Performance on Trinity (a Cray XC40) with Acceptance-Applications and Benchmarks

Nathan Wichmann, Cindy Nuss, Pierre Carrier, Ryan Olson, Sarah Anderson and Mike Davis
Cray Inc.,
wichmann, cnuss, pcarrier, ryan, saraha, u3186@cray.com

Randal Baker, Erik W. Draeger, Stefan Domino, Anthony Agelastos, Mahesh Rajan
rsb@lanl.gov, draeger1@llnl.gov, spdomin@sandia.gov, amagela@sandia.gov, mrajan@sandia.gov

***Abstract*—Trinity is NNSA's first ASC Advanced Technology System (ATS) targeted to support the largest, most demanding nuclear weapon simulations. Trinity Phase-1 (the focus of this paper) has 9436 dual-socket Haswell nodes while Phase-2 will have close to 9500 KNL nodes. This paper documents the performance of applications and benchmarks used for Trinity acceptance. It discusses the early experiences of the Tri-Lab (LANL, SNL and LLNL) and Cray teams to meet the challenges for optimal performance on this new architecture by taking advantage of the large number of cores on the node, wider SIMD/vector units and the Cray Aries network. Application performance comparisons to our previous generation large Cray capability systems show excellent scalability. The overall architecture is facilitating easy migration of our production simulations to this 11 PFLOPS system, while improved work flow through the use of Burst-Buffer nodes is still under investigation.**

Keywords-component; Cray XC40, AVX2, MPI, OpenMP, performance optimization

I. INTRODUCTION

Trinity is architected to meet the capability simulation needs of NNSA's ASC program. It is anticipated that due to its improvements in compute, memory and storage capabilities, it will enable larger model geometries and support higher fidelity physics, while meeting programmatic time-to-solution needs. Acceptance of the Phase-1 of the Trinity procurement was concluded in December of 2015. Phase-2 of the Trinity procurement is currently in progress, as volume shipments of Intel's KNL processors facilitate installation and acceptance in July 2016. Trinity architecture introduces new challenges to the code developers and analysts. These include the transition from multi-core to many-core, deeper memory hierarchies and wider SIMD/vector units. Additionally, we will have for the first time on a large production capability system, high-speed solid-state storage Burst-Buffer nodes, which promise to improve check point/restart reading and writing efficiencies and enable improved work flow through optimal movement of data in an analysis cycle. An overview of the Trinity and NERSC-8 procurement considerations can be found at Reference [1].

II. TRINITY ARCHITECTURE

The Trinity architecture is shown in Figure 1. The Phase-1 Haswell partition has 9,436 nodes with dual-socket Intel Xeon ES-2698 v3 running at 2.3GHz. Each processor has 16 cores and 4 memory channels connected to four 16GB DDR4 DIMMS clocked at 2.133GHz. The processors are set up to support Intel® Hyper-Threads and Intel® Turbo Boost and the operating clock frequency varies with the thermal load.

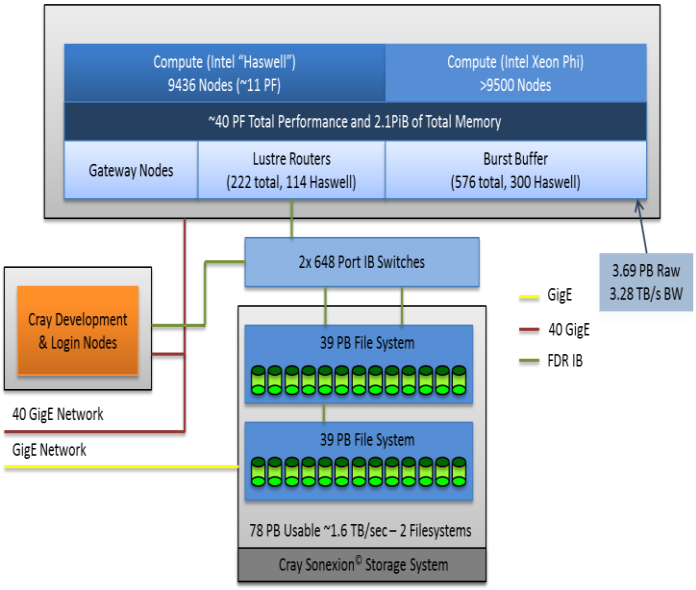

Figure 1. Trinity Architecture Diagram

Assuming a nominal 2.3GHz operation, the peak node double precision performance is: 32cores*16FLOPs/cycle*2.3GHz = 1,177.6 GFLOPS/node. Each core is capable of 16 DP FLOPs per cycle from the two 256 bit AVX2 units with FMA. Trinity is listed at 8,101 TFLOPS on top500.org and 182.6 TFLOPS on hpcg-benchmark.org.

III. ACCEPTANCE TESTS PERFORMANCE RESULTS

A. ASC Capability Improvement (CI) Application Performance

ACES management recognized the importance of good application performance at scale and made it a key Trinity acceptance requirement by specifying a set of metrics to quantify and measure the performance. A key figure used to gauge performance at near full scale is the Capability Improvement (CI) metric, which is computed as an average improvement in performance over Cielo (Cray XE6) [2], of three ASC applications: PARTISN (from LANL), Nalu (from SNL) and Qbox (from LLNL). The baseline performance data was collected on our previous generation ASC ACES platform Cielo,

using more than 2/3 of its compute partition. The CI metric is defined as:

CI Metric = problem-size-increase * run-time-speedup

The target performance for the CI metric is 8X over the baseline Cielo performance, but split into 4X for the Phase-1 Haswell partition (the focus of this paper) and 4X for the Phase-2 KNL partition. Such a metric was also used in the acceptance benchmarks of our previous generation ASC ACES capability platform, Cielo [2][3]. Table 1 provides side-by-side comparison of a few performance related architectural parameters of Cielo and Trinity.

Table 1. Cielo, Trinity Architectural Parameters

System	Cielo	Trinity
Total Nodes	8,894	9,436
Total Cores	142,304	301,952
Processor	AMD MagnyCours	Intel Haswell
Processor ISA	SSE4a	AVX2
Clock Speed(GHz)	2.40	2.30
Cores/node	16	32
Memory-per-core(GB)	2	4
Memory	DDR3 1,333 MHz	DDR4 2,133 MHz
Peak node GFLOPS	153.6	1,177.6
Channels/socket	4	4
Processor Cache L1(KB) L2(KB) L3(MB)	8 x 64 8 x 512 10	16 x 32 16 x 256 40
Interconnect Topology	Gemini 3D Torus 18x12x24	Aries Dragonfly

The following sections describe the three applications picked for the CI benchmark, their performance characteristics and specific efforts that were undertaken to meet the target performance set for Phase-1. These applications are representative of the production simulations planned for Trinity and should suggest possible approaches for tuning other production applications.

1) SIERRA/Nalu:

The SIERRA/Nalu is a low Mach CFD code that solves a wide variety of variable density acoustically incompressible flows spanning from laminar to turbulent flow regimes. SIERRA Mechanics [4] simulation code suite is the principal mechanics code used by SNL in support of the U.S. Stockpile Stewardship program. Open source versions of Nalu (version 1.0.0) along with the Trilinos solver (version 12.0.0) were used for this benchmark. Nalu is fairly representative of implicit codes that have been developed as part of Sandia mechanics simulation code, SIERRA. Open source Nalu can be downloaded from Github [5]. This generalized unstructured code base supports both elemental (control volume finite element) and edge (edge-based, vertex-centered) discretizations in the context of an approximate pressure projection algorithm (equal order interpolation using residual based pressure stabilization). The generalized unstructured algorithm is second order accurate in space and time. A variety of turbulence models are supported, however, all are classified under the class of modeling known as Large Eddy Simulation (LES). The chosen coupling approach (pressure projection, operator split) results in a set of fully implicit sparse matrix systems. Linear solves are supported by the Trilinos Tpetra interface.

Nalu's code base has been demonstrated to be 64-bit compliant and represents the path towards advanced architectures and can support mesh and degree-of-freedom counts well above the 2.14 billion count. The calculations are computationally intensive and require good cache usage. In typical applications, hundreds of thousands of time steps must be used. Communication patterns include both point-to-point exchanges typical of sparse graphs, consistent with assembly of partial sums, and collective reduction operations including global minimums, maximums, and summations. This code base is fairly representative of a wide range of implicit codes that have been developed in support of the Advanced Simulation and Computing (ASC) Integrated Codes (IC) project.

a) Problem Description:

The test problem of interest is a turbulent open jet (Reynolds number of ~6,000) with passive mixture fraction transport using the one equation Ksgs LES model. The problem is discretized on an unstructured mesh with hexahedral elements. The baseline problem R6 mesh consists of nine billion elements, with the total degree-of-freedom count approaching 60 billion. Given the pressure projection scheme in the context of a monolithic momentum solve, the maximum matrix size is ~27 billion rows (momentum) followed by a series of smaller 9 billion row systems, i.e., for the continuity system (elliptic Pressure Poisson), mixture fraction and turbulent kinetic energy.

*b) Figure of Merit (**FOM**)Description:*

Two FOMs were used; both involve the solution of the momentum equations. The speedups of the two metrics are weighted to produce a single speedup factor for Nalu. The first FOM is the average solve time per linear iteration. The second is the average matrix assemble time per nonlinear step. Speedup is defined as:

Speedup = Speedup-solve*0.67 + Speedup-assemble*0.33.

c) Capability Improvement Metric Run:

During a short window in December 2015, the focus was on running Nalu at near full scale of Trinity using as many nodes as available for the purposes of acceptance. The best performance was measured in a run using 9420 nodes (301,440 cores or MPI tasks) using the 9 Billion element R6 mesh. Therefore the complexity increase used in the CI computation was 1, i.e., the same mesh was used as for the Cielo baseline run. The

capability improvement as defined previously was measured at 4.009. All of that improvement accrues from the faster run time measured for momentum equation average assemble time (measured value was 31.8274 secs) and the moementum equation average solve time (measured value was 83.0502 secs). The improvement in this run time attests to the superior strong scaling characteristics of Trinity. It is useful to compare weak scaling of Nalu between Cielo and Trinity to supplement the single data point used for the CI metric. Figure 2 provides a weak scaling plot for the Assemble and Solve times.

The excellent scaling of the Nalu assembly computations resulted in a run time performance gain of 4.26X, at 9,420 nodes of Trinity, over the Cielo run at 8,192 nodes. This, combined with a performance gain of 3.89X for the matrix solve, resulted in the CI metric value of 4.009.

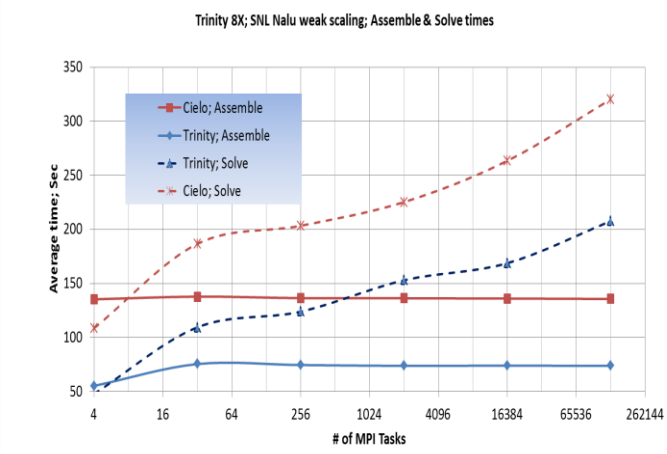

Figure 2. Trinity, Cielo Nalu weak scaling performance

2) PARTISN

LANL's PARTISN particle transport code [6] provides neutron transport solutions on orthogonal meshes in one, two, and three dimensions. A multi-group energy treatment is used in conjunction with the Sn angular approximation. Much effort has been devoted to making PARTISN efficient on massively parallel computers. The package can be used for time-dependent calculations where even one simulation can run for weeks on thousands of processors. The primary components of the computation involve KBA sweeps and associated zero-dimensional physics. The KBA sweep is a wave-front algorithm that provides 2-D parallelism for 3-D geometries, and is tightly coupled by dependent communications.

PARTISN relies heavily on MPI_Isend/MPI_Recv, while the most frequent collective is MPI_Allreduce. For a 1,024 rank run, the code executed around 6M sends, 6M recvs, and approximately 4k Allreduces.

a) Problem Description:

The test problem used is MIC_SN (MIC with group-dependent Sn quadrature). This problem is weak-scaled in the Y and Z dimensions so as to maintain a constant block shape per processor. A small set of parameters in the input file (jt, kt, yints, zints) are scaled to set up inputs for the weak scaling study determining the number of zones/core. These parameters are doubled when the core count/MPI task count is quadrupled. The number of OpenMP threads for each MPI task is also specified in the input file. The Cielo baseline runs with 2,880 zones/core were collected with four OpenMP threads per MPI rank. For runs on Trinity input parameters that led to 2,880, 5,760 and 11,520 zones/core were used to generate control files for runs up to 9,418 nodes (301,376 cores). PARTISN builds with both the Intel and the Cray CCE compilers, with and without OpenMP threading, were investigated for performance. Since the CCE compiler had slightly lower (1-2%) performance than the Intel compiler, the latter was used for the CI benchmark. A study of the hot-spots and MPI communications was conducted. The dominant routine, opt_sweep3d(), which actually performs the KBA sweep that comprises the wave-front algorithm, took 85% of the run time. As the code team had already ensured excellent vectorization of this function, no improvements were found or needed for the CI benchmark effort.

MPI profiling showed that MPI Isend/MPI_Recv communications were frequent on the 2D processor mesh. About 60% of the messages were 64 KB or smaller. In applications with significant time spent in point-to-point communications, optimal MPI rank mapping can lead to good gains in performance. On Cray systems like Trinity, one may experiment with a simple environment variable setting

MPICH_RANK_ORDER_METHOD=n

to study the impact of round-robin rank placement (n=0), SMP rank placement (n=1; default) or folded rank placement (n=2). For PARTISN a custom rank order placement obtained with the use of the Cray utility *grid_order* was very beneficial. An example of a remap with *grid_order* for a run on Trinity with 301,376 MPI ranks is provided below.

grid_order -R -Z -m 301376 -n 32 -g 554x544 \
-c 4x8 >MPICH_RANK_ORDER

The parameters specify:

-R is row-major ordering of ranks
-Z option (default) lists successive rows of cells in the same order
-m max rank count
-c is the desired node MPI grid
-g is the global MPI grid

At run time setting the environment variable:

MPICH_RANK_REORDER_METHOD=3

uses the MPICH_RANK_ORDER file to map MPI ranks to cores and nodes to ensure most of the communication exchanges are within a node, thereby lowering the overall MPI communication time. The utility g*rid_order* does not take system topology into account: it simply "repacks" MPI ranks so that Cartesian mesh communication neighbors are more often on a node. An experiment with PARTISN using *grid_order* on a

run using 16,384 PEs showed a 42% improvement in the MPI time and an 18% improvement in overall run time. Figure 3 shows the speedup resulting from the use of *grid_order* at various scales for PARTISN on Trinity.

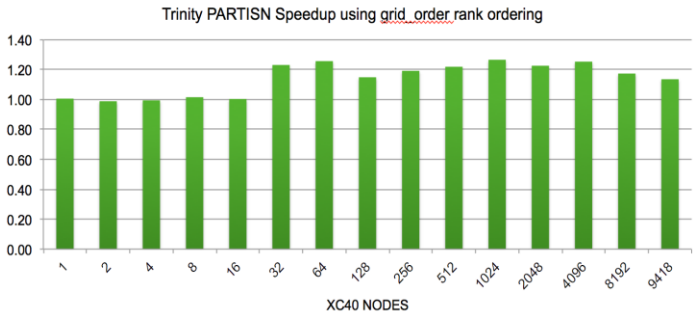

Figure 3. Performance gain with grid_order for PARTISN

*b) Figure of Merit (**FOM**) Description:*

For PARTISN the FOM used is the *Solver Iteration Time*. Ideally this should stay constant for weak scaling. When the baseline performance data was measured on Cielo, optimal scaling and performance were observed with 4 MPI tasks per node and 4 OpenMP threads per MPI task. The results of an investigation to find the optimal MPI Task/OpenMP thread mapping on Trinity, with 2,880 zones/core (labeled "1X") is shown in Figure 4. One thread/MPI rank gave the best performance on Trinity.

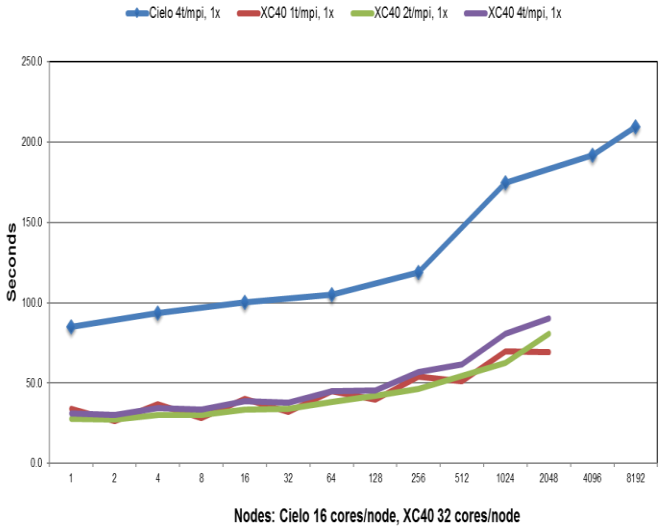

Figure 4. PARTISN MPI task and threading performance

A number of runs on Trinity were attempted to obtain the best possible performance for calculation of the CI metric. Figure 5 compares the scaling plots against the baseline Cielo measurements. For the all the runs on Trinity in this figure one OpenMP thread per MPI task was used. 2,880 zones/core and 11,520 zones/core are the two input cases shown. The label 'asis' refers to default MPI grid mapping and the label 'grid' refers to a run with *grid_order* mapping as previously described.

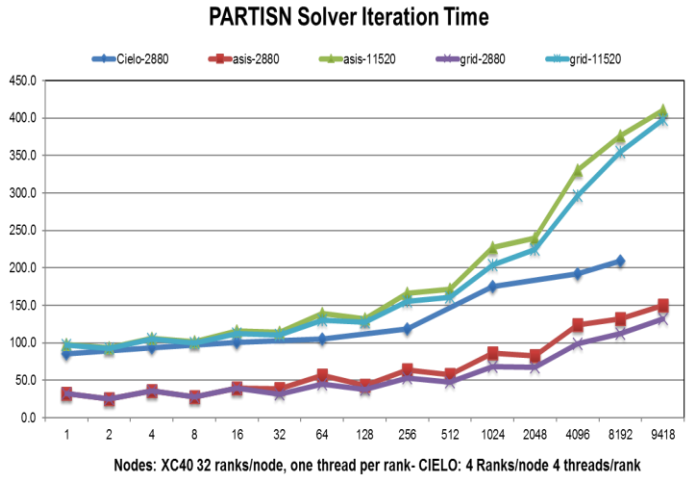

Figure 5. Weak scaling for Cielo and Trinity, with two problem sizes comparing default and *grid_order* rank reordering

c) Capability Improvement Metric Run:

For the CI computation, a run on Trinity using 9,418 nodes (301,376 cores) and an input of 11,520 zones/core, produces a FOM *solver iteration time* of 397.71 secs. The baseline Cielo run was on 8,192 nodes (131,072 cores) with an input of 2,880 zones/core, and produced a *solver iteration time* of 209.4 secs. This results in a complexity scale factor of 9.19 and a run time ratio of 0.526, which yields a CI value for PARTISN of 4.83.

3) Qbox

Qbox is a first-principles molecular dynamics code used to compute the properties of materials at the atomistic scale [7]. The main algorithm uses a Born-Oppenheimer description of atomic cores and electrons, with valence electrons treated quantum mechanically using Density Functional Theory and a plane wave basis. Nonlocal pseudopotentials are used to describe the core electrons and nuclei, and derived to match all-electron single atom calculations outside of a given cutoff radius. The computational profile consists primarily of parallel dense linear algebra and parallel 3D complex-to-complex Fast Fourier Transforms. Efficient single-node kernels have been found to be necessary to achieve good peak performance. The communication patterns are complex, with nonlocal communication occurring both within the parallel linear algebra library (ScaLAPACK) and in sub-communicator collectives within Qbox, which are primarily MPI_Allreduce and MPI_Alltoallv operations. Threading is currently implemented as a mix of OpenMP and threaded single-node linear algebra kernels supplied by the hardware/compiler vendor. All results presented here were carried out using the qb@LL-r205 branch of the Qbox code.

a) Problem Description:

The Qbox benchmark problem is the initial self-consistent wavefunction convergence of a large crystalline gold system (FCC, a0 = 7.71 a.u). This problem is computationally identical to typical capability simulations of high-Z materials, but easier

to describe and generalize to arbitrary numbers of atoms. A Perl script is used to generate input files for weak scaling study. On Cielo, N=1,600 gold atoms were simulated with a norm-conserving pseudopotential and 17 valence electrons per atom, resulting in 13,600 occupied electronic orbitals. A planewave cutoff of 130 Rydbergs was used, and 2,784 additional unoccupied orbitals was included (approximately 20% of the number of occupied states) to allow finite temperature smearing of the occupation at the Fermi level. The computational complexity of the calculation scales as $O(N^3)$ where N is the number of electronic orbitals. The number of atoms was increased accordingly to generate scaled problems for Trinity capability improvement metric simulations. For example, 2,880 gold atoms would require approximately six times more computations than the Cielo benchmark problem with 1,600 atoms.

*b) Figure of Merit (**FOM**) Description:*

The run time metric for this benchmark is the maximum total wall time, to run a single *self-consistent iteration*, with three non-self-consistent inner iterations (corresponding to an input command of '*run 0 1 3*'). Qbox prints formatted XML tags for the timing of each part of the code at the end of the run, with the self-consistent iteration time marked as follows:

<timing where="run" name=" iteration" min="1234.5 " max="1234.5 "/>

The FOM is the timing in the max field.

Qbox CI performance was investigated extensively so as to improve the performance metric on Trinity. The primary factors impacting the CI metric at various scales were: input parameter *nrowmax*, hybrid coarse/fine-grain parallelism; i.e. number of OpenMP threads per MPI task, optimal MPI task mapping, and the number of atoms on input. Early investigations also showed that the Cray CCE compiler was slightly faster (by a few percent) than the Intel compiler. Use of *craype-hugePages* at 8MB also led to 5% performance gain (tested on small problem size) over the default 4KB page size. Cray Libsci OpenMP parallel linear algebra functions found heavy use in Qbox. The importance of the Cray Libsci usage strongly depends on the size of the problem and the number of nodes available for that run; typically, a problem with 2,400 gold atoms running on 2,048 nodes would spend about half the total time inside the ScaLAPACK BI_Srecv/BI_Ssend routines, with an extra 10% in the BLAS routine ZGEMM, and a few percent in MPI_Alltoallv and FFTW3. Larger problems will see an increase in the time spent in the ScaLAPACK routines. However more time will be spent inside ZGEMM and MPI_Alltoallv as the problem size gets smaller and smaller.

The *nrowmax* variable is used to determine the shape of the rectangular process grid used by Qbox. This process grid is the one used by the ScaLAPACK library. When Qbox starts, the *ntasks*, MPI tasks are assigned to processes arranged in a rectangular array of dimensions *nprow* * *npcol*. The default value of *nrowmax* is 32. The plane-wave basis is divided among *nprow* blocks, and the electronic states are divided among *npcol* blocks. At the program start up a simple algorithm coded in Qbox determines the values of *nprow and npcol*. Note that Cray Perftools includes an *MPI Grid Detection* algorithm that determines the shape of the numerical grid and offers optimum grid orderings, as further discussed below. Values 512, 1,024, 2,048 and 4,096 for *nrowmax* were investigated for their performance impact. At lower scales (< 512 nodes) *nowmax* of 512 was optimal. However as the problem size (number of atoms) and the number of nodes were increased it was found that up to 2,000 nodes the optimal *nrowmax* was 1,024 and at very large scales it was 2,048. This is the result of balancing the communication traffic across different ScaLAPACK functions and parallel FFTs, and is consistent with previous studies.

Performance investigation varying the number of OpenMP threads per MPI task showed variation based on scale. For less than 880 atoms, 2 OpenMP threads/task was optimal. For 2,400 atoms and above and at larger number of nodes 8 OpenMP threads/task gave the best performance. Runs on Trinity with a 5,600 atom Qbox simulation showed 2X performance gain for 8 OpenMP threads/task when compared to 2. The need for ASC applications to improve fine-grained node parallelism is illustrated by this application. The benefit in performance comes from the high parallel efficiency linear algebra OpenMP kernels and from reduced MPI inter-node communication overhead.

Figure 6 compares the weak scaling performance of Qbox measured on Cielo and Trinity. The 1600 gold atoms baseline data collected on Cielo and repeated on Trinity using the same number of 98,304 processing elements showed a nice performance gain of 5.3X on Trinity. The improved weak scaling on Trinity seen in Figure 6 is a consequence of the tuning steps outlined previously.

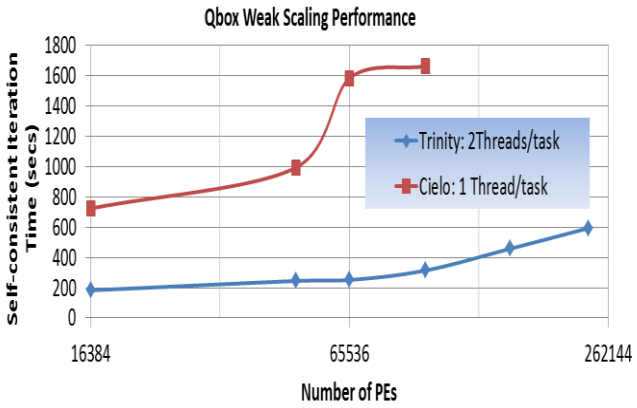

Figure 6. Qbox weak scaling performance; 1600 atoms

c) Capability Improvement Metric Run:

The number of nodes and memory per node on Trinity (4X that of Cielo) permitted runs of Qbox with up to 8,800 atoms. Figure 7 shows the capability scaling characteristics of Qbox on 9,418 nodes of Trinity using 8 OpenMP threads per rank for the various models (except for the 4,000 gold atoms model that used 2 OpenMP threads).

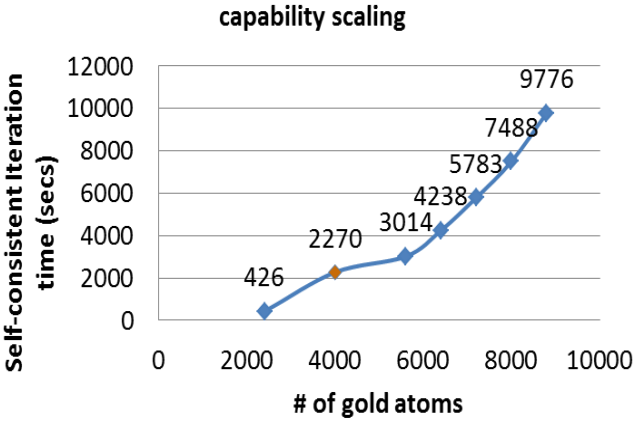

Figure 7. Qbox capability scaling performance, without MPI grid ordering, and hyperthreading.

The FOM, *self-consistent iteration time*, increases from 426 secs to 9,776 secs as the input complexity is varied from 2,880 atoms to 8,800 atoms. As mentioned previously the computation complexity grows as the cube of the number of atoms. The run used for the final CI computation for Qbox is the same 8,800 atoms case as shown in Figure 7 with additional improvement in performance obtained through MPICH_RANK_REORDER using the Cray *grid_order* utility:

grid_order -R -P -c 4,2 -g 68,1108 > MPICH_RANK_ORDER

The parameters specify:

-R is row-major ordering of ranks
-P is a Peano space filling curve (optimal for FFT-style communication)
-c is the desired node MPI grid
-g is the global MPI grid

This reduced the FOM *Iteration Time* from 9,776 secs to 7,974 secs. The problem size/complexity factor for the CI computation with the 8,800 atoms run on Trinity and 1,600 atoms run on Cielo is 166.375 and the run time ratio (with the Cielo baseline Iteration Time of 1,663 secs) is 0.208 giving a final CI value of 34.7. The grid_order definition shown above includes hyperthreading: there are 8 MPI ranks (i.e., defining the grid_order flags: "-c 4,2") and 8 OpenMP threads per rank, for a total of 64 cores per node. The grid order flags "-g 68,1108" are thus matched to the optimized Qbox parameter *nrowmax*. The positive effect of MPI grid ordering is shown in Table 2. Hyperthreading also benefitted performance.

Table 2. Qbox FOM using MPI Grid ordering
"grid_order -R -P -c 2,2 -g 34,1108" (without hyperthreading)
"grid_order -R -P -c 4,2 -g 68,1108" (with hyperthreading)

	2400 gold atoms	**8800 gold atoms**
Without grid order	456	9776
With grid order	315	8834
With grid order and hyperthreading	---	7974

4) Phase-1 CI performance summary:

Figure 8 summarizes the measured CI performance for each of the Tri-Lab applications and the average of the three applications. The achieved average CI performance of 14.517 exceeds the target of 4.0 set for Phase-1.

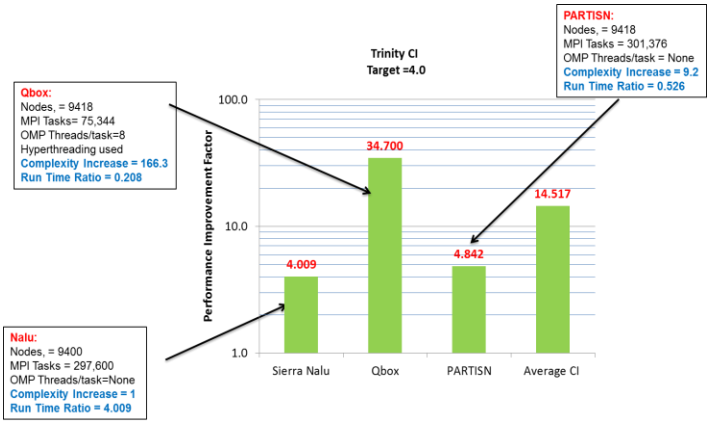

Figure 8. Trinity Capability Improvement performance

B. *SSP Results*

The System Sustained Performance (SSP) benchmark developed by NERSC [8] is useful as a way of measuring, reporting, and projecting the performance of a given system using a set of benchmark programs that represent a workload. SSP is computed as a geometric mean of the performance of eight Tri-Lab and NERSC benchmarks [8]: *miniFE, miniGhost, AMG, UMT, SNAP, miniDFT, GTC and MILC*. A second performance goal for Trinity Phase-1 was a target SSP of 400. This is roughly 8X throughput improvement in performance over the reference baseline measured on NERSC's Hopper (a Cray XE6). Table 3 shows the baseline SSP performance and calculations on Hopper.

Table 3. SSP baseline performance on Hopper

Hopper Nodes	6384					
Hopper SSP						
Application Name	MPI Tasks	Threads	Nodes Used	Reference Tflops	Time (seconds)	Pi
miniFE	49152	1	2048	1065.151	92.4299	0.0056
miniGhost	49152	1	2048	3350.20032	95.97	0.0170
AMG	49152	1	2048	1364.51	151.187	0.0044
UMT	49152	1	2048	18409.4	1514.28	0.0059
SNAP	49152	1	2048	4729.66	1013.1	0.0023
miniDFT	10000	1	417	9180.11	906.24	0.0243
GTC	19200	1	800	19911.348	2286.822	0.0109
MILC	24576	1	1024	15036.5	1124.802	0.0131
					Geom. Mean=	0.0082
					SSP=	52.1212

The Reference Tflops in Table 3, measured on Hopper is to be used for the calculation of SSP on Trinity. The other specified factor in the SSP runs on Trinity is the input problem size for each of the benchmarks, provided with the benchmark tar file and labelled "large" [8]. However the benchmark dos not specify the

number of MPI tasks, the threads per task, nor the number of nodes used, which is a potential shortcoming. The SSP performance measured on Trinity is shown in Table 4. The last column (pi), a measure of throughput per node (Teraflops/second-per-node) is measured by dividing the reference TFLOPS by the product of the run time and number of nodes used. The geometric mean of the eight benchmarks is calculated as shown and the SSP metric is obtained as a product of this number and the number of nodes on Trinity, 9436. NERSC and ACES are improving the usefulness of this metric to better capture the true intent to gauge architectural improvement of proposed future systems by eliminating few shortcomings such as lower incentive to achieve good strong scaling and difficulty in measuring accurate FLOP counts. The achieved performance on Trinity was 500 and it exceeded the target of 400 set for phase-1.

Table 4. SSP performance on Trinity

Trinity Nodes	9436					
Trinity SSP						pi: Rate(TF/s per Node)
Application Name	MPI Tasks	Threads	Nodes Used	Reference Tflops	Time (seconds)	Pi
miniFE	49152	1	1536	1065.151	49.5116	0.0140
miniGhost	49152	1	1536	3350.20032	1.77E+01	0.1229
AMG	49152	1	1536	1364.51	66.233779	0.0134
UMT	49184	1	1537	18409.4	454.057	0.0264
SNAP	12288	2	768	4729.66	1.77E+02	0.0348
miniDFT	2016	1	63	9180.11	377.77	0.3857
GTC	19200	1	300	19911.348	868.439	0.0764
MILC	12288	1	384	15036.5	393.597	0.0995
					Geom. Mean=	0.0530
					SSP=	500.0177

C. *Extra-Large mini app performance*

An additional requirement for Trinity in the SOW, was to measure and document the scaling and performance of five of the eight mini-apps used in the SSP benchmark (*miniFE, miniGhost, AMG, UMT and SNAP*) using approriate scaled inputs, at near full scale of 9436 nodes. These were added to the acceptance tests to help identify any potential hurdles to applications scaling to the full size of Trinity. All the benchmarks completed successfully.

D. *Micro-benhcmark results*

As part of this effort to gather performance characteristics of Trinity, a number of micro-benchmarks [9] were run. Benchmark performance data from *Pynamic, Ziatest, OMB, SMB, mdtest, IOR, PSNAP* and *mpimemu* have been very useful in providing a deeper understanding of the system and factors affecting performance of applications. This section provides a short summary of a few of the benchmarks run during Trinity acceptance.

1) *HPCG*

The HPCG benchmark was run on Trinity in the fall of 2015. The Intel version 2.4 of the benchmark was used, and no changes were made to the code. The runs were scaled up to 9,419 nodes, using 2 MPI ranks per node and 16 OpenMP threads per rank. Local domain dimensions of 80x160x160 were set using the HPCG command line options --nx, --ny, --nz, and the execution time was set to 4,000 seconds using the HPCG command line option --t. To ensure optimal placement of ranks and threads on the cores, the environment variable KMP_AFFINITY was set to 'compact' and the aprun option '-cc depth' was used. The best GFLOP/s rating reported was 182,562.

2) *Ziatest*

This test executes a new proposed standard benchmark method for MPI startup that is intended to provide a realistic assessment of both launch and wireup requirements. Accordingly, it exercises both the launch system of the environment and the interconnect subsystem in a specified pattern. Details on how the test is designed and tar file with the benchmark can be obtained from [9]. *Ziatest* was run on Trinity on 9,334 nodes and it measured a launch time of 12 seconds with 32 MPI tasks per node.

3) *Mpimemu*

Benchmark *mpimemu* helps measure approximate MPI library memory usage as a function of scale. It takes samples of /proc/meminfo (node level) and /proc/self/status (process level) and outputs the min, max and avg values for a specified period of time. More information is provided by NERSC [9].

mpimemu was run on Trinity and Table 5 shows the MPI library memory used with 64 MPI tasks-per-node (using the -j 2 option of *aprun*) as a function of scale. For smaller scales the memory used was found to be less than 2% of the available 128 GB per node.

Table 5. *mpimemu* benchmark results

Trinity number of nodes	1024	2048	4096	9344
Avg. node memory used (GB)	2.6	3.5	5.2	8.9

4) *PSNAP*

PSNAP is a System Noise Activity Program from the Performance and Architecture Laboratory at Los Alamos National Laboratory. It consists of a spin loop that is calibrated to take a given amount of time (typically 1 ms). This loop is repeated for a number of iterations. The actual time each iteration takes is recorded. Analysis of those times allows one to quantify operating system interference or noise. Details on how the test is designed and tar file with the benchmark can be obtained from NERSC [9].

PSNAP was intended to run on the entire system and was executed on all the available Trinity nodes on a run using 9,436 nodes, with 32 MPI tasks per node. It was run after the module '*atp*' was unloaded and the environment variable ATP_ENABLED was unset. The output was processed using the script *psnap_reduce* provided with the benchmark to obtain a histogram of the actual time taken to run the timing loop for each MPI task. The resulting histogram showed acceptable results of OS jitter. A summary of the noise characteristics obtained from the run output with the provided "*psnap_reduce*" script showed:

NR: 9436
Average Slowdown: 0.148236
Min Slowdown: 0.131174
Max Slowdown: 0.178586

To summarize, the maximum percentage slowdown at a core was measured to be 0.178586%.

5) STREAM

STREAM is a simple, synthetic benchmark designed to measure sustainable memory bandwidth (in MB/s) and a corresponding computation rate for four simple vector kernels. The version used for the Trinity benchmark is the OpenMP enabled version of STREAM and can be downloaded from [9]. It was built with the Intel compiler options:

-O2 –xAVX -static -openmp -opt-streaming-stores always

The *Array size* was set to 3,435,973 which correspond to 78 GB of total memory required. It was run on a Trinity node with:

aprun -j 1 -n1 -cc none -d 32 ../stream_c.exe

The measured STREAM performance is shown in Table 6.

Table 6. STREAM benchmark results

Function	Copy	Scale	Add	Triad
Rate (MB/s)	108,014	108,653	118,850	119,077

6) OSU MPI Message Benchmarks

The OSU MicroBenchmark suite is a collection of independent MPI message passing performance microbenchmarks developed and written at the Ohio State University. It includes traditional benchmarks and performance measures such as latency, bandwidth and message rate.

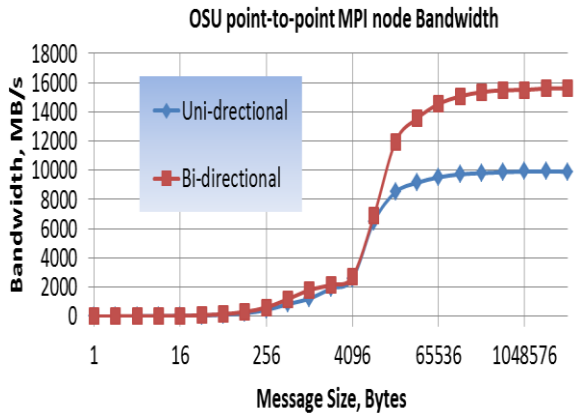

Figure 9. OMB node-to-node MPI bandwidth

Pont-to-point bandwidth, latency and message rate benchmarks were run. Figure 9 shows the uni-directional and bi-directional bandwidth between a pair of tasks on two nodes (node numbers: 2,336 and 2,464) and Figure 10 shows the MPI collective Allreduce latency as function of message size on a run using 300,480 MPI tasks on 9,390 nodes. This shows that on the full system frequently used 8 byte MPI_Allreduce completes in 28 microseconds.

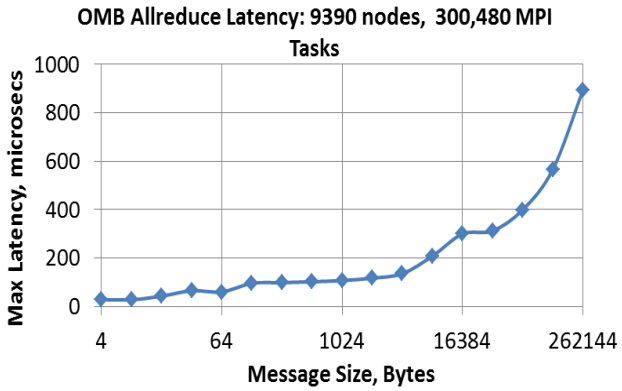

Figure 10. MPI_Allreduce Latency on 9,390 nodes

The OSU multi-latency benchmark showed that inter-node latency for small messages was less than 2 microseconds.

The omb_mbw_mr, message rate benchmark performance between two nodes has been of interest to us because, small message size messaging-rate impacts scalability and performance of our implicit codes with multi-level solvers. Figure 11 helps explain the better scaling seen on many of our applications on Trinity when compared to our commodity InfiniBand clusters.

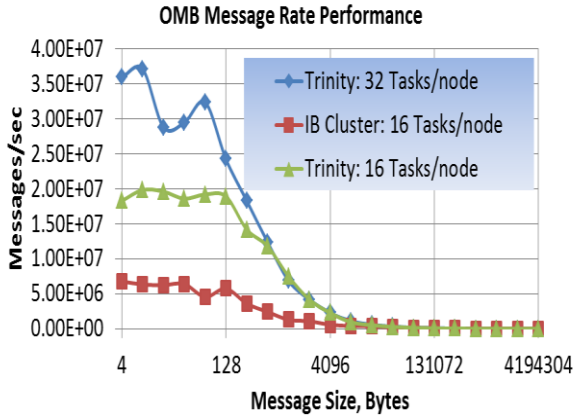

Figure 11. omb_mbw_mr message rate performance

IV. Conclusions

A wealth of data on the performance of a large Cray XC40 has been collected as part of the Trinity procurement and acceptance tests. This paper documents the many man-months of effort to run and optimize the benchmarks. Many of these runs are the first time that these benchmarks have been run at scales using in excess of 300,000 cores. We also compared Trinity performance to our current capability system Cielo. Our investigation confirms that Cray XC40 has good scalability and as expected should give us the needed performance and throughput gain for the Tri-Lab's growing simulation needs. Based on benchmark results we anticipate production Trinity applications would see a performance gain of 2x to 6x over Cielo

depending upon potential gains from AVX2, use of threads and optimal MPI task mapping. The use of Cray's *grid_order* utility and *hugepages* should be explored as codes are ported to Trinity. We are hopeful that the lessons learned from this exercise are helpful to our users as Trinity begins to fully support production applications in early 2016.

ACKNOWLEDGMENT

This work was supported in part by the U.S. Department of Energy. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States National Nuclear Security Administration and the Department of Energy under contract DE-AC04-94AL85000. We thank all the Cray, LANL and SNL staff for their invaluable support during Trinity acceptance tests.

REFERENCES

[1] https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement

[2] D. Doerfler, M. Rajan, C. Nuss, C.Wright, and T.Spelce, "Application-Driven Acceptance of Cielo, an XE6 Petascale Capability Platform," CUG 2011, May 23-26 2011, Fairbanks, Alaska

[3] M. Rajan, C.T. Vaughan, D.W. Doerfler, R.F. Barrett, P.T. Lin, K.T.Pedretti, and K.S. Hemmert. "Application-driven Analysis of Two Generations of Capability Computing Platforms: Purple and Cielo," Computation and Concurrency: Practice and Experience, 2012.

[4] http://www.sandia.gov/asc/integrated_codes.html

[5] https://github.com/spdomin/Nalu

[6] Randal S. Baker and Kenneth R. Koch,"An Sn Algorithm for the Massively Parallel CM---200 Computer", Nucl. Sci. and Eng., Vol. 128, pp. 313-320 (1998)

[7] http://qboxcode.org.

[8] https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/ssp/

[9] https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/