. These runs were again performed using 2 MPI ranks per node, 32 threads per rank with 16 grid points per element on nodes using flat memory mode. The libxsmm optimized version is used. The scaling results can be observed to show a similar drop in parallel efficiency as the node count increases. The curves for the larger problems are shifted to the right, with given levels of parallel efficiency hit at higher node counts. While strong scaling parallel efficiency is expected to drop as the node count is increased due to the diminishing computational work available per node, the strong scaling for Nekbone on Theta can likely be improved by optimizing the communication using the rank re-ordering.

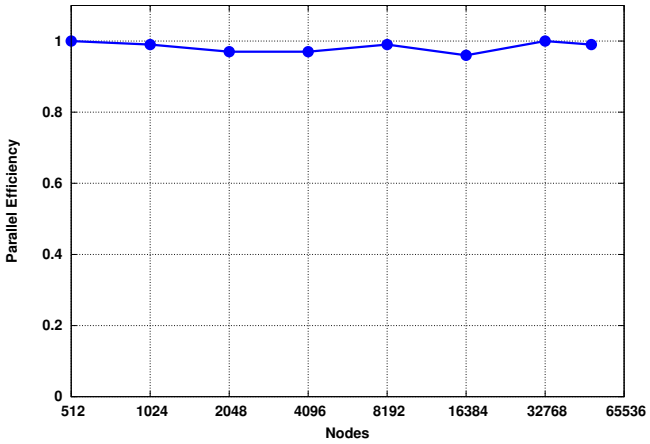

Fig. 15. Nekbone BG/Q weak scaling

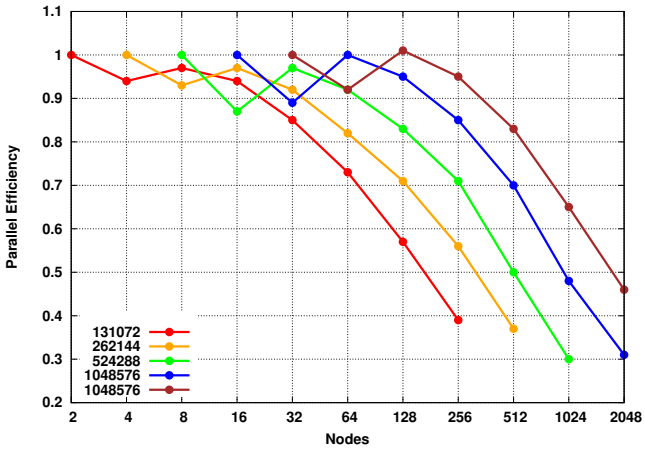

Fig. 16. Nekbone Theta strong scaling

The routines in Nekbone may be grouped into three primary types: streaming kernels which are memory bandwidth bound, compute kernels which are primarily performing small matrix-matrix multiply operations, and communication kernels which perform point-to-point communication operations and reductions. At node counts where Nekbone is performing with 80% parallel efficiency, the time for the streaming kernels accounts for approximately 50% of the overall runtime, with matrix-matrix multiplications accounting for approximately 20% of the time, and communication related operations accounting for approximately 30% of the time. The streaming kernels were observed to achieve between 70% and 95% of peak flat mode memory bandwidth, and the matrix multiply operations achieved approximately 40% of peak FLOP rate when libxsmm is used.

V. Conclusion

This study provides an initial evaluation of the Cray XC40 system, Theta, including performance characterization of the system components such as KNL micro-architecture, memory subsystem and the dragonfly interconnect. Three applications,

LAMMPS, MILC and Nekbone were evaluated in the context of these component characteristics.

The peak floating point performance of the KNL core and node are 35.2 GFlops and 2.25 TFlops respectively. An optimized DGEMM benchmark was found to achieve around 86% of the peak performance. While the KNL core has a theoretical peak throughout of 2 IPC (instructions per clock cycle) actual throughput can be limited by factors such as instruction width (opcode length) and power constraints. Power measurements show better computational efficiency when using fewer hyperthreads. OS noise and the shared L2 cache contention have been identified as the sources of core to core variability on the node. Cray core specialization helps address the OS noise which could have significant impact on the timing of microkernels. The overhead of OpenMP constructs such as Barrier and Reduce was quantified and found to be related to the latency of main memory access due to the lack of a shared last level cache. A simple performance model was developed to quantify the overhead of OpenMP pragmas which scale as the square root of the thread count.

The peak observed memory bandwidth of the DDR and IPM memory was measured using the STREAM Triad benchmark. Considerable variation was found in memory bandwidth between the flat and cache memory mode configurations. Use of Streaming stores in place of the regular stores improves the bandwidth in flat mode but can be detrimental in cache mode. It was observed that IPM cache mode page conflicts are not fully prevented by the Zone sort algorithm for free page management which results in considerable node to node variability in cache mode IPM bandwidth. Power measurements show better efficiency when using IPM versus DDR memory.

The OSU MPI benchmarks were used to characterize the performance of the Aries network and the Cray MPI implementation. A peak messaging rate of 23.7 million messages per sec and a corresponding 11.4 GB/s peak off node bandwidth were observed. The latency for 1 byte message transfers between nearest nodes was found to in the range of 3-5 us. RMA performance was found to increase significantly when the DMAPP interface is used.

The performance optimization and scalability of three applications, LAMMPS, MILC, and Nekbone are shown with respect to the performance characteristics identified at the component level. In all cases, the applications were successfully able to scale up to large fractions of Theta and direct comparisons made to performance on ALCF's current production resource Mira. For a 32 million particle simulation, LAMMPS continued to show performance improvements up to 2048 nodes running 64 MPI ranks per node and memory bandwidth measurements motivate exploration of explicitly allocating specific data structures to IPM and DRAM. In weak-scaling data for the MILC application, again running 64 MPI ranks per node, effects on performance due to both cache-mode and network variability were discussed. Strong-scaling performance data for the Nekbone mini-app up to the full Theta machine was discussed, as well as single-node performance which showed no significant difference in runtime as MPI ranks and OpenMP threads were varied. A common theme in the three application studies was consideration of the variability and contention in the interconnect and their effects on point-to-point and collective communication. Looking forward, Theta will serve as an invaluable resource for preparing science and engineering applications for the upcoming 180 PF Intel-Cray Aurora system.

ACKNOWLEDGMENT

This research has been funded in part and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

The authors would like to thank the ALCF application and operations staff for their assistance. They gratefully acknowledge the help provided by the application teams whose codes are used herein.

REFERENCES

[1] CORAL Benchmarks. https://asc.llnl.gov/CORAL-benchmarks/.

[2] The MIMD Lattice Computation (MILC) Collaboration. http://www.physics.utah.edu/detar/milc/.

[3] Argonne National Laboratory Selects IBM Supercomputer to Advance Research. http://www03.ibm.com/press/us/en/pressrelease/33586.wss, Feb. 2011.

[4] Bob Alverson, Edwin Froese, Larry Kaplan, and Duncan Roweth. Cray XC series network. *Cray Inc., White Paper WP-Aries 01-1112*, 2012.

[5] J. Mark Bull. Measuring synchronisation and scheduling overheads in openmp. In *Proceedings of First European Workshop on OpenMP*, pages 99–105, 1999.

[6] P. Fischer, J. Lottes, D. Pointer, and A. Siegel. Petascale algorithms for reactor hydrodynamics. *Journal of Physics: Conference Series*, 2008.

[7] Green 500. Green 500 List. http://green500.org.

[8] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[9] Cray Inc. XC Series GNI and DMAPP API User Guide . *(CLE6.0.UP01)*, 2017.

[10] J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, 1991-2007.

[11] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comp. Phys.*, 117:1–19, 1995.

[12] Howard Pritchard, Igor Gorodetsky, and Darius Buntinas. A ugni-based mpich2 nemesis network module for the cray xe. In *Proceedings of the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface*, EuroMPI'11, pages 110–119, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] Avinash Sodani. Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor. In *Hot Chips 27 Symposium (HCS), 2015 IEEE*, pages 1–24. IEEE, 2015.