Performance on Trinity Phase 2 (a Cray XC40 utilizing Intel Xeon Phi processors) with Acceptance Applications and Benchmarks

A. M. Agelastos*, M. Rajan*, N. Wichmann§, R. Baker†, S. Domino*, E. W. Draeger‡,
S. Anderson§, J. Balma§, S. Behling§, M. Berry§, P. Carrier§, M. Davis§,
K. McMahon§, D. Sandness§, K. Thomas§, S. Warren§, and T. Zhu§
* *Sandia National Laboratories, Albuquerque, NM*
† *Los Alamos National Laboratory, Los Alamos, NM*
‡ *Lawrence Livermore National Laboratory, Livermore, CA*
§ *Cray, Inc., St. Paul, MN*

***Abstract*—Trinity is the first Advanced Technology System (ATS-1) from the Advanced Simulation and Computing (ASC) Program, which supports the Dept. of Energy's (DOE) National Nuclear Security Administration (NNSA), and is designed to provide the throughput required for the Nuclear Security Enterprise. Trinity Phase 1, currently in production use, has 9,436 dual-socket Haswell nodes while Phase 2, the focus of this paper, has 9,984 Intel Knights Landing Xeon Phi nodes. This paper documents our experiences with the Phase 2 performance acceptance tests which include the Capability Improvement applications, Sustained System Performance benchmarks, and several micro-benchmarks. Lessons learned to achieve optimum performance include which Knights Landing modes to utilize depending upon the application's ability to support the memory hierarchy present, and that each team should investigate huge pages, MPI and thread task mapping, and static linking, which are all commonly considered for extreme scale. Moreover, MPI everywhere scales well for this platform depending upon the application's communication pattern, and optimal performance typically requires making good use of all the available cores on Knights Landing. The studies discussed herein provided us with the needed information to guide optimal use for upcoming production simulations beginning in the summer of 2017.**

***Keywords*-Cray XC40, AVX512, MPI, OpenMP, performance optimization, MIC**

I. Introduction

Trinity, the first of NNSA's Advanced Technology Systems (ATS-1), is a large Cray XC40 with approximately 20,000 compute nodes. This system was procured in the following 2 "phases."

- Phase 1 This portion of Trinity contains 9,436 dual-socket Intel Haswell (HSW) nodes and is currently in production use.
- Phase 2 This portion of Trinity contains 9,984 Intel Knights Landing (KNL) Xeon Phi nodes.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This unclassified document is approved for unlimited release (SAND2017-4888C).

Formal acceptance of Trinity Phase 1 was concluded in December of 2015. Phase 2 of the Trinity procurement was concluded with formal acceptance in December 2016. The Trinity architecture introduces new challenges to the code developers and analysts; these include the transition from multi-core to many-core, deeper memory hierarchies including high-bandwidth memory (HBM) with MCDRAM on the KNL nodes and wider SIMD/vector units. Additionally, we have, for the first time on a large production capability system, high-speed, solid-state Burst Buffer storage. The Trinity performance assessment for Phase 2 Acceptance leverages the following three tiers of application complexity:

1) production apps with one selected from each member of the Tri-Labs (i.e., Lawrence Livermore National Laboratory (LLNL), Los Alamos National Laboratory (LANL), and Sandia National Laboratories (SNL)), referred to as Capability Improvement (CI) apps
2) National Energy Research Scientific Computing Center's (NERSC) Sustained System Performance (SSP) benchmark suite which consists of some full and miniature applications that incorporate relevant kernels from a number of the Tri-Labs' and NERSC's full apps
3) micro-benchmarks

These applications and benchmarks cover a wide range of Tri-Lab- & NERSC-relevant algorithms, programming models, and use cases. Ultimately, the CI metric for the 3 selected production apps quantify the performance at near-full scale and the SSP metric provides a measure of the throughput gain over the Alliance for Computing at Extreme Scale's (ACES) previous-generation capability system, Cielo. Both the CI and SSP metrics, benchmarked on Trinity Phase 2, exceeded their targets and met the requirements for formally "accepting" the system. This paper compliments our CUG paper last year [1] documenting the same set of metrics used for acceptance of Trinity Phase 1. An overview of the Trinity and NERSC-8 performance acceptance benchmarks and code descriptions can be found at Reference [2].

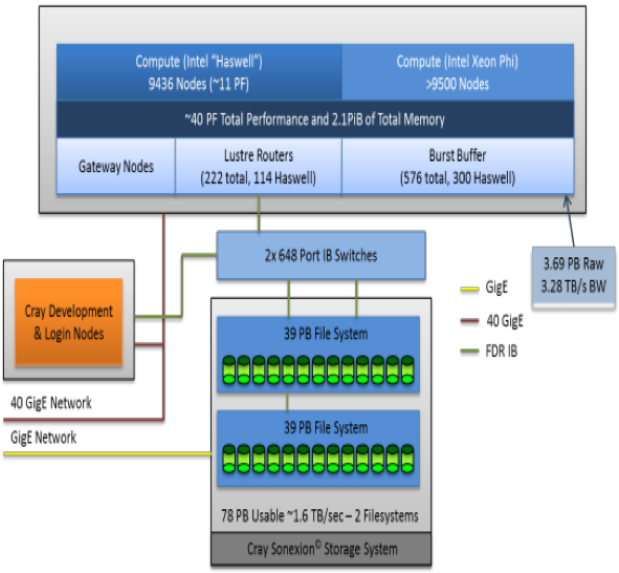

Figure 1. High-level diagram of Trinity's architecture.

Table I
TRINITY ARCHITECTURAL PARAMETERS.

Feature	Phase 1	Phase 2
Total nodes	9,436	9,984
Total cores	301,952	678,912
Cores per node	32	68
Processor	Intel Haswell	Intel KNL
Clock speed	2.3 GHz	1.4 GHz
Peak node GFLOPs	1,177.6	3,046.4
DDR4 clock speed	2,133 MHz	2,400 MHz
DDR4 per core	4.0 GB	1.41 GB
Channels per socket	4	6
Processor cache		
L1	16×32 KB	68×32 KB
L2	16×256 KB	$34 \times 1,024$ KB
L3	40 MB	16 GB MCDRAM (if in *cache* mode)
Interconnect topology	Aries Dragonfly	

II. TRINITY ARCHITECTURE

The Trinity architecture is shown in Figure 1. Table I provides a comparison of some performance-related architectural parameters of Trinity Phase 1 and Phase 2; the reader is referred to [1] for a similar comparison to Cielo.

The Phase 1 HSW partition has 9,436 nodes with dual-socket Intel Xeon ES-2698 v3 running at 2.3 GHz. Each processor has 16 cores and 4 memory channels connected to four 16 GB DDR4 DIMMS clocked at 2.133GHz. The processors are set up to support simultaneous multithreading (SMT) and Intel Turbo Boost and the operating clock frequency varies with the processor load.

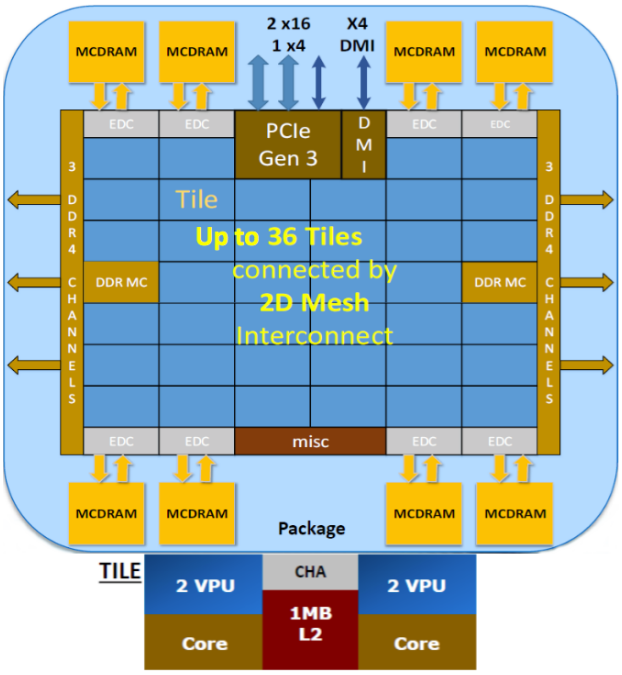

Figure 2. Diagram of Trinity's KNL node architecture.

Assuming a nominal 2.3 GHz operation, the peak node double precision (DP) performance is 32 cores $\times$ 16 FLOPs/cycle $\times$ 2.3 GHz $= 1,177.6$ 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.

The Phase 2 KNL partition has 9,984 nodes with single-socket Intel Xeon Phi running at 1.4 GHz. Each processor has 68 cores arranged in 34 tiles (2 tiles are deactivated) interconnected by a mesh interconnect as shown in Figure 2. Each tile has 2 cores, each with two AVX-512 VPU units and the cores share a 1 MB L2 cache. Each processor has 96 GB of DDR4 memory using 6 memory channels clocked at 2.4 GHz and 16 GB of MCDRAM (high-bandwidth memory, HBM) that has 4 to 5 times the bandwidth of the DDR4 at the cost of a small increase in latency. The physical core supports 4 hardware (SMT) threads and the Intel Turbo Boost technology enables operating clock frequency to vary with the processor load. Assuming a nominal 1.4 GHz operation, the peak node DP performance is 68 cores $\times$ 32 FLOPs/cycle $\times$ 1.4 GHz $=$ 3,046.4 GFLOPS/node. Each core is capable of 32 DP FLOPs per cycle from the two 512-bit AVX-512 units.

This processor can be booted into multiple memory modes, i.e., *flat*, *cache*, and *hybrid* (*flat+cache*), and multiple sub-NUMA cluster (SNC) modes, i.e., *hemisphere*/*quadrant*, *all2all*, and *snc2*/*snc4*. Permutations with these memory and cluster modes result in $\sim$20 different available modes. The modes that resulted in optimal Acceptance application

performance utilized quadrant cluster mode (affinity between CHA and memory channel) and the flat (MCDRAM and DDR as different NUMA nodes) and cache (MCDRAM used as direct-mapped cache) memory modes; these are referred to as *quad/flat* and *quad/cache*, respectively.

Cache thrashing can occur when the MCDRAM is configured as a direct-mapped cache. More recent kernel modification (i.e., Zone Sort) reduce the observed thrashing. Running any application sensitive to KNL direct mapped cache thrashing immediately after the compute nodes are rebooted significantly reduces these issues.

III. Acceptance Tests' Performance Results

The "Acceptance" period for Phase 2 encompassed the bulk of 2016. During this time frame, the operating environments of Phase 2 and Trinity's test beds (i.e., Mutrino, Trinitite) were evolving to better handle KNL. The bulk of the results presented herein were generated under an UP01 or UP02 operating environment; refer to [3] for more information regarding these environments.

Acceptance of Phase 2 ended up occurring while on a "restricted" network. Approximately 1,068 Phase 2 nodes were merged with Phase 1 which left approximately 8,907 Phase 2 nodes for testing. The benchmarks discussed herein were all tested at as high of scale of possible up to 8,907 nodes. Despite not having access to the "full" ultimate Phase 2 system, all benchmarks surpassed their Phase 2 targets and a lot of information was learned about how to develop for and utilize KNL.

Application developers need to invest resources into investigating not just what memory mode best suits their application but also what the optimal rank and thread layout would be for maximum performance. The methodologies for doing this are platform and compiler specific. Some of the oft-used methods for acceptance are described below since they will be referenced within forthcoming sub-sections.

The primary method of launching MPI applications on Trinity during the acceptance period was by using Cray's `aprun` command. This utility's `--cpu-binding`, or `-cc` as the short version, command line argument provides the user a lot of options for how the ranks and threads are bound on a KNL node. A common option used for many of the acceptance cases was `depth`. This enables the parent and threads to float around on cores or hardware threads that are relatively close together depending on the depth requested (i.e., the `--cpus-per-pe`/`-d` command line option). For example, if `-d2` and `-j2` (i.e., this is shortened version of `--cpus-per-cu` which defines how many CPUs per compute unit) options are set, then both the parent and its thread will run on the same core on different hardware threads, however if `-d4` and `-j2` are set, then the parent and threads will be confined to the 2 cores of a tile with each core running 2 hyperthreads.

Sometimes the `-cc` option was not sufficient for some of the more custom arrangements desired. Cray systems support setting the `MPICH_RANK_REORDER_METHOD` environment variable which provides some additional MPI rank re-ordering methods that can be used to augment the selected `-cc` option selected. The `MPICH_RANK_REORDER_METHOD` settings, and their descriptions (directly extracted from the `intro_mpi` manpage), used by the Acceptance Team are:

- `0` "Specifies round-robin placement. Sequential MPI ranks are placed on the next node in the list. When every node has been used, the rank placement starts over again with the first node."
- `1` "Specifies SMP-style placement. This is the *default* `aprun` placement. For a multi-core node, sequential MPI ranks are placed on the same node."
- `3` "Specifies a custom rank placement defined in the file `MPICH_RANK_ORDER`. The `MPICH_RANK_ORDER` file must be readable by the first rank of the program, and reside in the current running directory. The order in which the ranks are listed in the file determines which ranks are placed closest to each other, starting with the first node in the list."

A Cray utility to help generate a `MPICH_RANK_ORDER` file is `grid_order`. The `grid_order` parameters, and their descriptions, used by the Acceptance Team are:

- `-R` is row-major ordering of ranks.
- `-P` is a Peano space filling curve (optimal for FFT-style communication) that lists successive rows in alternating order.
- `-Z` lists successive rows of cells in the same order, jumping from the end of one row to the beginning of the next.
- `-c` is the desired node MPI grid.
- `-g` is the global MPI grid.
- `-n` is the number of ranks per line.
- `-m` is the max. rank count.

A. ASC Application Capability Improvement (CI)

A key figure used to gauge performance at near full scale is the Capability Improvement metric [4], which is computed as an average improvement in performance over Cielo (Cray XE6), of three ASC applications: PARTISN (from LANL), Nalu (from SNL), and Qbox (from LLNL). The baseline performance data was collected by using more than $2/3$ of the compute partition from our previous generation ASC ACES platform Cielo in 2013. The CI metric is defined as

$$\underbrace{\text{CI Metric}}_{CI} = \underbrace{\left(\begin{array}{c}\text{Problem Size/}\\ \text{Complexity}\\ \text{Increase}\end{array}\right)}_{\chi} \times \underbrace{\left(\begin{array}{c}\text{Runtime}\\ \text{Speedup}\end{array}\right)}_{\tau} \quad (1)$$

Trinity's target performance for the CI metric is $8\times$ over the baseline Cielo performance. The Phase 2 Acceptance

process specified that the average CI of these applications must be $4\times$ the Cielo benchmark. Such a metric was also used in the acceptance benchmarks of Phase 1 and our previous-generation ASC ACES capability platform, Cielo [1][5].

A description of the 3 applications picked for the CI benchmark, their performance measured on Phase 1, and comparisons to Cielo are available in Ref [1] and is not repeated here in detail. The Phase 2 aggregate CI, computed as the arithmetic mean of the individual CI metrics, is 9.37, which exceeds the target 4.0. Each of the three CI applications is discussed below.

1) SNL's Nalu: Nalu is a low Mach computational fluid dynamics (CFD) code built atop the Sierra Toolkit and Trilinos solver Tpetra stack. Its capabilities are discussed within [6] and it can be downloaded from [7]. Nalu version 1.01 was used for the CI simulations for Phase 2.

Problem Description: The problem of interest, which is also a lightly modified version of "milestoneRun" within Nalu's regression test suite [8], is a turbulent open jet (Reynolds number of $\sim 50,000$) with passive mixture fraction transport using the one equation Ksgs large eddy simulation (LES) model. The problem is discretized on an unstructured mesh with hexahedral elements. The baseline problem "R6" mesh, created with 6 consecutive uniform mesh refinement steps, consists of 9 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 $\sim$27 billion rows (momentum) followed by a series of smaller $\sim$9 billion row systems for the continuity system (elliptic Pressure Poisson), mixture fraction, and turbulent kinetic energy.

Figure of Merit (FOM) Description: There are 2 FOMs used for Nalu's CI Metric; 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. Nalu's runtime speedup is defined as

$$\begin{aligned}\begin{pmatrix}\text{Runtime}\\ \text{Speedup}\end{pmatrix} &= \begin{pmatrix}\text{Speedup From}\\ \text{Momentum Solve}\end{pmatrix} \times 0.67 \\ &\quad + \begin{pmatrix}\text{Speedup From}\\ \text{Momentum Assemble}\end{pmatrix} \times 0.33 \\ &\Leftrightarrow \\ \tau &= \tau_S \times 0.67 + \tau_A \times 0.33 \end{aligned} \tag{2}$$

The Nalu CI simulation performs 25 time steps. Each time step performs 2 nonlinear iterations and then a variable number of linear iterations until tolerances are achieved. The momentum assemble occurs for every nonlinear iteration, i.e., a total of $25 \times 2 = 50$ times for the CI simulation, whereas momentum solve occurs for every linear iteration, i.e., a total of $25 \times 2 \times$ (no. avg. linear iter.) times. Nalu's FOM and problem description are discussed in detail within [9].

Table II
THIS TABLE PROVIDES KEY NALU CI PARAMETERS.

Parameter	**Cielo**	**Phase 1**	**Phase 2**
No. Nodes	8,192	9,420	4,096
No. MPI Ranks per Node	16	32	64
No. Threads per Rank	1	1	1
Mesh Refinement Level, R	6	6	6
Avg. Momentum Equation Assemble Time per Loop, t_A **(sec.)**	2.712	0.637	1.231
Avg. Momentum Equation Solve Time per Loop, t_S **(sec.)**	0.502	0.129	0.356
Complexity Increase, χ $\left(R_{\textbf{Trinity}} - R_{\textbf{Cielo}}\right) \times 8$		1.00	1.00
Runtime Speedup, τ $\left(\frac{t_{S,\textbf{Cielo}}}{t_{S,\textbf{Trinity}}}\right) \times 0.67 + \left(\frac{t_{A,\textbf{Cielo}}}{t_{A,\textbf{Trinity}}}\right) \times 0.33$		4.01	1.67
Capability Improvement, CI $\chi \times \tau$		*4.01*	*1.67*

Capability Improvement Metric Run: Nalu's formal CI value is 1.67 and was obtained in October 2016 running the same size problem as the Cielo baseline, i.e., the "R6" mesh which has no complexity increase, on 4,096 nodes with 64 MPI ranks per node in *quad/cache* mode. All of that improvement accrues from the faster run time measured for the average assemble time for the momentum equation (measured value was 113.532 sec.) and the average solve time for the momentum equation (measured value was 168.617 sec.). Strong scaling the same problem down to 2,048 nodes results in a CI value of 0.82. This exhibits better than linear scaling and, if these trends hold which they are expected to, Nalu was expected to achieve a CI value greater than 4.0 at the full Phase 2 node count. Table II contains the CI-relevant parameters for Nalu and compares them for the Cielo, Phase 1, and Phase 2 systems.

The resultant CI falls short of the desired 4.0 because there are out-of-memory (OOM) errors when run in *quad/cache* at the 8,192-node scale that are preventing Nalu from achieving its target. This issue is described in the following section.

Memory Debugging: Preliminary memory profiling of Nalu using its built-in memory statistics, adding custom ones, and leveraging Cray's MPICH Memory Report (accessed by setting `MPICH_MEMORY_REPORT` environment variable to `1`) indicated that MPI had more memory allocated to it than expected. This finding motivated 2 independent studies. The first study was to better understand Nalu's communication pattern. The second study was to quantify the memory needs of large-scale MPI applications that have a lot of MPI point-to-point (p2p) connections, e.g., `MPI_Send()`/`MPI_Recv()`.

Table III
NALU CONNECTIVITY STATISTICS.

Metric	*No. MPI ranks w/ 64 ranks/node* 1,024	8,192
Rank w/ max. p2p connections ($r_{\max}$)	14	13
Max. p2p connections	1,023	7,671
Min. p2p connections	120	103
Mean p2p connections	307.7	463.4
Degree p2p-connected	30.0%	5.7%
Mean p2p connections for rank $\in [r_{\max}-1, r_{\max}+1]$	1,021.3	7,595.3
Mean p2p connections for rank $\in [r_{\max}-2, r_{\max}+2]$	859.6	7,472.6
Mean p2p connections for rank $\in [r_{\max}-4, r_{\max}+4]$	734.4	5,820.8
Mean p2p connections for rank $\in [r_{\max}-8, r_{\max}+8]$	663.3	4,660.1
Mean p2p connections for rank $\in [0, r_{\max}+16]$	569.8	3,717.0
Mean p2p connections for rank $\in [0, r_{\max}+32]$	498.0	3,097.2
Mean p2p connections for rank $\in [0, r_{\max}+64]$	457.8	2,586.7

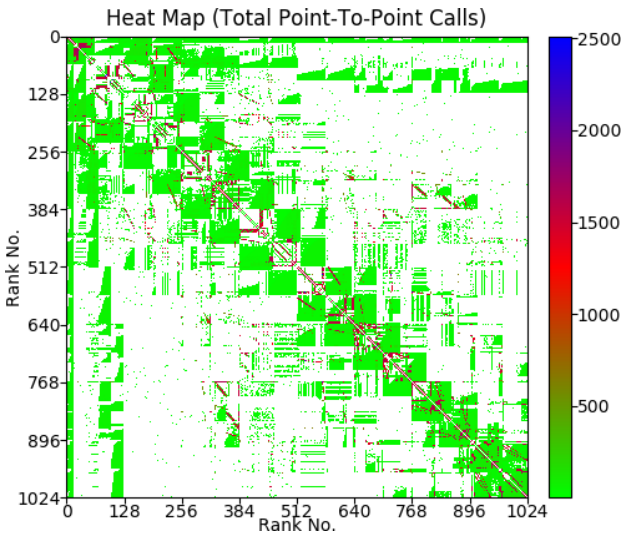

Figure 3. Heat map of total number of point-to-point (p2p) calls for "R3" Nalu simulation initialization and a single time step on 1,024 MPI ranks (16 nodes with 64 ranks per node).

CrayPat was utilized to instrument the MPI functions and gather statistics. Custom software was written to extract information from CrayPat's resultant AP2 file and generate the desired statistics and connection heat maps. Data was collected for the "R3" mesh utilizing 1,024 MPI ranks (16 nodes with 64 ranks/node) and for the "R4" mesh utilizing 8,192 MPI ranks (128 nodes with 64 ranks/node); these mesh and rank count sets are weak scaled from the "R6" mesh utilizing 524,288 MPI ranks (8,192 nodes with 64 ranks/node). Connection statistics for the "R3" and "R4" simulations are provided within Table III while the connection heat map for "R3" is depicted in Figure 3.

Table IV
THIS TABLE PROVIDES CRAY MPI MEMORY UTILIZATION LEVELS ON TRINITY. PHASE 2 COMPUTE NODES HAVE 2.5 GB/NODE OF MEMORY IN USE WITHOUT ANY HPC APPLICATIONS RUNNING.

Test Description	*No. MPI ranks across 8,192 nodes* **524,288**	**393,216**	**262,144**	**131,072**
Test 1: Full p2p connectivity	64.4 GB/node	36.5 GB/node	16.5 GB/node	4.3 GB/node
Test 2: Full p2p connectivity w/ 1 MPI mbox. alloc.	60.4 GB/node	34.3 GB/node	15.5 GB/node	4.1 GB/node
Test 3: Full connectivity w/ `MPI_Alltoall()`	6.7 GB/node	4.5 GB/node	2.3 GB/node	0.8 GB/node
Test 4: 1,024 p2p connections	6.2 GB/node	3.9 GB/node	2.0 GB/node	0.7 GB/node

Table III provides the MPI rank number for the simulation that has the most p2p connections, r_{max}. It also provides the maximum, minimum, and mean numbers of p2p connections across that simulation's ranks. Finally, it provides the average number of p2p connections that r_{max} and some number of its neighbors have; this ascertains how densely connected this rank and its neighbors may be. If r_{max} and many of its neighbors have a large number of p2p connections, then the node they fall upon will have to allocate a lot of buffers.

Table III indicates that some ranks tend to be connected in a point-to-point fashion to a majority of the MPI ranks. Specifically, rank 14 for the "R3" and rank 13 for the "R4" are connected to 1,023 and 7,671 ranks, respectively. This table also indicates that the neighbors of the ranks that are heavily connected are, also, heavily connected. These ranks are also represented by the horizontal block of green at the top of the heat map in Figure 3.

The utility `mpi_memused`, which uses the same method as NERSC's `mpimemu` benchmark to determine node-level memory usage, was used for the following 4 tests.

- Test 1 This test issues `MPI_Isend()` and `MPI_Recv()` between all ranks in the job.
- Test 2 This is the same as Test 1 but with setting the `MPICH_GNI_MBOXES_PER_BLOCK` environment variable to the number of connections being made. Cray MPI typically allocates GNI mailboxes in chunks which has some wasted memory at the end of each chunk. Setting this environment variable allocates all mailboxes in one

large chunk, which eliminates the aforementioned wasted memory.

Test 3 This test uses a `MPI_COMM_WORLD` `MPI_Alltoall()` collective call to exchange unique data between all ranks.

Test 4 This test is the same as Test 1 but with each rank only being connected to 1,024 other ranks.

The results from these tests are provided within Table IV and indicate that a large amount of memory is required for MPI buffers if ranks are heavily connected in a p2p fashion. In cases such as this, we recommend the use of collectives such as `MPI_Alltoall` to exchange the required data.

These 2 studies indicate that Nalu has contiguous ranks that are point-to-point connected to a majority of all ranks *and* these ranks will have a large amount of memory on the node allocated to MPI buffers. This is the likely cause of the OOM errors encountered. Research and development is ongoing to alleviate this issue.

2) LANL's PARTISN: PARTISN is a particle transport code that provides neutron transport solutions on orthogonal meshes in one, two, and three dimensions [10]. PARTISN uses a multi-group energy treatment in conjunction with the Sn angular approximation. PARTISN supports time-dependent calculations and its primary components of computation involved KBA sweeps and associated zero-dimensional physics. The KBA sweep wave-front algorithm provides two dimensional parallelism for three dimensional geometries and is tightly coupled by nearest-neighbor communications. PARTISN version 8.23 was used for the Phase 2 CI simulation.

Problem Description: The test problem is `MIC_SN`, i.e., MIC with group-dependent Sn quadrature. Its description from [1] is provided below.

> 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 (i.e., `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.

Figure of Merit (FOM) Description: PARTISN's FOM is its "Solver Iteration Time." Ideally, this FOM should remain constant for weak scaling. This parameter is directly extracted from its simulation output.

Capability Improvement Metric Run: PARTISN's final CI value is 5.68 and was obtained in December 2016. This CI result was from running PARTISN on a problem $9\times$ larger than the Cielo baseline with a $\sim 59\%$ increase in solver iteration time. The Cielo baseline utilized 2,880 zones per core with 4 OpenMP threads per MPI rank on 8,192 nodes with 4 MPI ranks per node for a total of 131,072 processing elements. The Phase 2 formal CI simulation utilized 6,480 zones per core with 2 OpenMP threads per MPI rank and 32 MPI ranks per node for a total of 524,288 processing elements. These parameters and their associated CI calculation parameters are provided within Table V.

Table V
THIS TABLE PROVIDES KEY PARTISN CI PARAMETERS.

Parameter	**Cielo**	**Phase 1**	**Phase 2**
No. Nodes	8,192	9,418	8,192
No. MPI Ranks per Node	4	32	32
No. Threads per Rank	4	1	2
No. Cores, n	131,072	301,376	524,288
Zones per Core, z	2,880	11,520	6,480
Solver Iteration Time, t **(sec.)**	209.400	397.710	332.047
Complexity Increase, χ $(n \times z)_{\textbf{Trinity}}/(n \times z)_{\textbf{Cielo}}$		9.19	9.00
Runtime Speedup, τ $t_{\textbf{Cielo}}/t_{\textbf{Trinity}}$		0.526	0.631
Capability Improvement, CI $\chi \times \tau$		*4.83*	*5.68*

This performance was achieved by leveraging 8 MB huge pages (i.e., `module load craype-hugepages8M`), rank reordering (see below for methodology used), hybrid paralellism (i.e., 32 MPI Ranks/node $\times$ 2 OpenMP Threads/node reduced simulation time by $\sim 2\%$ from pure MPI), and changing PARTISN's `nchunk` parameter to 16 (i.e., this reduced simulation time by $\sim 28\%$). All commands provided henceforth will be in BASH.

```
export MPICH_RANK_REORDER_METHOD=3
grid_order -R -Z -m 262144 -n 32 \
    -g 512x512 -c 4,4
```

Additionally, the OpenMP environment settings for the hybrid parallelism model utilized are below.

```
export OMP_WAIT_POLICY=active
export OMP_NUM_THREADS=2
export OMP_PROC_BIND=spread
```

3) LLNL's Qbox: Qbox is a Density Functional Theory (DFT) code used to compute the properties of materials at the atomistic scale. Qbox's primary algorithm uses a Born-Oppenheimer description of atomic cores and electrons with valence electrons treated quantum mechanically using DFT and a plane wave basis. Core electrons and nuclei are described by nonlocal pseudopotentials and are derived to match all-electron, single-atom calculations external to a cut-off radius. Qbox's computational profile consists primarily of dense, parallel linear algebra and three-dimensional, parallel, complex-to-complex Fast Fourier Transforms (FFTs)

Table VI
THIS TABLE PROVIDES KEY QBOX CI PARAMETERS.

Parameter	Cielo	Phase 1	Phase 2
No. Nodes	6,144	9,418	8,504
No. MPI Ranks per Node	16	8	16
No. Threads per Rank	1	8	4
No. Gold Atoms, n	1,600	8,800	6,000
Max. Iteration Time, t **(sec.)**	1,663	7,974	4,227.448
Complexity Increase, χ $\left(n_{\text{Trinity}}/n_{\text{Cielo}}\right)^3$		166.38	52.73
Runtime Speedup, τ $t_{\text{Cielo}}/t_{\text{Trinity}}$		0.208	0.393
Capability Improvement, CI $\chi \times \tau$		*34.7*	*20.74*

[11][12]. Qbox revision 206 (qb@LL-r205 branch) was utilized for the Phase 2 CI simulation.

Problem Description: The test problem is the initial self-consistent wavefunction convergence of a large crystalline gold system. Its description from [1] is provided below.

> 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.

Figure of Merit (FOM) Description: Qbox's FOM is the maximum total wall clock time to run a single self-consistent iteration with three non-self-consistent inner iterations. This value is extracted directly from the `max` key of Qbox's XML `timing` sub-element whose `where` key has `run` as its value and whose `name` key has `iteration` as its value.

Capability Improvement Metric Run: Qbox's final CI value is 20.74 and was obtained in December 2016 running a problem with 6,000 atoms (i.e., a complexity increase of $\sim$53 from the 1,600 atoms used for the Cielo baseline) on 8,504 nodes using 16 ranks per node and 4 threads per rank in *quad/cache* mode. These parameters and their associated CI calculation parameters are provided within Table VI.

This performance was achieved by leveraging 2 MB huge pages (i.e., `module load craype-hugepages2M`), rank reordering which is essential for both Phase 1 and Phase 2 (see below for methodology used), static linking (i.e., provided up to 40% reduction for some simulations, refer to Table VII for more information), and using Cray instead of Intel compilers. Slight modifications to `qb.C` was required to enable static linking; the change was to use `getenv()` instead of `getlogin()`.

```
export MPICH_RANK_REORDER_METHOD=3
```

Table VII
THIS TABLE PROVIDES QBOX SIMULATION TIMINGS FOR DIFFERENT LINKING STYLES; THE RUN CORRESPONDS TO A 256-NODE, 880-ATOM SIMULATION WITH 32 MPI RANKS PER NODE AND 2 OPENMP THREADS PER RANK WITH `NROWMAX` EQUAL TO 256.

Link Type	Walltime
Dynamic	330 sec.
Dynamic w/ statically linked Cray libsci	215 sec. ($\sim$35% reduction)
Static	198 sec. ($\sim$40% reduction)

Table VIII
THIS TABLE COMPARES BUILD AND RUN TIME PARAMETERS FOR THE 3 CI APPLICATIONS.

Parameter	Nalu	PARTISN	Qbox
Code changes	No	No	2 LOC
KNL mode	*quad/cache*	*quad/cache*	*quad/cache*
Language	C++	Fortran 90/95	C++
Compiler used	Intel v. 16.0.3	Intel v. 16.0.3	Cray v. 8.5.2
Linking used	Static	Dynamic	Static
Hugepages	No	8 MB	2 MB
Nodes	4,096	8,192	8,504
MPI ranks/node	64	32	16
Threads/rank	N/A	2	4
Grid ordering	No	Yes	Yes
CI	1.67	5.68	20.74
Average CI	*9.36*		

```
grid_order -R -P -n 16 \
    -g 128,1063 -c 4,4
```

Some early (first half of 2016) comparisons between KNL and Intel Broadwell (BDW) on a small problem (i.e., 256 atoms) with 512 MPI ranks (i.e., 8 KNL nodes, 16 BDW nodes) indicate that the overall subdivision of work within Qbox remains consistent. Additionally, hyperthreading did not increase Qbox's runtime speedup with similar test cases on KNL and Intel Broadwell; hyperthreading caused a $\sim$31% increase in wall time on a 1,280 atom simulation on 768 nodes with 32 ranks per node and 4 OpenMP threads per rank.

4) Phase 2 CI Performance Summary: A high-level comparison of build and run time information for the 3 CI applications is provided within Table VIII. Refer to Tables II, V, and VI for additional, detailed comparisons between Phase 2, Phase 1, and the Cielo baseline reference.

B. Sustained System Performance (SSP) Suite

The SSP suite is useful as a way of assessing the performance of a given system using a set of benchmark programs that represent a workload [13]. SSP is computed as a geometric mean of the performance of the following 8 Tri-Lab and NERSC benchmarks: miniFE, miniGhost, AMG, UMT,

Table IX
SSP BASELINE PERFORMANCE ON HOPPER.

App. / Parameter	MPI Tasks	Threads per Rank	Nodes	Ref. TFLOPs	Time (sec.)	Pi $\left(\frac{\text{TFLOPs}}{\text{sec.} \cdot \text{node}}\right)$
miniFE	49,152	1	2,048	1,065.15	92.43	0.0056
miniGhost	49,152	1	2,048	3,350.20	95.97	0.0170
AMG	49,152	1	2,048	1,364.51	151.19	0.0044
UMT	49,152	1	2,048	18,409.40	1,514.28	0.0059
SNAP	49,152	1	2,048	4,729.66	1,013.10	0.0023
miniDFT	10,000	1	417	9,180.11	906.24	0.0243
GTC	19,200	1	800	19,911.35	2,286.82	0.0109
MILC	24,576	1	1,024	15,036.50	1,124.80	0.0131
Pi Geom. Mean:				0.0082		
# Hopper Nodes:				6,384		
SSP:				***52.1212***		

Table X
SSP PERFORMANCE ON PHASE 2.

App. / Parameter	MPI Tasks	Threads per Rank	Nodes	Ref. TFLOPs	Time (sec.)	Pi $\left(\frac{\text{TFLOPs}}{\text{sec.} \cdot \text{node}}\right)$
miniFE[1]	15,360	32	3,840	1,065.15	7.20	0.0385
miniGhost	24,576	2	768	3,350.20	33.95	0.1285
AMG	49,152	2	768	1,364.51	160.36	0.0111
UMT[2]	49,184	4	769	18,409.40	467.26	0.0512
SNAP	24,576	2	768	4,729.66	212.00	0.0290
miniDFT	3,008	1	47	9,180.11	471.62	0.4142
GTC	9,600	1	150	19,911.35	2,118.73	0.0627
MILC[1]	24,576	1	384	15,036.50	631.27	0.0620
Pi Geom. Mean:				0.0582		
# Phase 2 Nodes:				9,984		
SSP:				***580.92***		
Target SSP:				***489***		

1 Run in *quad/flat* wholly in HBM instead of *quad/cache*.
2 Its first node had 8 threads per rank and 32 MPI ranks.

SNAP, miniDFT, GTC, and MILC. The Phase 2 SSP target is 489. The demonstrated performance on Phase 2 running these benchmarks is 580.92. The baseline SSP parameters from Hopper are provided within Table IX while the Phase 2 SSP parameters are provided within Table X. The following sub-subsections contain SSP application-specific notes and findings. This subsection is concluded with a summary.

1) miniFE: In May, 2016, a decomposition bug within version 1.4 of miniFE was discovered. At this time, it was decided to switch to version 2.0 since it did not exhibit the bug. MiniFE was also one of the applications used for Phase 2 "Factory Testing." Initially, miniFE's performance in *quad/cache* was below expectations. Further analysis concluded that this was due to direct mapped cache thrashing.

2) miniGhost: MiniGhost profiling indicates it is memory bandwidth bound on KNL. Cray streamlined its stencil kernel and modified it to do an `Allreduce` once per timestep of a size equal to the number of variables being reduced rather than once per variable per timestep.

3) AMG: Initial performance profiling revealed two key routines: `hypre_ParCSRRelax_L1()` and `hypre_CSRMatrixMatvec()`. Also, approximately 8.5% of the total wall time is spent within `MPI_Allreduce`. An analysis indicated that performance improvements could result from merging OpenMP parallel regions, reducing the synchronization costs, and the use of dynamic scheduling; these changes were implemented.

4) UMT: Cray's compilers resulted in better UMT performance than Intel's compilers. Moreover, version 15 of the Intel compiler had a Fortran bug that required legal `Size` variable to be renamed. Also, Intel version 16 requires no inlining to get a correct answer for UMT.

Significant optimizations were performed within `snflwxyz.F90`, `snswp3d.F90`, and `snqq.F90`. These are the same optimizations used for Phase 1. Also, a race condition was discovered in the original OpenMP implementation of UMT. Specifically, a memory overwrite race condition was found within subroutine `snreflect()` which is called within an OpenMP parallel region. The frequency of its occurrence depends on the number of OpenMP threads used (e.g., there is no race condition if OpenMP is not used), the optimization level of routines (i.e., more optimization increases likelihood), and the problem size.

It was observed that 2 MPI ranks always required 20-30% more time and memory than all other ranks. Multiple program, multiple data (MPMD) was utilized to greatly reduce load imbalance and total elapsed time which required grid reordering and one node which contained fewer MPI ranks compare to the other nodes, but each of these fewer ranks spawned twice as many OpenMP threads.

5) SNAP: A newer version of SNAP (version 1.07 instead of 1.01) was utilized since it had improved vectorization and better support for hybrid parallelism. Also, this version of SNAP using 2 MB page size is 3-22% faster than when using the default 4 KB page size; the benefit of using a 2 MB page size increases as the node count increases. It was also observed that SNAP is 6-33% faster using grid ordering than without. Moreover, both of these options appear to reduce run-to-run variations.

6) miniDFT: Intel's MKL libraries were used for ScaLAPACK, BLAS, and FFTW. Hyperthreading was evaluated on KNL however no performance improvements were observed. The 2 MB huge page size utilized has a large (i.e., $\sim$200% improvement) impact on miniDFT performance when compared to the default 4 KB page size. Additionally, the command line parameter `ntg` was set to 4 instead of 1.

7) GTC: Initial performance studies of GTC in *quad/cache* had severe performance problems. Ultimately, this was due to direct-mapped cache thrashing. Also,

Table XI
THIS TABLE COMPARES BUILD AND RUN TIME PARAMETERS FOR THE SSP APPLICATIONS.

App.	Compiler Used	Linking Used	Huge pages	Grid Ordering
miniFE	Intel v. 16.0.3	Static	8 MB	No
miniGhost	Intel v. 16.0.3	Static	No	Yes
AMG	Cray v. 8.5.2	Static	8 MB	Yes
UMT	Cray v. 8.5.3	Static	4 MB	Yes
SNAP	Intel v. 16.0.3	Static	2 MB	Yes
miniDFT	Intel v. 16.0.3	Static	2 MB	No
GTC	Intel v. 16.0.3	Static	2 MB	Yes
MILC	Intel v. 16.0.3	Static	8 MB	Yes

`MPICH_RANK_REORDER_METHOD` was set to 0 to "fold" the MPI ranks such that the toroidal communicator is contained within a node to help improve performance.

8) MILC: MILC utilized the newer version of Cray's MPICH library (version 7.4.0) versus the default. Additionally, compiler options within `ks_imp_dyn/Makefile.Intel_knl` and `libraries/Make_vanilla` were changed to improve optimization and eliminiate the overhead of OpenMP for single-threaded runs. These options are provided below; please note that Cray's module environment automatically adds the compiler-specific flag for the targeted platform, e.g., `-xMIC-AVX512` for the options below.

```
OPT = -O3 -fp-model fast=2 -fno-alias \
 -fno-fnalias -qopt-report=5 -no-prec-div
OCFLAGS = -std=c99 \
 -DCRAY_CORRECT_FUNCTIONALITY
LMPI = -Wl,--whole-archive,\
 -ldmapp,--no-whole-archive
```

9) Phase 2 SSP Performance Summary: Table XI compares the compiler used, linking type, huge pages setting, and whether or not grid ordering was used for the SSP applications for their simulations.

C. Mini Apps Run at Extra-Large Scale

Acceptance requires that five out of the eight SSP mini-applications (i.e., miniFE, miniGhost, AMG, UMT, and SNAP) are to be run at near the full scale of Phase 2. These simulations have been performed and their results are provided below.

1) miniFE: It was selected to utilize *quad/cache* for miniFE's "large" simulation as opposed to how it was run for the SSP calculation, which was in *quad/flat*. Figure 4 depicts miniFE weak scaling trend on Phase 2 with *quad/cache*. This problem was weak scaled from the SSP setup. The SSP result at 3,840 nodes in *quad/flat* was also provided for reference.

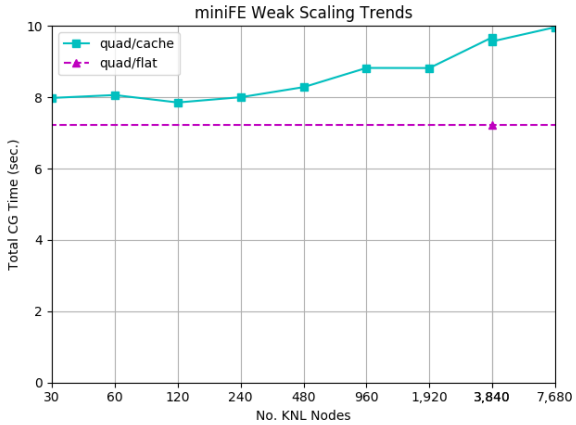

Figure 4. MiniFE weak scaling for *quad/cache* sized from SSP problem; this is a semilog plot where the x axis is logarithmic with a base of 2.

Table XII
THIS TABLE PROVIDES SOME "LARGE" MINIGHOST RESULTS THAT COMPARE TIMINGS WITH AND WITHOUT `GRID_ORDER` USAGE; THESE PROBLEMS CORRESPOND TO 32 MPI RANKS PER NODE AND WITH 2 OPENMP THREADS PER RANK.

No. Nodes	Mem./ Node	Global Grid	Local Grid	MPI Grid	Grid Order? No (sec.)	Grid Order? Yes (sec.)
768	~64GB	$5{,}376^3$	$168\times112\times336$	$32\times48\times16$	38.9	34.0
6,144	~64GB	$10{,}752^3$	$168\times112\times336$	$64\times96\times32$	54.9[1]	36.2
8,820	~72GB	$12{,}600^3$	$210\times150\times225$	$60\times84\times56$	48.5	46.2
1,029	~77GB	$6{,}300^3$	$225\times150\times225$	$28\times42\times28$	47.5	45.2
8,232	~77GB	$12{,}600^3$	$225\times150\times225$	$56\times84\times56$	85.7[1]	47.0
8,820	~87GB	$13{,}440^3$	$224\times160\times240$	$60\times84\times56$	91.7[1]	48.0

1 These values are larger than expected; cache thrashing may be the cause, however this hasn't been confirmed.

2) miniGhost: The SSP problem for miniGhost could only weak scale up to $8 \times 768 = 6{,}144$ nodes, which corresponds to a factor of 2 in each dimension. It was chosen to set a requirement that the global grid must remain cubic for this miniGhost scaling study. Analysis indicates that miniGhost is sensitive to local and global grid decompositions. Additional cubic global grids with larger memory per node were also utilized in order to run larger problems on large node counts. All of this data is within Table XII. Some of the timings were larger than expected. It is theorized that cache thrashing is the culprit, however this hasn't been confirmed yet.

3) AMG: The SSP problem size for AMG is on 768 nodes using 64 ranks per node totaling 49,152 ranks. The x/z and y/z ratios were kept fixed at $2/3$ to enable weak scaling from the SSP problem. OOM errors were encountered at 1,500 nodes and beyond with this methodology. Profiling of AMG indicates that `MPI_Allreduce` has the highest percentage of time than the other MPI functions. The

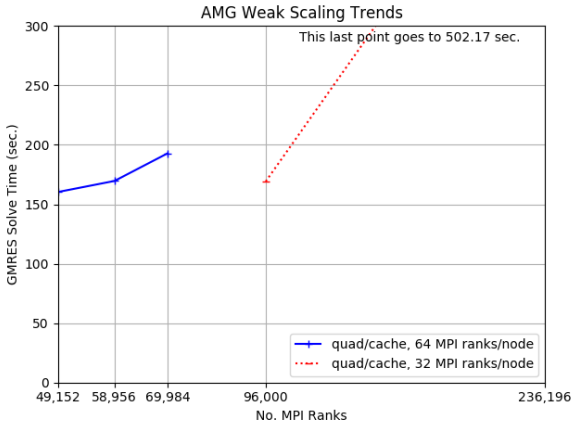

Figure 5. AMG weak scaling for *quad/cache* sized from SSP problem; this is a semilog plot where the x axis is logarithmic with a base of 2.

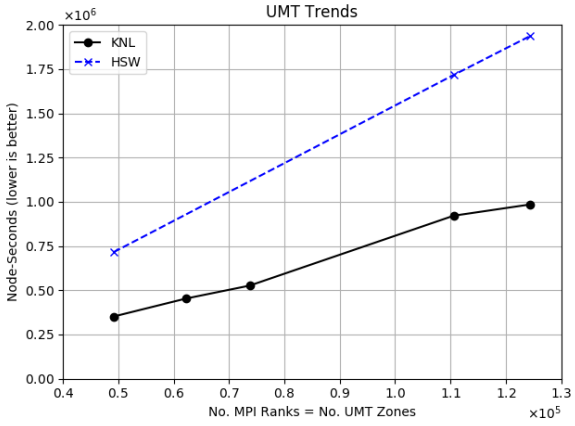

Figure 6. UMT scaling comparison between HSW and KNL for "large" MPI rank counts.

memory requirements of `MPI_Allgatherv`, also present within AMG, grow with the number of ranks; this is one of the culpable areas of memory growth. To work around this constraint, the ranks per node configuration was halved to 32 and the target node counts were doubled to scale upwards; Figure 5 contains this data with the 64 and 32 MPI ranks per node data separated. This was sufficient to run a "large" problem on Phase 2, however memory growth was still observed. This is likely due to other MPI buffers in AMG that grow as a function of the number of ranks. This is still being investigated.

4) UMT: UMT is an approximately weak scaling code. The number of iterations taken (i.e., the amount of work performed) increases with increasing number of MPI ranks. Each MPI rank corresponds to a zone in its mesh. With these characteristics, it is insightful to examine UMT with the metric "node-seconds" rather than just its time. Figure 6 provides the "node-seconds" comparing UMT on HSW with KNL (*quad/cache*). The number of iterations increases from

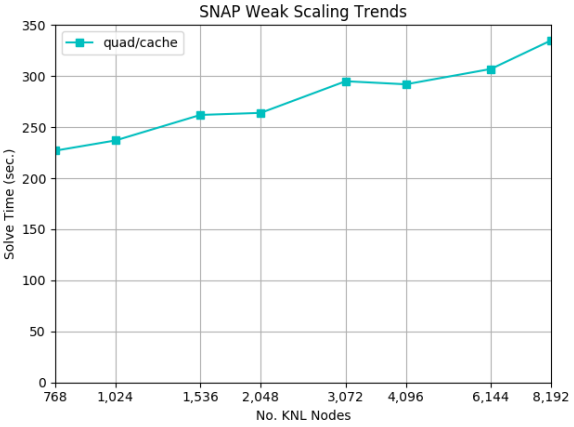

Figure 7. SNAP weak scaling for *quad/cache* sized from SSP problem; this is a semilog plot where the x axis is logarithmic with a base of 2.

116 to 119 for the 110,592 and 124,416 MPI rank cases which means these points do more work than the lower MPI rank count.

A segmentation violation occurs at 133,632 MPI ranks and points to a problem within the `CMG_CLEAN` mesh generation code. Integer(s) overflowing is the current diagnosis from `strace`. Replacing `int` with `long` declarations was not attempted.

5) SNAP: SNAP weak scaling on Phase 2 is provided within Figure 7. Its parallel efficiency from 768 to 8,192 nodes is $\sim$68%.

D. Micro-benchmarks

As part of this effort to gather performance characteristics of Trinity, a number of micro-benchmarks [2] 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 micro-benchmarks run during Trinity acceptance.

1) HPCG: The HPCG benchmark was run on Trinity in the fall of 2016. The Intel version 2.4 of the benchmark was used and no changes were made to the code. The runs were scaled up to 8,704 nodes using 4 MPI ranks per node and 34 OpenMP threads per rank. Local domain dimensions of $160 \times 160 \times 160$ (Global `nx:ny:nz=5120:5120:5440`) were set using the HPCG command line options `--nx`, `--ny`, `--nz`, and the execution time was set to 4,000 seconds using the 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` command line option `-cc depth` was used. The best GFLOP/s rating reported was 343,071.

2) ZiaTest: ZiaTest, through 7 well-defined steps, performs a new, proposed benchmark method for MPI startup

Table XIII
THIS TABLE PROVIDES THE PHASE 2 MPIMEMU BENCHMARK RESULTS.

No. Nodes Used on Phase 2	1,024	2,048	8,192	8,792
Avg. Memory Used per Node	2.79 GB	3.55 GB	7.76 GB	8.34 GB

that intends to provide a realistic assessment of launch and wireup requirements. Additionally, it analyzes, in a specified pattern, the environment launch system and the interconnect subsystem. Details on how the test is designed and tar file with the benchmark can be obtained from [2]. Ziatest was run on the full scale of Trinity Phase 2 and measured it a launch time of 58 seconds with 68 MPI tasks per node.

3) mpimemu: Benchmark mpimemu helps measure approximate MPI library memory usage as a function of scale. It samples `/proc/meminfo` at the node level and `/proc/self/status` at the process level and outputs the min., max., and avg. values for a specified period of time. More information is provided by NERSC [2]. Mpimemu was run on Phase 2 and Table XIII shows the MPI library memory used with 68 MPI tasks per node 2 at different node counts. For smaller scales, the memory used was found to be less than 2% of the available 96 GB DDR4 per node.

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 [2]. PSNAP was intended to run on the entire system and was executed on all the available Phase 2 nodes with 68 MPI tasks per node. The maximum percentage slowdown at a core was measured to be 0.244%.

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 [2]. The measured STREAM Triad performance for DDR on the KNL node was measured at 90.237 GB/s.

6) OSU MPI Message Benchmarks: The OSU Micro-Benchmark suite is a collection of independent MPI message passing performance micro-benchmarks developed and written at Ohio State University. It includes traditional benchmarks and performance measures such as latency, bandwidth, and message rate.

Point-to-point bandwidth, latency, and message rate benchmarks were run. Figure 8 shows the uni- and bi-directional bandwidth between a pair of tasks on two nodes for Phase 1 and Phase 2. Single rank bandwidth is lower due to increased latency from the PCIe interface to memory on

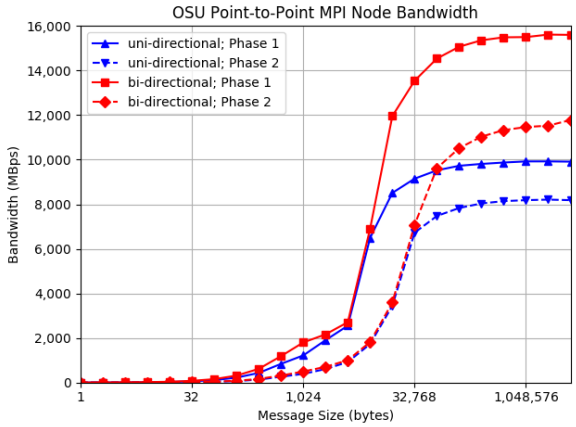

Figure 8. OMB node-to-node MPI bandwidth.

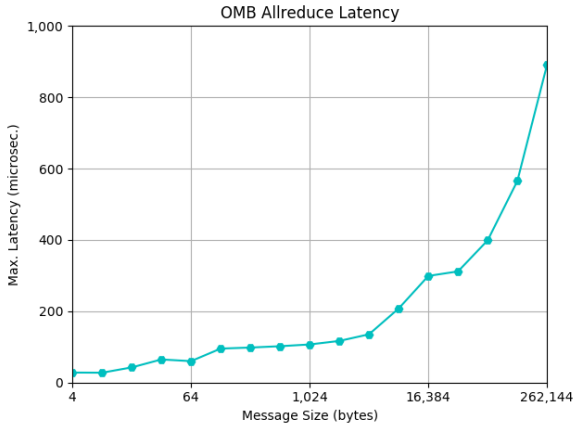

Figure 9. MPI `Allreduce` latency on 1,024 nodes.

KNL. Latency is increased compared to HSW mostly due to the slower scalar processor on KNL. Figure 9 shows the MPI collective `Allreduce` latency as function of message size on a run using 69,632 MPI tasks on 1,024 nodes of Phase 2. The 8 byte latency on 1,024 nodes (69,632 MPI tasks) was measured to be 246 microseconds and it is considerably higher than the 28 microseconds reported for Phase 1 [1] when run on close to 9,390 (300,480 MPI tasks) Haswell nodes.

IV. CONCLUSIONS

This paper documents the year that ACES and Cray performance teams spent optimizing and running the various Acceptance applications for Phase 2. This paper's goal is to disseminate the plethora of information collected that is relevant to the broader community. A list of observed trends from the work performed on the CI, SSP, and micro-benchmark applications to achieve their desired performance is provided below.

- KNL's *quad/cache* mode is a good-performing, general-purpose mode for applications who have not yet directly

addressed the MCDRAM/DDR4 memory hierarchy present. For example, "early" Qbox testing focused on the different KNL modes utilized a 2 node test case and came to a similar conclusion. Specifically, the wall time for this test case, in seconds, was 84.8 for *snc2/cache*, 75.5 for *snc4/cache*, 56.2 for *quad/flat*, 35.9 for *snc2/flat*, 31.7 for *quad/cache*, and 31.3 for *quad/equal*.

- KNL's *quad/flat* mode is an excellent mode to use if the entire simulation will fit in MCDRAM. In this scenario, it will be as performant as possible and it will have less run-to-run variation than *quad/cache*.
- The optimal compiler for KNL varies between Intel and Cray and is dependent upon the application. It should be noted that Intel recommends using version 17 (or later) of their compiler suite for KNL. All applications discussed herein focused on version 16 since 17 was not released when the acceptance effort began.
- *Static linking*, more often than not, achieves better performance than dynamic linking. In some cases, e.g., Qbox in Table VII, it is much better performance.
- *Huge pages* typically provides a performance increase and should be investigated for each application.
- *Grid ordering* improves performance for many applications and should also be investigated for each application, e.g., miniGhost in Table XII.
- *Hybrid parallelism* can improve performance over MPI everywhere on KNL, however Cray's MPICH implementation on Trinity scales remarkably well and can be used in the interim while developing hybrid parallelism within an application. It is important to closely monitor application communication patterns as they begin to scale up to ensure that there are a minimum number of point-to-point connections made to minimize MPI buffers; this is alleviated with hybrid parallelism.
- *Core specialization*, although not mentioned earlier, was used for the bulk of all simulations. Core specialization pins the OS to specific and, preferably, unused cores to minimize OS noise and interference with running applications. For these cases, enabling this increased performance, decreased run-to-run variability, or both. This option is enabled by the Cray `aprun` command line argument `--specialized-cpus`, or its shortened version `-r`.

ACKNOWLEDGMENTS

The authors would like to thank the *entire* Trinity Acceptance Team from ACES and Cray for their help and support in making this work possible through supporting late-night dedicated machine access, troubleshooting awkward behavior, and providing forward-thinking feedback. The authors are also grateful to Micheal Glass, Stephen Kennon, Paul Lin, Simon Hammond, Greg Sjaardema, David Glaze, Kendall Pierson, Mark Pagel, Mark Hoemmen, Michael Heroux, and Robert Hoekstra for their assistance with investigating, and developing eventual solutions for, the Nalu/MPI OOM issue. Additionally, the authors are thankful to Douglas Doerfler, Katerina Antypas, and Brian Austin from NERSC for their collaboration.

REFERENCES

[1] M. Rajan, N. Wichmann, R. Baker, E. W. Draeger, S. Domino, C. Nuss, P. Carrier, R. Olson, S. Anderson, M. Davis, and A. Agelastos, "Performance on Trinity (a Cray XC40) with Acceptance Applications and Benchmarks," in *Proc. Cray User's Group*, 2016.

[2] "NERSC-8 / Trinity Benchmarks." [Online]. Available: https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/

[3] "CrayDoc 4.5." [Online]. Available: http://docs.cray.com/relnotes/

[4] D. Doerfler, M. Rajan, C. Nuss, C. Wright, and T. Spelce, "Application-Driven Acceptance of Cielo, an XE6 Petascale Capability Platform," in *Proc. Cray User's Group*, 2011.

[5] M. Rajan, C. T. Vaughan, D. W. Doerfler, R. F. Barrett, P. T. Lin, K. T. Pedretti, and K. Scott Hemmert, "Application-driven analysis of two generations of capability computing: the transition to multicore processors," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 18, pp. 2404–2420, 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.2830

[6] Domino, S., "Sierra Low Mach Module: Nalu Theory Manual 1.0," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2015-3107W, 2015.

[7] "NaluCFD/Nalu: a generalized unstructured massively parallel low Mach flow code designed to support a variety of energy applications of interest (most notably Wind ECP)." [Online]. Available: https://github.com/nalucfd/nalu

[8] "NaluRtest/nightly/milestoneRun at master." [Online]. Available: https://github.com/NaluCFD/NaluRtest/tree/master/nightly/milestoneRun

[9] A. M. Agelastos and P. T. Lin, "Simulation Information Regarding Sandia National Laboratories' Trinity Capability Improvement Metric," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2013-8748, October 2013.

[10] R. S. Baker and K. R. Koch, "An Sn Algorithm for the Massively Parallel CM-200 Computer," *Nuclear Science and Engineering*, vol. 128, no. 3, pp. 312–320, Jan 1998.

[11] F. Gygi, "Architecture of Qbox: A scalable first-principles molecular dynamics code," *IBM Journal of Research and Development*, vol. 52, no. 1.2, pp. 137–144, Jan 2008.

[12] "Qbox code." [Online]. Available: http://qboxcode.org

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