Analysis of Cray XC30 Performance using Trinity-NERSC-8 benchmarks and comparison with Cray XE6 and IBM BG/Q.

M.J. Cordery B. Austin
H.J. Wassermann C.S. Daley
N.J. Wright
NERSC, Lawrence Berkeley National Laboratory,
Berkeley, CA
mjcordery,baustin,hjwasserman,csdaley,njwright@lbl.gov

S.D. Hammond D. Doerfler
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM
sdhammo,dwdoerf@ sandia.gov

Abstract

In this paper, we examine the performance of a suite of applications on three different architectures: Edison, a Cray XC30 with Intel Ivy Bridge processors; Hopper and Cielo, both Cray XE6's with AMD Magny–Cours processors; and Mira, an IBM BlueGene/Q with PowerPC A2 processors. The applications chosen are a subset of the applications used in a joint procurement effort between Lawrence Berkeley National Lab, Los Alamos National Laboratory and Sandia National Laboratories. Strong scaling results are presented, using both MPI-only and MPI+OpenMP execution models.

Keywords benchmarking, HPC, Performance

1. Introduction

The classic parallel programming model, MPI, faces several new challenges on petaflop computing platforms, which are dominated by multicore-per-node architectures [6, 14]. These challenges include a reduced memory capacity per core, reduced memory and network bandwidth per core, and the inefficiency of using two-sided messages to handle a large amount of fine-grain communication. These challenges will only be exacerbated as the field of high performance computing moves forward into the exa-scale era wherein application developers will no longer be able to achieve significant performance and scalability gains with an MPI-only programming model. As on-node parallelism increases, effective use of future technologies will require exposing more fine-grained data parallelism, better management of data placement and data movement, exploiting longer vector units, and exploring task-based parallelism and communications reducing algorithms. To this end, several laboratories within the Department of Energy (DOE) are collaborating on the FastForward project to research both new technologies and new execution models. While this program advances, DOE laboratories are working with their scientists and code development teams to address these issues.

This collaborative effort also extends to the realm of system procurement where, in a debut effort, Lawrence Berkeley Laboratory (LBL), Los Alamos National Laboratory (LANL) and Sandia National Laboratories (SNL) (the later two comprising ACES, the Alliance for Computing at Extreme Scales), have entered a partnership to jointly procure two systems. While one of the goals of this partnership is to drive favorable economies of scale, a substantial goal is the opportunity for various DOE labs to better understand each other's system requirements and workload characteristics. It is the desire that this understanding will yield future architectures that cover the broadest range of scientific computing needs and are not defined by and targeted at any specific workload. As part of this procurement, each of the involved laboratories has contributed a selection of benchmark codes that represent an important part of their workload. The primary aim of this paper is to evaluate the performance characteristics of this new suite of benchmarks on state-of-the art platforms, especially at high concurrencies. Furthermore we are interested in how well different execution models perform on different architectures, especially the comparison between the classical MPI-only execution model and one using many relatively lightweight threads. To this end, we present results showing how each benchmark strong scales on three different architectures: Edison, a state-of-the art Cray XC30 at NERSC; Hopper, a Cray XE6 (also at NERSC); and Mira and Vulcan, both IBM Blue Gene/Q machines at Argonne National Laboratory and Lawrence Livermore National Laboratory, respectively. We compare and contrast the performance of the selected benchmarks on each machine when using an MPI-only execution model and, at the other extreme, how each scales when using the maximum number of OpenMP threads possible on a node (or, in the case of Hopper and Edison, the maximum number of threads possible in a NUMA domain). Short of an exhaustive study, this will give us some sense of the range of performance possible for intermediate mixes of MPI tasks and OpenMP threads. It is also of interest to us how this new suite of benchmarks impacts previous metrics of system performance, in this case NERSC's Sustained System Performance (SSP) metric, and why.

In summary, the principle contributions of this paper are

- On a node-per-node basis, the Cray XC-30 offers a significant performance advantage over both the Cray XE6 and IBM's BlueGene/Q, by 1.8-3.8x and 1.8-9.4x respectively, over a range of node counts. Based on a metric of performance per watt, however, the Cray XE6 and the BlueGene/Q are more equivalent.
- For the benchmarks used in this paper, over the range of nodes considered, hybrid MPI-OpenMP applications currently run slower than MPI applications across all platforms. The principle reason for this appears to be that the OpenMP implementations of the applications are not as efficient as the MPI ones at expressing parallelism.
- The benchmarks used, which represent the workloads at leading DOE supercomputing centers, have of low computational intensity and their performance is primarily limited by memory bandwidth.

The paper is organized as follows: Section 2 describes the experimental platforms. Section 3 presents a description of the benchmark applications used as well as their general strategies for both MPI and OpenMP parallelism. Performance results of the benchmark applications and microbenchmarks are presented in Section 4. Related work is presented in Section 5. Finally, we summarize our conclusions and future work in Section 6.

2. Test Platform Descriptions

2.1 BlueGene/Q: Mira and Vulcan

BlueGene/Q is the third revision to IBM's high-performance BlueGene architecture. Each BG/Q node uses a core based upon an embedded PowerPC technology with a 256-bit SIMD vector processing unit clocked at 1.6GHz. Each core is dual-issue, 4-way multithreaded, and has a 16KB L1 data cache. In order to run at the dual-issue rate, at least two threads must be running per core. Each BG/Q processor chip contains 18 cores (with 16 being available to the user, one to handle OS tasks, and a spare core to increase chip yields) connected with a crossbar to a 32MB L2 and the network interface. The two memory controllers per chip can provide a sustained bandwidth of up to 28 GB/s to 16GB of DRAM. Nodes are connected in a high-bandwidth 5D torus. In this paper, we use both the Mira machine located at Argonne National Laboratory (49,152 nodes) and Vulcan, an open-science relative of Sequoia, installed at the Lawrence Livermore National Laboratory. Although the machines vary in size the operating system and compiler implementation are identical and so we treat them as equivalent for the purposes of benchmarking the BlueGene/Q architecture.

2.2 Cray XE6: Hopper

Hopper is a production Cray XE6 machine deployed at NERSC. The XE6 is based on commodity AMD processors connected via HyperTransport to a custom inter-connect. Each processor includes six 2.1GHz AMD Opteron cores. Each core has a 128-bit SIMD (SSE3) floating-point unit, and 64KB L1 and 512KB L2 caches. Cores are connected to a 6MB L3 cache (1MB reserved as a probe filter) and two DDR3-1333 memory controllers. There are four processor chips per node. The interconnect is a Cray custom-designed "Gemini" architecture. Each Gemini chip is connected to two nodes, and the Gemini chips are connected together in a 3d-torus with dimensions 17x8x24.

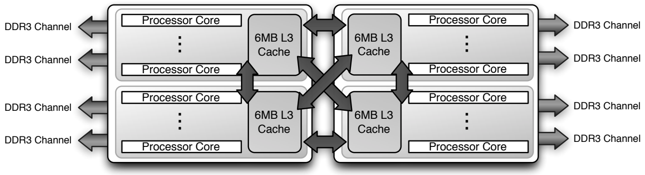

Figure 1: Node Architecture of Hopper

2.3 Cray XC30: Edison

Edison is a Cray XC30 (Cascade) supercomputer recently installed at NERSC. The XC30 architecture is based on commodity Intel processors connected via PCI-Express to a custom interconnect. Each processor is a 12-core, 2.4 GHz Intel E-series Xeon (Ivy Bridge). The core includes 256-bit SIMD (AVX) vector floating-point and 32KB L1 and 256KB L2 caches. Each processor-core permits up to two-way hyper-threading and is connected to the four DDR3-1866 memory controllers and 20MB L3 cache via an arbitrated ring-bus. As a result of greater hardware resources the STREAM bandwidth per socket on Edison is higher than that achieved on BlueGene/Q nodes. Nodes of Edison feature two processors sockets and are then grouped into quad-node blades for connection to a dragonfly topology interconnect via a Cray Aries NIC. In total, the machine contains 5200 compute nodes, providing over 120,000 compute cores.

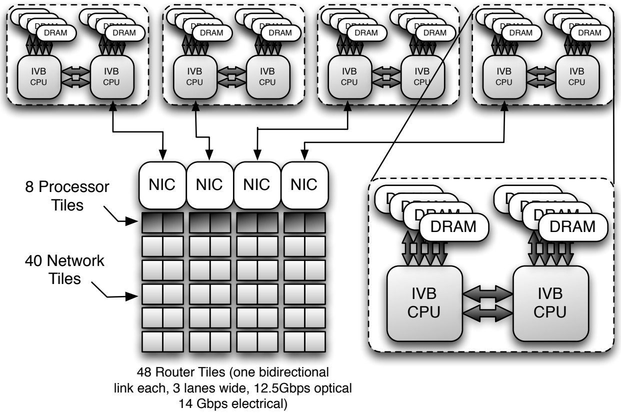
DRAM
IVB CPU
NIC
NIC
NIC
NIC
8 Processor Tiles
40 Network Tiles
DRAM
DRAM
IVB CPU
IVB CPU
48 Router Tiles (one bidirectional link each, 3 lanes wide, 12.5Gbps optical 14 Gbps electrical)

Figure 2: Cray XC30 Node Architecture

3. Benchmarks Descriptions and Problem Definitions

In the emerging many-core era, it becomes impractical for applications to run with an MPI only programming model. The rapidly increasing number of cores per node and the relatively slow growth of associated memory and memory bandwidth means that each MPI task will not only be able to access smaller amounts of memory and memory bandwidth than today, but will also encounter more contention for on- and off-node network resources. For these reasons, there is increasing pressure to move applications to hierarchical threaded execution models where, say, MPI is used for decomposing problems across nodes at a coarse level and a lightweight threaded API is used to perform finer-grained compute work (and possibly handle communications) on a node. To this end, we are interested in the ability of the applications presented below to scale with an increasing numbers of threads and how that performance compares to an MPI-only programming model.

For each application, we present a set of strong scaling results using an MPI-only execution model and an MPI+OpenMP model. In the former, we increase only the number of MPI ranks and in the latter we increase the number of MPI ranks but only use 1 MPI rank per socket, filling the remainder of the compute cores on each socket with OpenMP threads. In each case, we completely fill each node on each machine with either tasks or tasks and threads (though we do not examine hyperthreading on Edison) and then compare results on a node-per-node basis and examine what tradeoffs or limitations might exist.

In summary, for the three test platforms our execution modes are:

- **Mira/Vulcan**
  - 16 MPI tasks per node, 4 OpenMP threads per core
  - 1 MPI task per node, 64 OpenMP tasks
- **Hopper**
  - 24 MPI tasks per node
  - 4 MPI tasks per node, 6 OpenMP threads per NUMA domain (24 OpenMP threads in total)
- **Edison**
  - 24 MPI tasks per node
  - 2 MPI tasks per node, 12 OpenMP threads per NUMA domain (24 OpenMP threads in total)

Each of the benchmark problems is a smaller version of larger problems defined for the NERSC8/Trinity procurement. Each problem was weak-scaled down by two to four times in order to provide a sufficiently interesting range of data points for given the compute capabilities of the test platforms. Each of the benchmark codes and problem definitions is now briefly described.

3.1 MILC

MILC (MIMD Lattice Computation) is a widely used, computationally intense application designed to compute gauge fields 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 3x3 complex vectors, at the grid points and gluon variables, defined as 3x3 unitary matrices, defined at the 'links' between grid points [3]. The most computationally intense part of the program is the conjugate gradient solver which determines how the motion of the quarks is affected by the gluons [3]. The four dimensional lattice is decomposed so that the sub-grid assigned to each MPI task has the minimum possible surface-to-volume ratio. Following Gottlieb and Tamhankar [7], the code has fine-grain parallelism implemented with OpenMP directives, mostly on loops over all grid points in the lattice. Communications in MILC are largely dominated by point-to-point transfers associated with the 4D halo exchanges and global reductions associated with the conjugate gradient solver. MILC has been an important part of previous NERSC procurements as it is representative of NERSC's high energy physics workload and because the stencil computation and conjugate gradient solver stress both the memory and interconnect bandwidths, respectively.

The four dimensional space-time grid (x, y, z, t) used for this paper has dimensions 64x64x64x192. At the base concurrency of 12288 MPI tasks, this yields an 8x8x8x8 grid for each MPI task.

3.2 GTC

GTC is a 3-dimensional code used to study microturbulence in magnetically confined toroidal fusion plasmas via the Particle-In-Cell (PIC) method [4]. GTC is used for fusion energy research and thus represents an important part of NERSC's workload. It solves the gyro-averaged Vlasov equation in real space using global gyrokinetic techniques and an electrostatic approximation. The Vlasov equation describes the evolution of a system of particles under the effects of self-consistent electromagnetic fields. The unknown is the flux, $f(t, x, v)$, which is a function of time t , position x, and velocity v, and represents the distribution of particles (electrons and ions) in phase space. This model assumes a collision-less plasma; i.e., the particles interact with one another only through a self-consistent field and thus the algorithm scales as N instead of N^2, where N is the number of particles. The version of GTC used here uses a fixed 1-D spatial decomposition with 64 domains in the toroidal direction and P particle domains within a toroidal domain for a total of 64*P MPI tasks. Fine-grained parallelism is implemented by using OpenMP over the particles in a particle domain and some grid related work within a toroidal domain. Communications in GTC are largely dominated by MPI allreduces that merge each task's copy of the field and MPI send/receives that move particles between domains. Furthermore, because of the gather/scatter particle operations in GTC, the code is known to be particularly sensitive to memory latency [4], though it is also sensitive to memory bandwidth.

It is not possible to strong scale GTC without fundamentally changing the problem because the number of MPI tasks is fixed by the number of particle domains (see above). Increasing the MPI concurrency would also increase the number of particles being simulated. Hence, rather than examining strong scaling through MPI, we examine the strong scaling through OpenMP threads, i.e. we fix the MPI concurrency and increase the number of nodes used by increasing the number of OpenMP threads per node. The base problem size is defined for 4800 MPI tasks (75 particle domains) with 32,359 particles per MPI task.

3.3 FLASH

FLASH is a publicly available, multi-physics code with core capabilities which include Adaptive Mesh Refinement (AMR) and explicit solvers for hydrodynamics and magneto-hydrodynamics [5, 13]. It has been used to simulate X-ray bursts, Type Ia supernovae, Core Collapse supernovae, galaxy cluster formation and laser-driven High Energy Density Physics (HEDP) experiments. It is parallelized by dividing the underlying mesh into blocks (patches) and assigning the blocks to different MPI tasks. Each block contains a halo of guard cells which are updated after each solver time-advancement. The solvers are multithreaded using conditionally compiled OpenMP directives around either loops over blocks or loops over grid points.

In this paper we run the Sedov test case, which is a pure hydrodynamics problem involving the self-similar evolution of a spherical blast wave from an initial pressure perturbation. The application is configured to use the unsplit hydrodynamics solver and a uniform resolution grid containing 1152^3 grid points. We use a uniform mesh and not the adaptive mesh provided by Paramesh because Paramesh is not multithreaded and so the MPI vs MPI+OpenMP comparison would be less interesting. The uniform mesh provides 1 block per MPI task and so the OpenMP directives over grid points are conditionally compiled into the application. Note that the uniform mesh is an important capability which is appropriate in simulations with relatively smooth fluid flow, such as simulations of weakly-compressible homogeneous isotropic turbulence carried out on BG/L [1].

3.4 Finite Element (MiniFE)

Many of the engineering applications in use at Sandia and other HPC computing sites require the implicit solution of a nonlinear system of equations. As these systems increase in size and complexity, the runtime becomes dominated by the performance of basic mathematical operations that are employed in the solver routines - typically these feature some combination of parallel dot-products, vector scaling or AXPBY operations and sparse-matrix-vector products.

The MiniFE mini-app [9], part of the Mantevo suite [8], is an implementation of a finite-element generation, assembly and solve for an unstructured problem. Although the solver employed in MiniFE - a simple conjugate gradient solver - is more simplistic than those used for production applications, the kernels that contribute to the CG solver provide many of the characteristics of those used in production applications and, in a number of studies, have been shown to provide reasonable runtime and behavioral correlation.

3.5 Unstructured Mesh Transport (UMT)

UMT is a proxy application from the NNSA's ASC program, written and maintained by LLNL, which performs the solution of a time- and energy-dependent discrete ordinate radiation problem in three dimensions on an spatially unstructured grid. The algorithm employs deterministic S_n methods to model the transfer of thermal protons in a three dimensional domain. Parallelism within the UMT code is provided by decomposing the unstructured spatial grid onto MPI tasks and using OpenMP threads to implement fine-grained parallel processing over angles during the transport phase.

4. Performance Results

4.1 STREAM

To measure the memory bandwidth performance, which can significantly impact many scientific codes, we ran the STREAM benchmark on each of the test platforms. For each platform, we configured the test to utilize 60% of the on–node memory. For Hopper and Edison, we ran separate copies of STREAM on each of the NUMA domains and used enough OpenMP threads to fill each domain. For Mira, we only ran one instance of the benchmark and ran with 64 OpenMP threads. The reported STREAM Triad results are as follows: Hopper - 53.9 GB/s, Edison - 103.3 GB/s, Mira - 28.6 GB/s per node.

Knowing the relative magnitude of the memory bandwidth between machines can be useful when comparing the performance of codes that are memory bandwidth sensitive. In Figure 3, we show roofline models of the three test platforms using the measured STREAM values and the known peak gigaflops/s/core rate defined by each platform's CPU clock speed and peak flops/clock. The roofline model [15] is a convenient visual means of identifying if a code is compute bound or memory bandwidth bound and can be used to guide optimization efforts. If a code makes good use of spatial and temporal locality in its memory references, the memory subsystem should be able to keep the vector units of the CPU full and thus the code should operate at near the peak floating point rate (compute bound). If not, a code's performance will be limited by the memory bandwidth (memory bandwidth bound). In the roofline model, these two variables, floating point performance and memory bandwidth, are assumed to be related through operational intensity, i.e the number of floating point operations per byte of DRAM traffic. Thus, the roofline of a platform is defined by the following formula

$$\begin{aligned}&PeakGFlops/s = MIN(\\&\quad PeakFloatingPointPerformance\\&\quad PeakMemoryBandwidth * OperationalIntensity)\end{aligned}$$

The roofline for each compute platform can be seen to be divided into two segments. The horizontal segment represents the upper floating point limit imposted by the architecture. The sloped portion of the roofline represents the upper limit of performance imposed by the peak memory bandwidth of the system.

If we now measure the compute intensity and gigaflops/s rate of a code we can plot them in the figure. Codes which tend to fall on the horizontal portion of the roofline for a platform are considered to be compute bound as their performance is limited by the number of floating point operations that a CPU can execute each clock cycle. Codes which fall on the sloping part of the roofline are considered to be limited by memory bandwidth.

Figure 3 shows the measured performance of each of the Trinity/NERSC8 applications when running each application's 'large' test case on Hopper using an MPI-only execution model. The operational intensity for each code was measured using Cray's Craypat performance analysis tool and the gflops/s rates were determined using the floating point counts reported from IPM performance analysis tool and the run time values returned by each application. The results in this figure point out that the applications in the procurement, and those studied in more detail in this paper, are limited in performance by the rate that the memory subsystem can feed the processor. Simply adding more floating point capability will not increase performance. The other observation is that all of the benchmarks have relatively low computational intensity (<1), though it must be stressed that that the data points shown are for the entire code and not for any individual kernel which may show higher performance. Because of this, it will be difficult for any of these applications to attain a platform's peak floating point performance. This fact may have an impact on both machine intercomparisons and the selection of systems for procurement. In the former case, architectures become compared based largely on their peak memory bandwidth and not the inherent computational advantages available on each processor. In the latter case, application developers and system procurement teams may find it easier to choose machines with higher peak memory bandwidth rather than refactoring their applications, or researching new algorithms, to better use the CPU. As CPUs increase in core count and complexity these issues may become increasingly prominent.

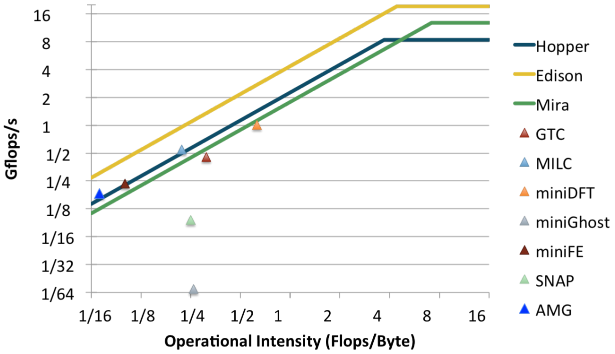

Figure 3: Roofline model of test systems with NERSC8/Trinity benchmarks. For each of the test systems, the plot shows a roofline model (colored lines) constructed from the peak floating point performance per core of each system and the measured memory bandwidth from the STREAM benchmark. Each colored triangular symbol marks the results obtained from Hopper for test cases that run on order 1000 nodes.

4.2 NERSC-6 Applications on Hopper and Edison

While the majority of this paper focuses on performance analysis of codes selected from the Trinity-NERSC8 benchmark suite, we also present results for the NERSC-6 application benchmarks[2] to facilitate comparison to previous benchmarking work on other computational platforms. Like the Trinity-NERSC8 suite, the NERSC-6 benchmarks were selected to span an appropriate cross-section of scientific domains and algorithms. The Community Atmospheric Model (CAM) is a significant component of the climate science workload; it uses 3-dimensional finite volume methods to simulate dynamical (e.g. fluid flow) and physical (e.g. precipitation) processes in the atmosphere. GAMESS implements a broad range of *ab initio* models of quantum chemistry. IMPACT-T is a relativistic particle-in-cell code used to simulate accelerator physics. MAESTRO is an astrophysics code that uses algebraic multigrid methods to simulate pre-ignition phases of Type-IA supernovae. PARATEC is a plane-wave density functional theory code used for materials science; its functionality and performance characteristics are quite similar to MiniDFT, which has supplanted it in the Trinity-NERSC8 benchmark suite. GTC and MILC are included in both benchmark suites and were described in Section 3. Detailed descriptions of the NERSC-6 codes and inputs are available in ref. [2].

One feature that distinguishes Edison's Ivy Bridge processors from Hopper's Magny–Cours processors is the availability of Hyperthreading Technology. When hyperthreading is enabled, each physical core presents itself to the OS as two logical cores. The logical cores share some resources of the physical core (such as cache, memory bandwidth and FPUs), but have independent architectural states. Hyperthreading has the potential to increase resource utilization if an application cannot exhaust a critical shared resource with a single instruction stream. Thus, on Edison jobs can be run in a single-stream mode (with one process per physical core) or dual-stream mode (with two processes per physical core). The sharing of resources generally causes dual-stream jobs to run roughly half as fast as single-stream jobs with the same MPI concurrency, but a net increase in throughput may be achieved if the dual-stream job uses half as many nodes for less than twice the single-stream walltime.

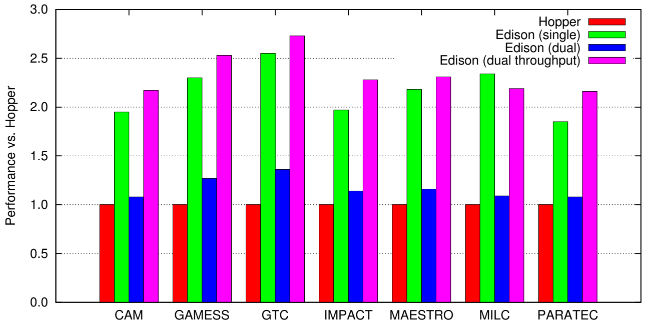

Figure 4: NERSC-6 Application speedup relative to Hopper. Each benchmark was run on Hopper and on Edison in single- and dual-stream modes. The dual-stream throughput accounts for reduced node counts needed in dual-stream mode.

Figure 4 compares the performance of the NERSC-6 benchmarks on Hopper and Edison. Edison's single-stream performance is 1.9-2.6 times faster than Hopper. The dual-stream performance is predictably about 50% less than single-stream, however dual-stream 'throughput performance' (which is identically 2x the dual-stream performance) is up to 2.75 greater than Hopper, and marginally better than the Edison single-stream performance for all codes except MILC. Edison's NERSC-6 SSP is 253 TF/s, nearly double Hopper's 137 TF/s.

4.3 Application Performance On Test Platforms

4.3.1 MILC

The strong scaling performance results for for MILC are shown in Figure 5. The base configuration for these experiments was 12288 MPI tasks (512 nodes on Hopper/Edison and 192 nodes on Mira). Note that, for Mira, the node count is lower as we placed 64 MPI tasks on a node since MILC's memory usage for this problem can easily fit within the 16GB per node available.

Across the range of nodes where they overlap, the MPI-only runs on Edison are 2.2-3.8 times faster than on Hopper whereas on Mira they are 1.9-3.8 times slower. The hybrid models on Edison compared to Hopper show a similar speedup as the MPI-only runs, but the hybrid runs on Mira are 6.8-9.4 times slower than on Edison.

For all three platforms, the hybrid model is slower than the MPI-only model. If we look in ranges where the parallel efficiency is still reasonably good, the hybrid models on Hopper are 2-3x slower than the MPI-only version, on Edison they are about 2x slower, and on Mira they are about 3.5x slower. Looked at another way, on both Hopper and Edison, the hybrid models need approximately twice the concurrency to equal the same performance as an MPI-only model. On Mira, the hybrid model needs nearly four times the concurrency. On the x86 architectures at least, this indicates that there are many serial sections remaining in the code.

As for scaling, the MPI-only/hybrid code on Hopper shows a 2.5x/3.9x speed up over an 8x increase in concurrency whereas on Edison there is a 4.2x/6.5x speedup. On Mira, the same versions show a 22.3x/24.5x speedup over a 64x increase in concurrency. One interesting feature in the bend in Mira's MPI-only scaling curve at 3072 nodes, possibly due to a change in MPI protocols. This is supported by the fact that the hybrid model on Mira does not show this same behavior and it is expected that the decomposition protocol for MILC will minimize the surface to volume ratio of each sub-grid assigned to an MPI task.

While it appears that the Hopper and Edison MPI-only models both become slower than their hybrid counterparts between 1024 and 2048 nodes, this effect is the result of a loss of scaling due to increased MPI traffic at higher MPI concurrencies. If we look at the compute time (wall clock - communication time) then this cross-over disappears and the hybrid compute times are slower than the MPI-only compute times across the range of nodes shown. The compute time only efficiency curves highlight the fact that while all of the models scale well out to 2048 nodes, they decline markedly after that point, presumably because they've reached the point where the serial portions of the OpenMP code start to become important.

The parallel efficiency figure is interesting that, while MILC on Edison shows some evidence of superlinear speedup for both the hybrid and MPI-only models, neither Hopper nor Mira do. The lack of superlinearity in the latter two platforms may simply be due to cache size differences. While MPI-only models of MILC often show superlinearity because of their typical memory footprint on a node, the hybrid version shows more. This is presumably because the working set per core of the hybrid code is smaller than in the MPI only version of the code.

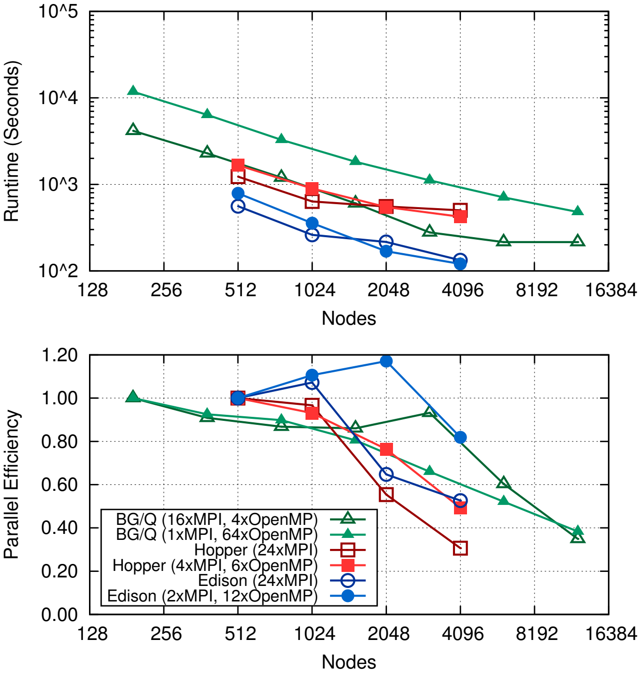

Figure 5: MILC Performance

4.3.2 GTC

The performance results for the OpenMP strong scaling experiment for GTC are show in Figure 6. For Hopper, we ran with 1,2,3 and 6 threads per NUMA domain, for Edison we ran with 1,2,3,4,6, and 12 threads per NUMA domain. On Mira, we started with 16 MPI tasks per node and 4 OpenMP threads and then decreased/increased the number of MPI-tasks/OpenMP-threads per node by factors of two until we reached 64 OpenMP threads per node. On Hopper, GTC speeds up 4.1x using six threads, on Edison it sees a 9.2x speedup over twelve threads, and on Mira it sees an 8.72x speed up when going from four threads per node to 64 (a factor of 16) The differences in the performance between the three different platforms may, to first order, be explained by differences in clock speed and memory bandwidth.

On a node per node basis, Edison is approximately 2-2.6x faster than Hopper, increasing with node count, and Mira is 2.6-2.8x slower than Hopper, with Edison being approximately 7-7.8x faster than Mira. To look at it in a different way, to run as fast as the 200 node case on Edison, Hopper requires three times as many nodes and Mira requires over nine times as many nodes. Based on the trends in the scaling curves and in the parallel efficiency curves, we do not expect a point where Mira's performance exceeds Edison's even if we ran the problem on Mira at a higher node count.

To first order, the differences in the parallel efficiency curves can be understood by removing the MPI communication times from the run times and then recalculating the parallel efficiency. Following that procedure, all three platforms follow nearly the same parallel efficiency curve, with Edison's curve being only marginally affected by this correction. At larger numbers of nodes, the overall performance of GTC becomes limited by the growing influence of MPI collective (allreduce) communications. However, this appears to be less of a factor on Edison as evidenced by it's better scaling.

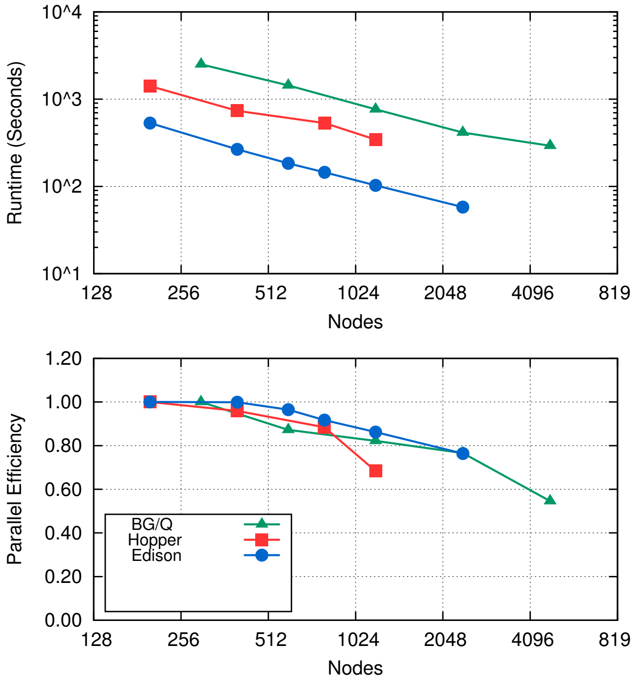

Figure 6: GTC Performance

4.3.3 FLASH

The FLASH experiments are run with a uniform resolution grid of 1152^3 cells on 512 to 4096 nodes. In the 1 MPI rank per core configuration, this gives a workload per MPI rank which is representative of a typical production FLASH simulation with Paramesh. In such a simulation, each MPI rank typically updates 10 to 20 blocks, each consisting of 16^3 cells. In comparison, our base 512 node experiment on Hopper assigns a local grid of 124416 cells to each MPI rank. This local grid contains approximately the same number of cells as 30 16^3 blocks and so our strong scaling study is important since it tests key workloads corresponding to 30, 15, 8 and 4 16^3 blocks per MPI rank. In all experiments we obtain the FLASH run time from the "evolution" time stamp in the FLASH log file. This is appropriate because initialization time is small and becomes negligible in production simulations which consist of multiple 12-hour runs chained together.

The performance results in Figure 7 show that at a given node count the fastest time to solution is obtained on Edison in 1 MPI rank per core configuration. We see that it takes approximately a factor of 8x more nodes on Mira to improve upon a given Edison time. The parallel efficiency of FLASH tails off at higher node count mainly because the unsplit hydrodynamics solver in FLASH replicates certain guard cell computations. We find that the biggest strong scaling loss comes from the computation of Riemann state values for *all* cell interfaces within a single block. Work could be saved by communicating the guard cell Riemann state values instead of replicating this computation. The communication vs. computation trade-off should be investigated because the replicated work is actually more than the necessary local work in FLASH simulations with Paramesh and blocks of 16^3 cells.

In all cases, the 1 MPI rank per core experiments are faster than the 1 MPI rank per NUMA domain experiments. One notable observation is that the Hopper platform shows the smallest difference between the MPI per core and MPI per NUMA domain performance. This is partially because there are only 6 OpenMP threads per MPI rank instead of 12 (Edison) or 64 (Mira) and so the impact of serial code sections is smaller. But perhaps more interesting is that the total time in MPI is always less in the per NUMA domain experiments than in the per core experiments. This is the opposite to what we observe on both Edison and Mira and perhaps indicates contention in the network is slowing down the MPI rank per core guard cell exchange on Hopper.

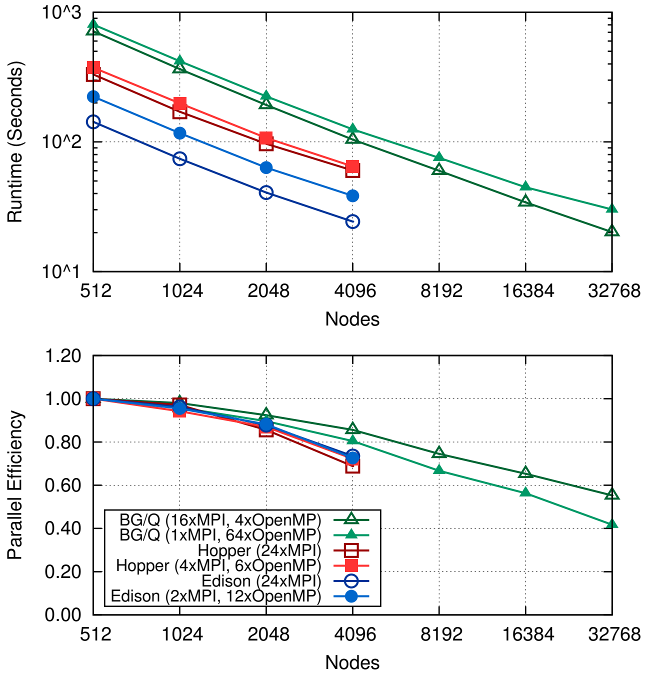

Figure 7: FLASH Performance

4.3.4 MiniFE

The results for MiniFE are for a strong scaled study with data points at 512, 1024, 2048 and 4096 nodes for each platform. The input parameters were chosen to use nominally 4 TB of aggregate memory capacity, 8 GB, 4 GB, 2 GB and 1 GB per node respectively, in order to be sufficiently larger than the last level cache and hence fully utilize the memory hierarchy. The metric chosen for this study is overall solve time for the conjugate gradient solver. The CG solver contains three distinct operations, a DOT product, a WAXPY operation, and a sparse matrix vector (SpMV) product. All three operations have been parallelized with OpenMP. The SpMV product takes the majority of time in the calculation, approximately 80% for the Hopper cases at 512 nodes. The amount of time spent in MPI communication is not negligible and there can be effects as scale increases on less balanced architectures.

The timing and parallel efficiency results for MiniFE are shown in Figure 8. In general MiniFE performs similarly for the two mixes of MPI and OpenMP on the respective platforms. There are some deviations at 4096 nodes, but not significant. MiniFE scales well on both the BG/Q and Edison platforms, near 90% or better parallel efficiency. On the Hopper platform, scaling is somewhat erratic. This behavior has been observed with miniFe in other studies of the Cray XE6 architecture, with the primary contributor being the DOT product operation. This behavior is repeatable and has been demonstrated on multiple instantiations of the architecture. It is believed that it is an artifact of the non-uniform communication performance of the Gemini 3D torus and how the problem is laid out on the machine. Although this is usually not a major performance issue, as can be seen in this study the performance degradation at 4096 nodes is significant as parallel efficiency drops to less than 70%. The authors surmise that if run at 8192 nodes, parallel efficiency would improve to the 90+% range as seen at 2048 nodes. In Summary, Edison provides the best overall time to solution. Both Hopper and the BG/Q platform require approximately four times as many nodes to achieve similar performance.

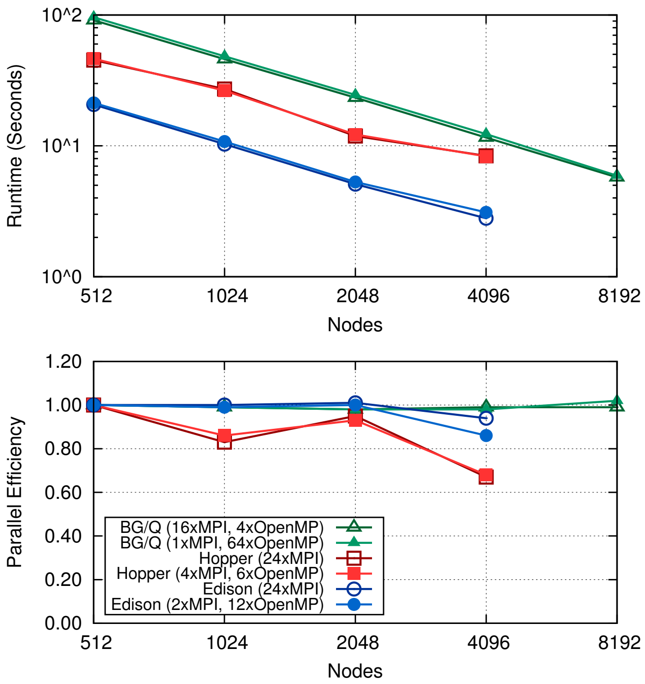

Figure 8: MiniFE Performance

4.3.5 UMT

The results for UMT are for a strong scaled study with data points at 512, 1024, 2048 and 4096 nodes for each platform. The input parameters were chosen to use nominally 4 TB of aggregate memory capacity, 8 GB, 4 GB, 2 GB and 1 GB per node respectively, in order to be sufficiently larger than the last level cache and hence fully utilize the memory hierarchy. The metric chosen was cumulative work time. For UMT much of the work for each MPI rank does not contain OpenMP directives, the only section of the code that has OpenMP is the step which loops over all the angles of the transport problem. Although this is a major computational part of the solve phase, if a node has weak single core performance, cases with minimal MPI parallelism may inherently have lower performance. However, as shown below the results there can be exceptions.

The timing and parallel efficiency results for UMT are shown in Figure 9. For the BG/Q and Edison platforms, the cases which use more MPI parallelism perform significantly better than the respective cases with higher OpenMP parallelism, as surmised above. However, for the Hopper platform the two cases show essentially equal performance across all scales. Further analysis shows that the 24xMPI case does indeed spend approximately 20% less time in the computational sections of the code, but spends approximately 70% more time in the MPI routines than the 4xMPI/6xOpenMP case. So for this problem, on this platform, the total solve time is roughly equal.

Looking at the parallel efficiency graph, Edison shows good scaling for both cases. The BG/Q platform has good scaling for the 1xMPI/64xOpenMP case, but scaling drops off significantly as the number MPI ranks per node is increased in the 16xMPI/4xOpenMP case. Hopper scales consistently between the two cases, but overall performance drops to less than 40% parallel efficiency at 4096 nodes. In summary, best overall performance is obtained with Edison using 24 MPI ranks per node. Neither Hopper nor Vulcan achieve the same level of performance, even when using four times as many nodes.

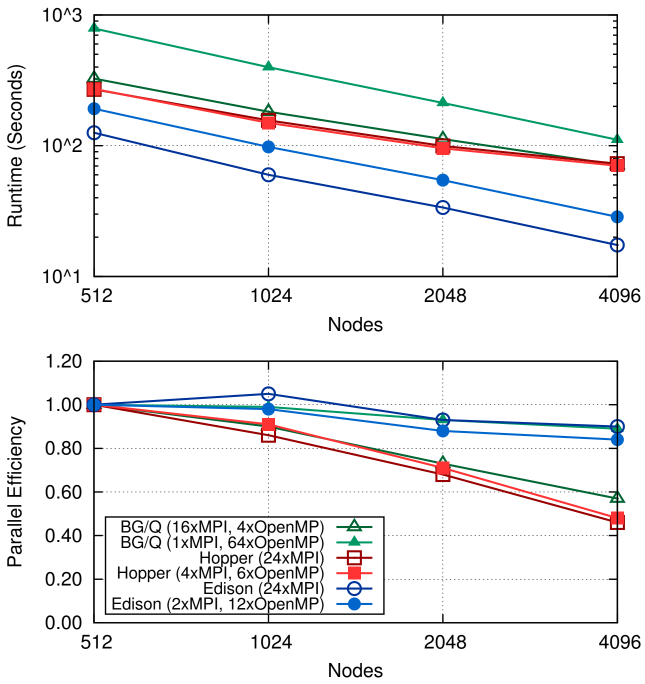

Figure 9: UMT Performance

5. Related Work

The work most directly related to this study is that of Kerbyson et al. [11] who compared the performance of the IBM Blue Gene/Q with the Cray XE6 and an Infiniband system. That study is different from the work presented herein in several ways: in particular, they presented a more detailed analysis of the network interconnect performance and their application performance comparison focused on weak-scaling of codes without examining thread-level parallelism. Like Kerbyson, we observe the excellent scaling characteristics of the Blue Gene/Q interconnect, but we also observe comparable scaling performance in the Cray XC30 interconnect, making this system attribute less of a discriminating performance factor between the two. Though it is difficult to directly compare weak and strong scaling results of different problems over different node counts, we observe that both GTC and MILC on the Cray XE6 had roughly similar speedups over the Blue Gene/Q as those observed in [11].

6. Summary and Conclusions

As we have seen in this study, it is possible to see significant performance gains on modern architectures with larger faster caches and better memory bandwidth. In terms of raw performance acquired through improvements in memory bandwidth and an improved interconnect, the Cray XC30 is a significant step forward from its predecessor, the Cray XE6. For the MPI-only codes, for the smallest node counts and where parallel efficiency is still good, the range of speedup from Hopper to Edison is about 2x which is expected based on clock speeds and memory bandwidth figures. Comparing Edison to Mira is a bit more interesting as on a per core basis we expect an Edison core to be 5.4x the speed of a Mira hardware core. Comparing on a node level, and correcting for node counts, we expect an Edison node to have 8x the performance of a Mira node (single thread per core). In this case Mira performs more admirably with MiniFE and Flash running on Edison only 4.2 and 4.3x faster, respectively, which is what we might expect if performance on a Mira node had peaked at two hardware threads per core. Both MILC and UMT perform even better showing relative speedups on Edison of about 2-2.5x. For the case of MILC, the improved performance is possibly the result of Mira's 5D torus network being more amenable to MILC's 4D halo exchanges and Mira's ability to hide memory latency, which is also an important performance limiter for MILC. The UMT code also shows a relative speedup on Edison of 2.6x, implying that it is able to exploit at least two hardware threads on Mira effectively and may also be able to exploit hardware threading to hide the latency of its indirect addressing. At higher node counts, all the codes showed a decrease in performance due to increased network traffic resulting in increasing gaps in performance between Edison and the other two platforms. In this case, Edison increased it's performance advantage over Hopper up to a range of about 2.5-3.8x and over Mira from a range of 3.5-6x. The Flash code showed the least degradation of performance relative to Edison (about 10-20%) whereas the other three codes degraded on the order of 30-40%.

Comparing performance at low node counts, the hybrid codes show a similar performance improvement to the MPI-only codes on Edison relative to Hopper of about 1.8-2.5x. Flash, UMT, and MiniFE all perform 3.5-4x better on Edison than Mira, whereas GTC and MILC have poorer OpenMP performance and Edison has a 6.2-6.7x performance advantage over Mira. The latter two codes are older full applications, more representative of NERSC's current workload than the more recently developed mini-applications. Thus, we expect that if we migrate to newer code bases with better OpenMP implementations and less serialization between OpenMP sections, this performance difference would be reduced.

At high node counts the effect of increasing MPI traffic, again, decreases application performance for the hybrid models and widens the performance advantage of Edison. However, in general, because we are using far fewer MPI tasks than in the MPI-only models, the effect for most codes is relatively modest with Edison gaining at most about a factor of 0.0-1.5x over Hopper and Mira. Still, both GTC and MILC take substantial performance hits on Mira at high node counts with Edison's advantage increasing to 7.8x and 9.4x, respectively, thus adding network overhead to already marginal OpenMP performance.

Through this study, we can see the advantages of Edison's improved memory bandwidth and interconnect on performance both on individual application performance and on NERSC's system performance metric, the SSP. Comparing the XC30 system to the BG/Q system is more difficult. While the Cray provides rougly 4x greater application performance per core (as shown in Section 4) and many codes lack the degree of fine-grained parallelism needed to maximize performance on BG/Q, the IBM system may be more attractive on an efficiency or total energy cost basis. BG/Q nodes require less electrical power than the XC30: Edison uses about 280 W/node when running LINPACK, and Mira uses about 80 W/node. Thus, at peak utilization, one Edison node uses 3.5x the power of a Mira node, which places the two systems on near-equal footing when compared by a performance per Watt metric.

Finally, we also observed that all of the benchmarks used in this study have markedly low computational intensity making them sensitive to memory bandwidth performance. Figure 3 clearly shows that all of the codes used in this study are significantly limited by the memory bandwidth of each platform, and overall the results principally track the differences in memory bandwidth between the machines. All would benefit greatly from code optimizations to increase their computational intensity, regardless of any increases in memory bandwidth performance. A telling feature of this study is that while a few purpose-written mini-applications (e.g. MiniFE) may exhibit nearly co-equal performance between MPI and hybrid versions, the hybrid version of most applications is slower, in some cases markedly so. It is clear that application developers will need to invest substantial time and effort into either refactoring their codes for the many-core era or selecting new algorithms to improve threaded performance. These changes must be made with one eye toward reducing serialization and communications, and another toward increasing data reuse in order to reduce memory traffic. In fact, this work has already begun for several of these codes [10, 12]. The hybrid results presented here simply reflect a slightly older code base.

7. Acknowledgements

All authors from Lawrence Berkeley National Laboratory were supported by the Office of Advanced Scientific Computing Research in the Department of Energy Office of Science under contract number DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under contract DE-AC04-94AL85000.

This research 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.

References

[1] K. Antypas, A. Calder, A. Dubey, R.T. Fisher, M.K. Ganapathy, B. Gallagher, L.B. Reid, K. Riley, D.J. Sheeler, and N. Taylor. Scientific Applications on the Massively Parallel BG/L Machine. In *PDPTA*, volume 2006, pages 292–298, 2006.

[2] K. Antypas, J. Shalf, and H. Wasserman. NERSC-6 Workload Analysis and Benchmark Selection Process. Technical Report LBNL 10143, Lawrence Berkeley National Laboratory, 2008.

[3] 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.

[4] S. Ethier, W. M. Tang, and Z. Lin. Gyrokinetic particle-in-cell simulations of plasma microturbulence on advanced computing platforms. *Journal of Physics: Conference Series*, 16:1–15, 2006.

[5] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal Supplement Series*, 131(1):273, 2000.

[6] AI Geist. Sustained petascale: The next MPI challenge. In *EuroPVMMPI*, October 2007.

[7] S. Gottlieb and S. Tamhankar. Benchmarking MILC with OpenMP and MPI. *Nucl.Phys.Proc.Suppl.*, 94:841–845, 2001.

[8] M.A. Heroux. Mantevo project web page. `https://software.sandia.gov/mantevo/`.

[9] M.A. Heroux et al. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, September 2009. `https://software.sandia.gov/mantevo/`.

[10] B. Joó, D. D. Kalamkar, K. Vaidyanathan, M. Smelyanskiy, K. Pamnany, V. W. Lee, P. Dubey, and W. Watson III. Lattice QCD on Intel®Xeon Phi$^{\mathrm{TM}}$Coprocessors. *Lecture Notes on Computer Science*, 7905:40–54, 2013.

[11] Darren J. Kerbyson, Kevin J. Barker, Abhinav Vishnu, and Adolfy Hoisie. Comparing the performance of Blue Gene/Q with leading Cray XE6 and InfiniBand systems. In *Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems*, ICPADS '12, pages 556–563, Washington, DC, USA, 2012. IEEE Computer Society.

[12] L. Oliker. personal communication, 2013.

[13] The Flash Center for Computational Science, University of Chicago. FLASH User's Guide. Version 4.0. `http://flash.uchicago.edu/site/flashcode/user_support/flash4_ug.pdf`, September 2012.

[14] Challenges for the message passing interface in the petaflops era. `www.cs.uiuc.edu/homes/wgropp/bib/talks/tdata/2007/mpifuture-uiuc.pdf`.

[15] Samuel Webb Williams, Andrew Waterman, and David A. Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Technical Report UCB/EECS-2008-134, EECS Department, University of California, Berkeley, Oct 2008.