SAND2016-4315C

Trinity: Architecture and Early Experience

K. Scott Hemmert, Mahesh Rajan, Rob Hoekstra,
Michael W. Glass, Simon D. Hammond
Sandia National Laboratories
Albuquerque, NM, USA

Shawn Dawson
Lawrence Livermore National Laboratory
Livermore, CA, USA

Manuel Vigil, Daryl Grunau, James Lujan,
David Morton, Hai Ah Nam, Paul Peltz Jr.,
Alfred Torrez, Cornell Wright
Los Alamos National Laboratory
Los Alamos, NM, USA

***Abstract*—This paper presents a high-level summary of the architecture and early experience with Trinity, the first DOE ASC Advanced Technology System (ATS). Trinity is a Cray XC40 supercomputer with planned delivery in two phases: a Haswell first phase, with Knights Landing being added to phase 2. The paper describes many aspects of the overall Trinity platform and project.**

Keywords-HPC

I. Introduction

Trinity is the U.S. Department of Energy's (DOE) Advanced Simulation and Computing (ASC) major computing system. Phase 1 of Trinity was delivered in fiscal year 2015. The system is being procured and operated by the New Mexico Alliance for Computing at Extreme Scale (ACES), a joint Los Alamos National Laboratory (LANL) and Sandia National Laboratories (SNL) partnership, and will be installed at LANL. ACES is funded by the DOE's ASC Program. Procurement of a major system is a complex and time-consuming process, with the Trinity contract awarded to Cray Inc. on July 9, 2014. The early phase of the procurement was a joint effort with National Energy Research Scientific Computing (NERSC).

Trinity is designed to support the largest, most demanding Directed Stockpile Work (DSW) applications that support the NNSA's Stockpile Stewardship Program. ATS platforms are used by applications from all three nuclear weapons laboratories, and the mission need was developed with tri-lab input. The mission need concentrates on increases in geometric and physics fidelities in 3D, while satisfying analysts' time-to- solution expectations. The 3D weapon applications are mainly constrained by available memory. The main driver for Trinity is the desire to run multiple large jobs on the system. Trinity was the first DOE system specified by *memory*, not by *FLOPS* (floating operations per second).

The ASC national computing strategy defines two types of systems [1]. The Commodity Technology Systems (CTS) are robust, cost-effective systems that are designed to meet the day-to-day simulation needs of the Stockpile Stewardship Program .

The Advanced Technology Systems (ATS) are first-of-a-kind systems that identify and foster technical capabilities and features that are beneficial to ASC applications. These systems have a dual purpose, to meet unique mission needs of the Stockpile Stewardship Program, and to help prepare the ASC Program for future system designs. These are leadership-class systems, among the largest in the world. When procuring an ATS there is a tension between acquiring the right-sized platform to meet the mission needs and pursuing promising new technologies. ATS procurements include Non-Recurring Engineering (NRE) funding to enable delivery of new technologies for leading-edge platforms.

The ASC notational computing platform procurement timeline is shown in Figure 1. The strategy includes deliberate efforts to transition the application codes to each ATS platform. Trinity is the first ATS procured by ASC.

Trinity NRE covers improved burst buffer software, advanced power management, and the Trinity Center of Excellence (COE). The COE directly supports modifying select applications for Trinity, and is an essential element in making effective use of Trinity.

Trinity is a single system that contains both Intel Haswell and Xeon Phi Knights Landing (KNL) processors. It is based on the Cray XC architecture. The Haswell partition, delivered in FY15, is well suited to existing codes and provides the immediate ability to partially satisfy stockpile stewardship mission needs while the application codes are modified for the KNL partition. The KNL partition, delivered in FY16, results in a system significantly

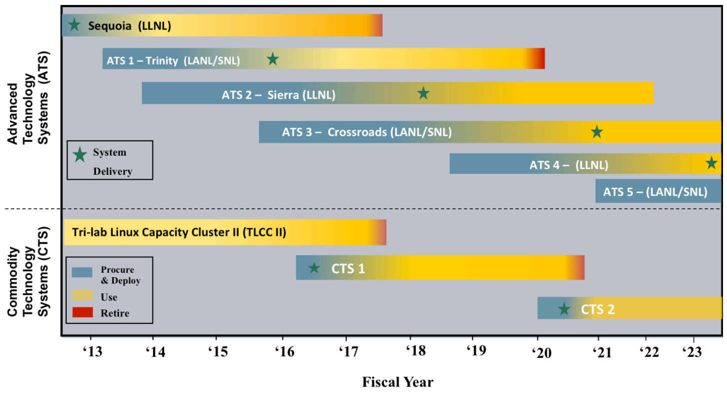

Figure 1: The notational ASC computing platform timeline

more capable than current platforms and provides the application developers with an attractive next-generation processor architecture. The intentional mix of processor types (Haswell and KNL) results in a platform that meets both of the ATS requirements, support of large simulations for Stockpile Stewardship Program with current ASC applications while advancing the development and use of emerging programming models and work flows.

II. Architecture

Trinity is a Cray XC40 supercomputer, with delivery over two phases; phase 1 is based on Intel Xeon Haswell compute nodes, and phase 2 will add Intel Xeon Phi Knights Landing (KNL) compute nodes. The high level Trinity architecture is shown in Figure 2.

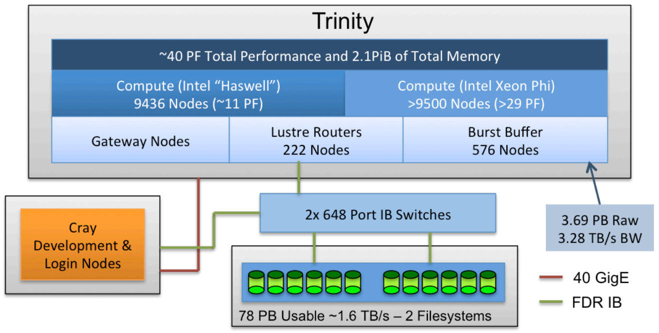
Trinity
~40 PF Total Performance and 2.1PiB of Total Memory
Compute (Intel "Haswell") 9436 Nodes (~11 PF)
Compute (Intel Xeon Phi) >9500 Nodes (>29 PF)
Gateway Nodes
Lustre Routers 222 Nodes
Burst Buffer 576 Nodes
2x 648 Port IB Switches
3.69 PB Raw 3.28 TB/s BW
Cray Development & Login Nodes
78 PB Usable ~1.6 TB/s – 2 Filesystems
40 GigE
FDR IB

Figure 2: Trinity high-level architecture

Phase 1 was delivered and accepted in the latter part of 2016, and consists of 54 cabinets, including multiple node types. Foremost are 9436 Haswell-based compute nodes, delivering ~1 PiB of memory capacity and ~11 PF/s of peak performance. Each Haswell compute node features two 16-core Haswell processors operating at 2.3 GHz, along with 128GiB of DDR4-2133 memory, spread across 8 channels (4 per CPU). Phase 1 also includes 114 Lustre router nodes (see Section III.B) and 300 burst buffer nodes (see Section IV). Trinity utilizes a Sonexion-based Lustre filesystem with 78 PB of usable storage and approximately 1.6 TB/s of bandwidth. However, due to the limited number of Lustre router nodes in Phase 1, only about half of this bandwidth is currently achievable. Phase 1 also includes all of the other typical service nodes: 2 boot, 2 SDB, 2 UDSL, 6 DVS, 12 MOM, and 10 RSIP. Additionally, Trinity utilizes 6 external login nodes.

Phase 2 is scheduled to begin delivery in mid-2016. It adds more than 9500 Xeon Phi Knights Landing (KNL) based compute nodes. Each KNL compute node consists of a single KNL with 16 GiB of on-package memory and 96 GiB of DDR4-2400 memory. It has a peak performance of approximately 3 TF/s. In total, the KNL nodes add ~1 PiB of memory capacity and ~29 PF/s peak performance. In addition to the KNLs, Phase 2 also adds the balance of the Lustre router nodes (108 additional, total of 222) and burst buffer nodes (276 additional, total of 576). When all burst buffer nodes are installed, they will provide 3.69 PB of raw storage capacity and 3.28 TB/s of bandwidth.

III. Early Performance Results

A. *Application Performance*

ACES management recognized the importance of good application performance and setup as part of the four Trinity SOW performance related deliverables. They were completed during Trinity Phase 1 acceptance in December 2015. Of the four, ASC application code Capability Improvement (CI) metric, measuring applications performance at near full scale was a principal focus. The CI metric measured three applications' performance improvement, defined as the product of an increase in problem size, and/or complexity, and an application specific runtime speedup factor over baseline measurement on NNSA's Cielo [2][3](a Cray XE6). For example, if the problem size increases by a factor of eight and the run time speedup is 1.2, the CI is 8x1.2=9.6. The three applications picked as representative of the Tri-Lab workload are: SIERRA/Nalu [4][5] (SNL), PARTISN [6] (LANL), and, Qbox [7] (LLNL). For Trinity, target performance for the CI metric is an average of eight, but split into four for the Phase 1 Haswell partition and four for the Phase 2 KNL partition. The second important acceptance performance deliverable was a target of 400 on the Sustained System Performance (SSP) metric [8], measured as a geometric mean of the performance of eight applications. The third deliverable was to run five chosen applications/benchmarks from the SSP suite at near full scale of Trinity. This along with a fourth deliverable to run several micro-benchmarks provided excellent insight into performance and scaling characteristics of Trinity. The sections below provide short descriptions of results from Trinity Phase 1 performance acceptance tests. Further details are available in [9].

To provide some context a short description of a few primary factors that have an impact on performance follows. Trinity nodes are set up to support Hyperthreads. Processor Turbo Boost is turned on and the operating clock frequency varies with the thermal load. Assuming a nominal 2.3 GHz operation, the peak node double precision performance (7.67X of Cielo) is 32cores*16FLOPs/cycle*2.3 GHz = 1,177.6 GF/s/node. Each core is capable of 16 DP FLOPs per cycle from the two 256 bit AVX2 units with FMA. The software environment included Craype 2.4.2, Cray Libsci 13.2.0, Cray Alps 1.8.3, Intel 15.0 compiler, Cray CCE 8.4.0 compiler, GNU 5.1.0 compiler and Cray MPICH/7.2.5. Trinity is listed at 8,101 TF/s on top500.org and 182.6 TF/s on hpcg-benchmark.org.

1) *ASC Capability Improvement (CI) Application Performance*

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. The 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. 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 unstructured meshes with hexahedral elements. The problem run on Cielo and Trinity is the R6 mesh that consists of nine billion elements, with total degree-of-freedom count approaching 60 billion. Two figures of merit 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 figure of merit 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

Runs of Nalu on 9300 nodes on Trinity and 8192 nodes of Cielo were used for the CI computation. The excellent scaling of Nalu on Trinity resulted in a performance gain of 4.26X for the assembly time and a performance gain of 3.89X for the matrix solve time, resulting in a CI metric value of 4.009.

The 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. Runs of PARTISN on 9418 nodes on Trinity, with an input of 11,520 zones/core were compared to 8192-node baseline run on Cielo with an input of 2,880 zones/core. This gave a complexity increase of 9.19 and a run time ratio of 0.512, resulting in the CI metric value for PARTISN of 4.84.

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. The Qbox benchmark problem is the initial self-consistent wavefunction convergence of a large crystalline gold system (fcc, a0 = 7.71 a.u). The metric for this benchmark is the maximum total wall time to run a single self-consistent iteration with three non-self-consistent inner iterations. Runs of Qbox on 9418 nodes on Trinity, with an input of 8,800 atoms were compared to 6,144-node baseline run on Cielo with an input of 2880 atoms. The computation complexity grows as the cube of the number of atoms. This gave a complexity increase of 166.37. The run time ratio was measured to be 0.208, resulting in a CI metric value for Qbox of 34.7.

Figure 3 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. Investigations on performance optimizations with input parameter changes and run time optimizations such as threads per MPI rank and MPI task rank ordering are available in [9].

2) *Sustained System Performance(SSP) benchmarks*

A second performance goal was a target System Sustained Performance (SSP) of 400. SSP [8] is computed using the geometric mean of the run time of eight application benchmarks at various scales as shown in Table 1. The measured performance on Trinity running these applications was 500.

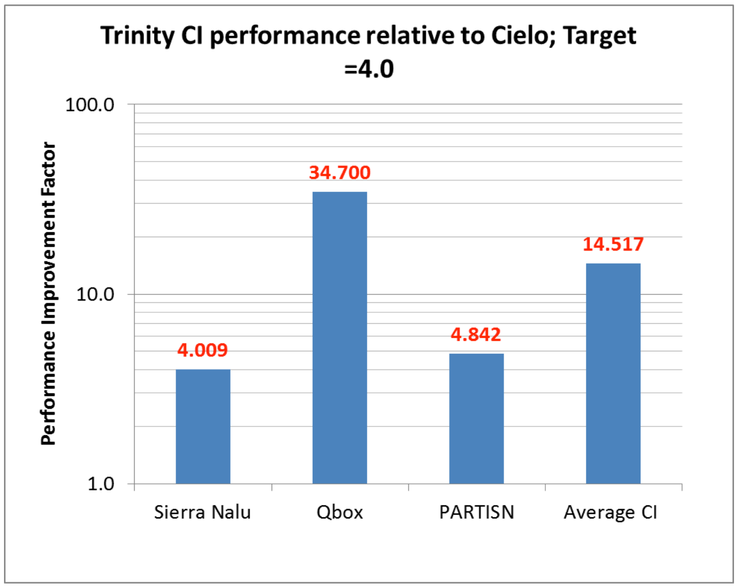

Figure 3: Trinity Capability Improvement Performance

Table 1: Trinity System Sustained Performance (SST) results

Application Name	MPI Tasks	Threads	Nodes Used	Reference Tflops	Time (seconds)	Pi
miniFE(Total CG Time)	49152	1	1536	1065.151	49.5116	**0.014005964**
miniGhost(Total time)	49152	1	1536	3350.20032	1.77E+01	**0.122949267**
AMG(GMRES Solve wall Time)	49152	1	1536	1364.51	66.233779	**0.013412384**
UMT(cumulativeWorkTime)	49184	1	1537	18409.4	454.057	**0.026378822**
SNAP(Solve Time)	12288	2	768	4729.66	1.77E+02	**0.034793285**
miniDFT(Benchmark_time)	2016	1	63	9180.11	377.77	**0.385726849**
GTC(NERSC_TIME)	19200	1	300	19911.348	868.439	**0.076425817**
MILC(NERSC_TIME)	12288	1	384	15036.5	393.597	**0.099486409**
					Geom. Mean=	**0.052990429**
					SSP=	**500.0176846**

As part of acceptance, some of the SSP applications were run at near full scale using "extra-large" inputs. This performance data together with important micro-benchmarks like OMB, SMB, STREAM, IOR, and PSNAP provide important insights into Trinity performance. These are discussed in [9].

B. File System Performance

The Trinity parallel file system is implemented using the Lustre-based Cray Sonexion 2000 product. The file system is divided into two equally sized scratch file systems. Phase 1 consists of approximately one-half of the LNET routers, thus bandwidth performance is expected to be one-half of the fully deployed phase 2 system. As part of acceptance, optimally tuned IOR benchmark runs were defined to achieve maximum performance. Maximum performance was evaluated while scaling stream counts per OST and processor counts per node. The benchmark was run against both scratch file systems to verify functionality and performance. File-per-processor (N-N) IOR runs exhibited performance of 600 GiB/s using 8 streams per OST and 2 processors per node . Optimally tuned shared file (N-1) IOR runs yielded performance of 350-400 GiB/s using 32 processors per node and a transfer size of 8MiB. The file was targeted to a Lustre directory striped nearly full OST count wide with a Lustre stripe size of 8M to match the transfer size.

Performance sweeps and IOR tuned to match application patterns are shown in Figures 4 through 7. The tests were configured to write 1GiB per processor and were executed on one of the two scratch file systems. Figure 4 shows file-per-processor (N-N) write results using 32 processors per node and transfer sizes of 4MB and 8MB. Peak performance of 401 GiB/s occurred at the 32K and 64K processor count. Read performance peaked at 420GiB/s at a processor count of 64K, as shown in Figure 5.

It should be noted that these plots do not necessarily represent peak performance. In some cases, the rates shown are 15% lower than previous identical runs. A post analysis showed that the file system problems did exist during the execution of the tests.

The phase 1 acceptance test included a test to dump 0.8TB in 20 minutes. This test was successful using IOR N-N targeting both file systems simultaneously. The rates obtained were 686 GiB/s for write and 572GiB/s for read. This test used the optimal 8 streams per OST and 2 processors per node configuration described previously.

The next set of performance sweeps were based on the N-1 strided I/O pattern. The target Lustre directory was striped full OST count wide and the stripe size was configured to match the IOR transfer size. Figure 6 shows N-1 write performance as the processor count is scaled. Peak N-1 performance of 301 GiB/s occurred at a processor count of 64k and using a 4MiB transfer size. Figure 7 shows the corresponding read performance.

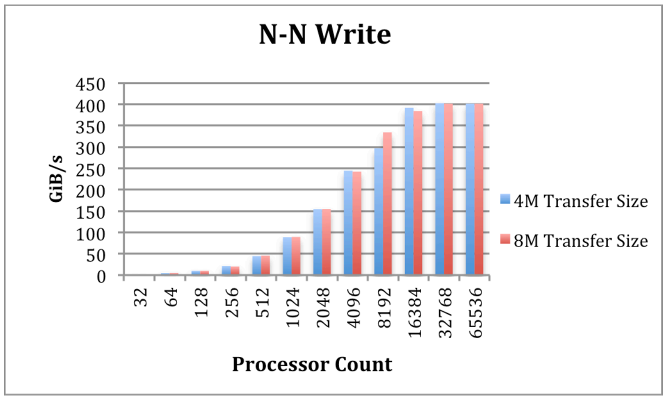

Figure 4: Trinity N-N write performance

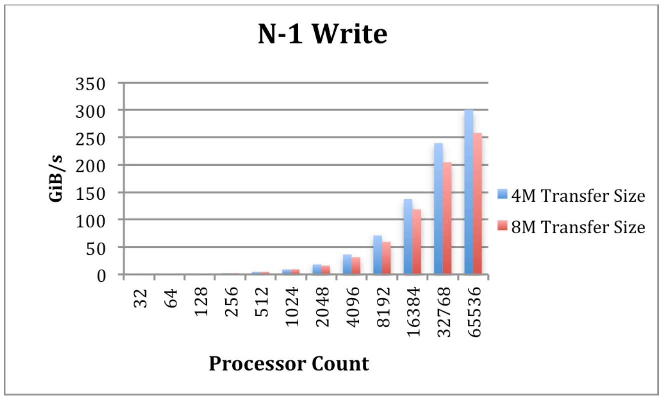

Figure 6: Trinity N-1 write performance

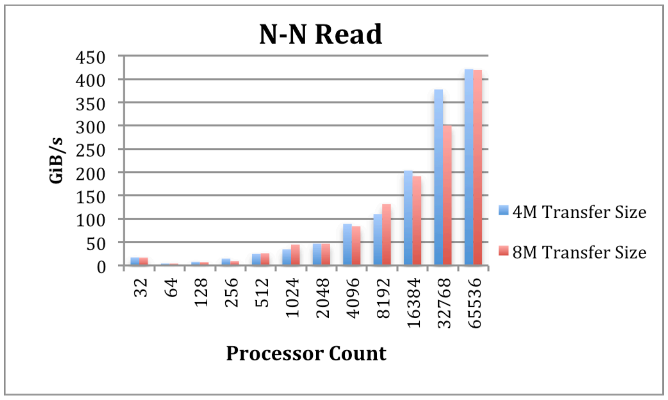

Figure 5: Trinity N-N read performance

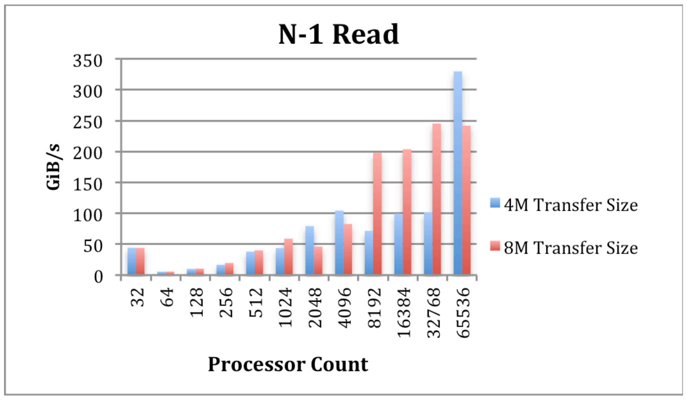

Figure 7: Trinity N-1 read performance

Performance peaked at 330 GiB/s at a processor count of 64K.

To gain insight on the impact of Lustre DNE[11], metadata performance was measured using mdtest targeting directories hosted by all Metadata Servers (one directory per Metadata Server). Results are shown in Figure 8. The goal of the test was to create, stat, and delete 1 million files while varying processor count. This test was run during acceptance and yielded peak create, stat, and delete rates of 172k ops/s, 458k ops/s, and 259k ops/s respectively.

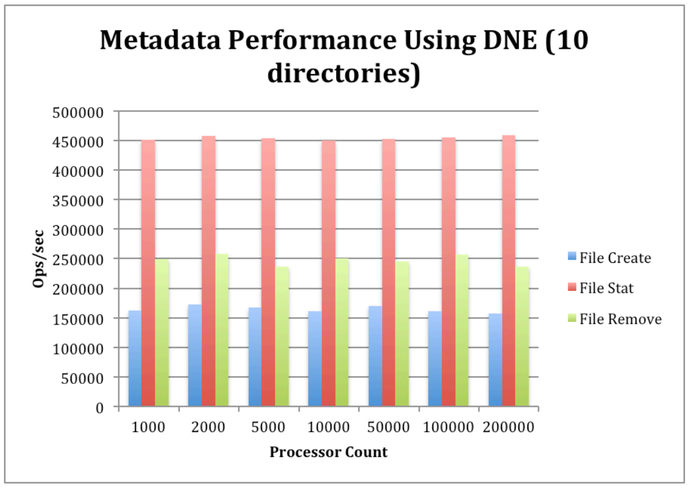

Figure 8: Trinity metadata performance

IV. Burst Buffer Integration and Performance

A. Design

Trinity includes the first large scale instance of on-platform burst buffers using the Cray DataWarp® product. The Trinity burst buffer is provided in two phases along with the two phases of Trinity. The phase 1 burst buffer consists of 300 DataWarp nodes. This is expanded to 576 DataWarp nodes by phase 2. In this section, unless otherwise noted, the phase 1 burst buffer will be described.

The 300 DataWarp nodes are built from Cray service nodes, each with a 16 core Intel Sandy Bridge processor and 64 gigabytes of memory. Storage on each DataWarp node is provided by two Intel P3608 Solid State Drive (SSD) cards. The DataWarp nodes use the Aries high speed network for communications with the Trinity compute nodes and for communications with the Lustre Parallel File System (PFS) via the LNET router nodes.

Each SSD card has 4 TB of capacity and is attached to the service node via a PCI-E x4 interface. The SSD cards are over-provisioned to improve the endurance of the card from the normal 3 *Drive Writes Per Day* (DWPD) over 5 years to 10 DWPD over 5 years. This reduces the available capacity of each card. The total usable capacity of the 300 DataWarp nodes is 1.7 PiB.

The DataWarp nodes run a Cray provided version of Linux together with a DataWarp specific software stack consisting of an enhanced Data Virtualization Service (DVS) server and various configuration and management services. The DataWarp nodes also provide a staging function that can be used to asynchronously move data between the PFS and DataWarp. There is a centralized DataWarp registration service that runs on one of the Cray System Management nodes. Compute nodes run a DVS client that is enhanced to provide support for DataWarp. The DataWarp resources can be examined and controlled via several DataWarp specific command line interface (CLI) utilities that run on any of the system's nodes.

DataWarp can be configured to operate in a number of different modes. The primary use case at ACES is to support checkpoint and analysis files, these are supported by the *striped scratch* mode of DataWarp. *Striped scratch* provides a single file name space that is visible to multiple compute nodes with the file data striped across one or more DataWarp nodes. A *striped private* mode is additionally available. In the future, *paging space* and *cache* modes may be provided. This section will discuss LANL's experience with *striped scratch* mode.

A DataWarp allocation is normally configured by job script directives. Trinity uses the Moab Work Load Manager (WLM). The WLM reads the job script at job submission time and records the DataWarp directives for future use. When the requested DataWarp capacity is available, the WLM will start the job. Prior to the job starting, the WLM uses DataWarp CLI utilities to request instantiation of a DataWarp allocation and any requested stage-in of data from the PFS. After the job completes, the WLM requests stage-out of data and then frees the DataWarp allocation. The stage-in and stage-out happen without any allocated compute nodes or any compute node involvement. The DataWarp allocation is made accessible via mount on only the compute nodes of the requesting job. Unix file permissions are effective for files in DataWarp and are preserved by stage-in and stage-out operations.

A DataWarp allocation is normally only available for the life of the requesting job, with the exception of a *persistent* DataWarp allocation that may be accessed by multiple jobs, possibly simultaneously. Simultaneous access by multiple jobs is used to support in-transit data visualization and analysis use cases.

B. Integration

Correct operation of DataWarp in conjunction with the WLM was achieved after several months of extended integration testing on-site at LANL. Numerous fixes and functional enhancements have improved the stability and usability of the DataWarp feature on Trinity. Due to this effort, production use of DataWarp has been limited as of late April, 2016.

C. Performance

All performance measurements were conducted with IOR. The runs were made with:

- 1 reader or writer process per node
- 32 GiB total data read or written per node
- 256, 512 or 1024 KiB block size
- Nodes counts from 512 to 4096
- The DataWarp allocation striped across all 300 DataWarp nodes

These characteristics were selected to approximate the IO patterns expected when applications use the HIO library. Additional investigation and optimization of IO characteristics is needed. Results are shown in Figures 9 through 12.

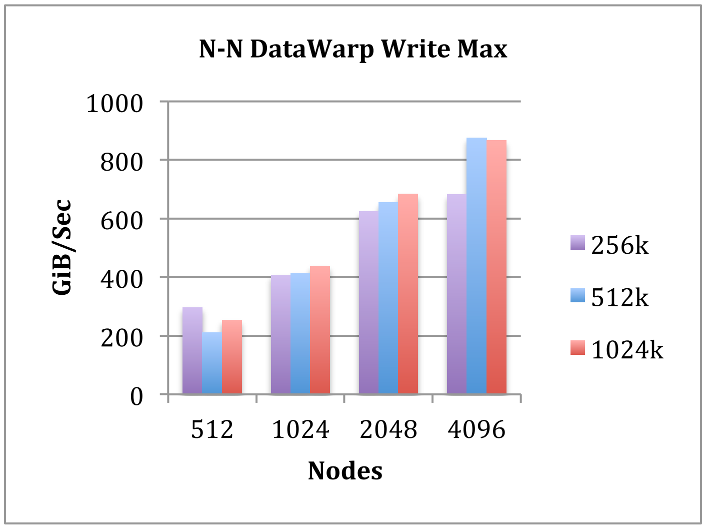

Figure 9 - DataWarp N-N Write Speed

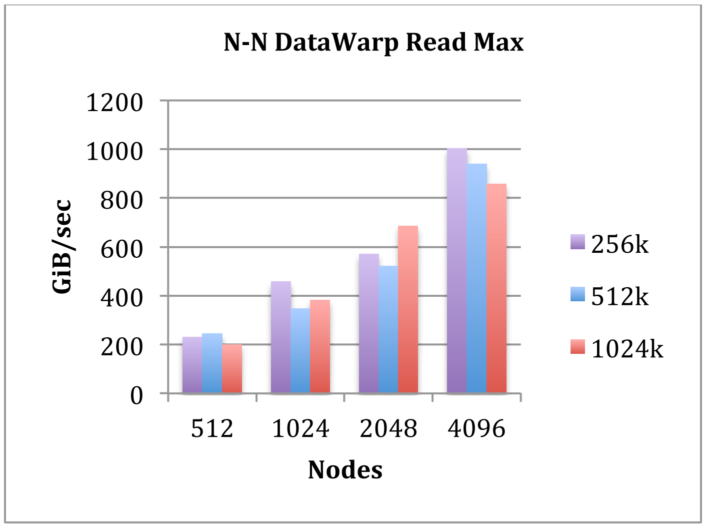

Figure 10 - DataWarp N-N Read Speed

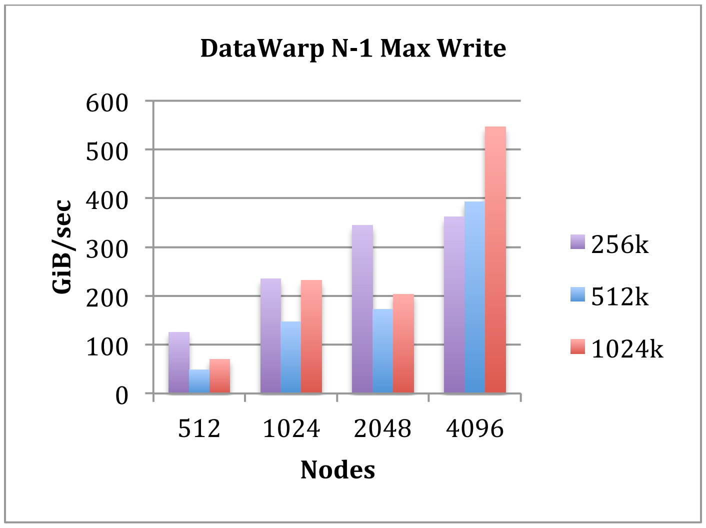

Figure 11 - DataWarp N-1 Write Speed

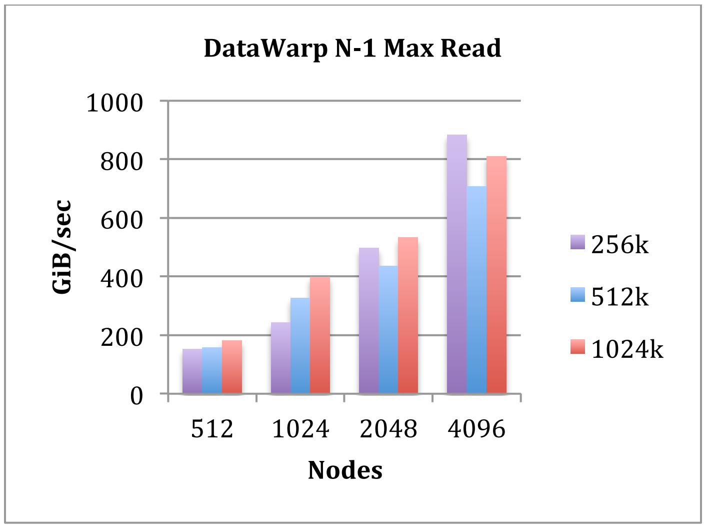

Figure 12 - DataWarp N-1 Read Speed

V. SYSTEMS MANAGEMENT AND INTEGRATION

A. *ACES/Cray Collaboration*

ACES and Cray have been collaborating on the Rhine/Redwood (CLE 6.0/SMW 8.0) software release while it was still in active development and through several beta releases until the release of UP00 in December 2015. This collaboration is unprecedented for Cray and it has proven to be very successful. As new releases of CLE 6.0/SMW 8.0 became available, system administrators at ACES would install, test, and give feedback directly to the developers. Through this tightly coupled collaboration, Cray was able to deliver a more refined and administrator friendly version of both the CLE and the SMW management tools because of the exposure of this product to a customer in the field. More often than not, the developer's implementation on an internal system in a controlled environment is not always adaptable to the customer managing a Cray machine at a site. Because of the close relationship with Cray, the ACES administrators had a rapid feedback cycle. The early use of CLE 6.0 at LANL uncovered a variety of bugs and design issues, as well as producing several new ideas which improved the eventual deployment on Trinity. This collaboration also provided invaluable experience to the ACES system administrators who would eventually support Trinity in production.

B. *Early Experience with Rhine/Redwood*

Trinity is the first Cray system to deploy the new CLE 6.0/SMW 8.0 system management software. This release pair is a complete redesign and modernization of the software stack, positioning Cray closer to industry standards than ever before. However, this new design brings new challenges. At the heart of Rhine/Redwood is an advanced '*configurator*' which, paired with the Ansible configuration management utility, handles the complete prescription of all system components - from SMW to compute and even external login nodes. The ACES systems team has worked extensively with Cray to refine and guide the development of CLE 6.0/SMW 8.0 (6.0/8.0) codenamed Rhine/Redwood (R/R) so that it is functional, scalable, and its system management is automatable. The ACES team has also strived to see that the new release would be a more robust product for the larger Cray administrator community. The work in this section describes the administrative decisions made by the ACES team and how our implementation provides system stability and consistency, reducing downtime and improving the user experience.

1) Pre-Release Evaluation and Preparation

The ACES team received a beta release of 6.0/8.0 in early June 2015 and installed it on a Test and Development System (TDS). Our experience with the initial install itself was difficult for a variety of reasons. The system management philosophy is radically different under 6.0/8.0 than with previous versions, making the learning curve for this new release difficult. A significant amount of time is required to perform the first install of Rhine/Redwood on a system that has never before seen the stack, even on a small TDS like ours. This is due to the fact that there is no upgrade process from the previous 5.x/7.x version to 6.0/8.0. The SMW and boot RAID are required to be reformatted and reconfigured for the new version. Once the base install of the SMW is completed, the new system must be defined using the configurator's "question and answer" text-based interface. This is a tedious process in which the administrator must specify, by hand, many of the details of the system's configuration, such as its entire network profile. The ACES team provided significant feedback to Cray on the issues that were encountered and feature requests to be included in future releases many of which were already in the process of being implemented in Cray's development release.

With subsequent releases of 6.0/8.0, new features were added such as worksheets. Worksheets allow the administrator to pre-configure the system's detail and then import those configurations into the configurator tool. These worksheets can then be used moving forward if the SMW would ever need to be reinstalled. Worksheets provide a mechanism to prepare for an initial install of a system as well. If its worksheets are fully populated, then most of the system will be prescribed on import, greatly simplifying the initial install process. However, we have learned there is still extensive work to be done after the initial install and boot of the system. Even so, the ACES team could quickly see the value in Rhine/Redwood's new philosophy and the flexibility that it provided. The removal of the shared root space, the advent of customizable images, and the use of Ansible plays to prescribe each node allow for much more flexibility and scalability than the functionality that was previously provided in 5.x/7.x.

2) Configuration Management

The ensuing work, post-install, was to learn how to effect change onto the system. The administration team was faced with the challenge of implementing its own configuration management strategy or to somehow make use of Cray's Ansible system to apply changes. At first it was decided to try to *supplement* Cray's Ansible infrastructure to fully prescribe the machine. This initial strategy proved to be the wrong path for a few reasons. First, if there was an issue with any defined Ansible play it most likely caused a failure in the boot process. The symptoms of these errors would typically mislead the administrator into searching for a system problem that had no association to the real Ansible failure. Secondly, the administrators realized that adding ancillary plays to Cray's Ansible playbooks increased the overall node boot time due to the additional burden of processing those plays. Therefore, it was decided to restrict the number of plays to be as few as necessary. Third, Cray's Ansible framework is truly intended to be run only once at boot time of the node. Execution of the full playbook is a lengthy process and resource-intensive. Therefore, a small subset of plays is selected to be run at a specified interval to maintain the configuration of a node throughout its lifecycle. Compute nodes are of even more concern because, while it is desirable to keep them updated, the overhead of running Ansible while jobs are executing is too high. It is a widely accepted fact that OS and network jitter can have a large impact on job performance. To mitigate this problem, the ACES team decided to implement a lightweight, disjoint Ansible play that would run in TORQUE's epilogue after a job was completed to update the node.

All of the challenges and issues with using the new CLE 6.0 software required a considerable amount of time and effort to overcome. Initially there was very little documentation so learning the new software consisted of trial and error, trying new things to see what would break. Another pain point was that Cray's Ansible infrastructure would take control of certain configuration files that are traditionally handled by the site's own configuration management strategy. The difficulty arises when more than one player is found playing in the same sandbox. One example of this is the management of the *sshd* service. Cray's Ansible play needs to configure the daemon at install time in order to have a functional system. However, conflicts may surface once the site decides to extend the customization of the configuration that Cray prescribes. At what priority should site plays be run when they conflict with vendor plays? How should configuration incompatibilities between site and vendor be handled? What about those configuration files whose content is only partially prescribed and the site needs to extend the prescription? Overcoming these issues is still an ongoing work for the most part. Workarounds have been implemented to solve many of these issues, but a better long-term strategy from Cray is required.

3) External Login Nodes

Another new feature of the CLE 6.0 stack is the replacement of the Bright Cluster Manager software used to provision the esLogin nodes. Cray has deployed OpenStack on the provisioning node, which is named the Cray System Management Software (CSMS) system. The key feature of this change is that the newly-named eLogin nodes are built from the same images that the internal systems are built from. This allows for a commonality between the system software and package versions found in all image types. The programming environment (PE) is also shared between the internal system and eLogin nodes. Unfortunately, the security model of OpenStack, by default, is not solid enough to meet the rigorous standards of the Department of Energy. Cray and ACES have worked closely to resolve these issues and help steer the CSMS product to have a better security posture.

C. *Integrating New Technologies*

Along with the new CLE 6.0 / SMW 8.0 software stack, Trinity is also deployed with some new hardware technologies. One such is the Sonexion 2000, which is a Lustre appliance. This Lustre deployment has been split into two equally sized 39PB file systems. One of the new features of the Lustre server on the system is DNE[11] which allows for multiple metadata targets (MDT) to distribute metadata performance across multiple servers. The current implementation of this feature, however, requires the manual placement of directories onto a specific MDT, forcing decisions to be made *a priori* by the user to take advantage of this feature. However, the ACES team has not enabled user control over the placement of directories on a specific MDT. This configuration will be controlled administratively upon request since it is important to maintain an even distribution of users across the platform.

Another new technology deployed on Trinity is Cray's DataWarp burst buffer solution. These SSD-endowed nodes allow for an intermediary scratch file system to interpose between the compute resource and parallel file system. Integration of this technology into the machine has been another one of the more difficult problems the administrators have faced. The interaction between the workload manager, Moab, and the DataWarp API has been an ongoing effort. Users need an on-demand allocation of the DataWarp file system on a per-job basis, along with a definition of the granularity and striping across multiple burst buffer nodes to achieve maximum performance. The complexities in scheduling these resources and the multitude of DataWarp failure modes have caused a variety of error scenarios that are difficult to recover from. The administrators have encountered orphaned allocations from dead jobs, and burst buffer nodes in a failed state. When these problems arise, their solution is not always intuitive and the recovery procedure is only now being documented to restore functionality to the system. As the DataWarp project matures however, and the Moab code improves to handle the scheduling of these resources, the administrators expect many of these failures to be handled more gracefully by the system. This will eventually make the recovery and restoration of the DataWarp service less burdensome.

D. *Current Challenges*

The current challenges that face the ACES administrative team are reliable boots, DataWarp at scale, and monitoring. Boot reliability is an ongoing issue, but is being actively investigated by Cray engineers. Boot integrity is a source of frustration for Cray administrators as well. All complete-system boot failures carry a large opportunistic cost in valuable uptime to users due to the necessity of the bounce, and route, which accompanies each boot process. Other failure modes may occur if the collection of Ansible plays malfunctions or is loath to run at all. This can leave the end node in a state in which no remote connectivity is possible. The only way to analyze such a failure is to use *xtcon* to connect to the node and inspect its Ansible logs. If a large set of nodes fail in this way, though, it is extremely difficult to recover apart from a full system reboot. Cray's Scalable Services system affords a parallel boot process by distributing configuration through various tiers, which are ideally comprised of existing service nodes. There have been obstacles with this composition in practice however, and currently a small set of re-purposed compute nodes are required to handle these duties. Cray has done extensive testing in this area and found that there is a small set of frequently accessed files that should be moved into the node's RAM image to reduce network load during boot. Compute nodes are being used for this task because they have much more memory available for Scalable Services to cache these distributed file systems. The practice of stealing compute resources for this purpose is a source of concern to the ACES team, but these issues are being addressed and should be fixed soon.

Another challenge for the administrators has been deploying and testing DataWarp at scale. There has been ongoing work with Cray and Adaptive to fully deploy an end-to-end solution. Many of the issues have arisen from the fact that ACES utilizes an external host for the workload manager and does not run the Moab server on the SDB. This has led to a number of communication and timing faults with the scheduling and management of DataWarp resources. Many of these issues are exposed on the small TDS so much of the testing has not been attempted at full scale on Trinity – yet another concern for the ACES administration team. There has not been enough exposure time to DataWarp and the Moab scheduler to be comfortable recovering from the failure modes that might arise. This will only be compounded when the system is in the classified network where debugging these issues becomes a serious effort.

Monitoring is another area in which much has been done in preparation for the sheer volume of log and performance data that Trinity will generate on a daily basis. Estimates are that about 4TB of uncompressed data will flow from the machine per day. Resiliency and failure tracking are a few of the many areas of interest in these logs. Features such as enhanced MCA logging [12] and SSD drive telemetry data [13] allow ACES to track failures and component health of the system. Collection and analysis of this aggregation of information has required the ACES team to develop an external monitoring cluster simply to store and process this bulk of data throughout the life of the machine.

E. *Continuing Collaboration*

Cray and ACES both continue to collaborate on the security and functionality issues that are encountered with this current release version of CLE 6.0. The features and fixes provided in CLE 6.0 UP01 promise to be a great improvement over the existing version and the ACES team is eager to begin testing it out. This will also be the first public release of this product which will expose it the entire Cray community, which will invite more feedback and drive the product to maturity. Software historically takes many revisions to improve in functionality and reliability and the ACES team looks forward to the continued development and improvements made to the Cray software management stack.

VI. Application Readiness – Center of Excellence

The Trinity Center of Excellence (COE) provides a collaborative long-term relationship between subject matter experts (SME) from Cray and Intel, with ASC codes teams, to support the transition of key scientific applications to Trinity. Applications from all three nuclear weapons laboratories were identified, to focus COE activities and limited resources. These multiphysics applications often have millions of lines of code and have additional complexity, such as having many third-party software dependencies and supporting computational campaigns that are long-running with performance profiles that can change drastically throughout the lifetime of the run or by input deck.

A wide range of computational motifs and approaches are important to mission-class applications including hydrodynamics, linear solvers, deterministic transport, Monte Carlo, molecular dynamics, and material contact amongst others.

Trinity provides codes with advanced high-performance computing resources, however reaping the benefits by using them effectively is non-trivial and creates a number of challenges for our code teams. Effective use of Trinity involves the on-node challenges of the Knights Landing processor including: 1) enabling (or not hindering) compiler vectorization with AVX-512 instructions; 2) increasing parallelism to use the increased number of cores and threads; 3) identifying data structures that will benefit from residing in high-bandwidth memory and explicitly managing the memory hierarchy. In addition, the composed Trinity system provides both challenges and opportunities, such as 1) using the burst buffer to reduce I/O overhead and enabling new workflow capabilities within a simulation run, 2) scaling simulations to utilize the massive scale of the system, 3) managing simulation data on the large parallel file system, and 4) identifying new workflow/simulation capabilities to use the heterogenous composition of Trinity with two types of nodes.

Vendor SMEs in performance optimizations, tools, compilers, and hardware work closely with code teams to address effective use of Trinity resources. COE activities began well in advance of receiving the system. COE activities include:

- SME integration into daily code team development activities,
- Application exploration using vendor tools,
- Training, workshops and working groups to share best practices, and
- Early access to hardware and software.

This section describes some of the activities during the early period of the COE when Knights Landing processors were not generally available. However, significant progress was possible through collaborative application exploration to identify where to focus application development efforts, proxy application development to enable future collaborations, and early access to hardware and software with direct feedback to vendors.

A. *Collaborative Application Exploration*

Collaborative application exploration, often called 'deep dive', 'hackathon', or the Intel-centric terms 'discovery' and 'dungeon' sessions, are an invaluable opportunity to help prepare code teams to adjust to more complex hardware environments. These activities brings vendor SMEs together with application developers to identify where to focus application development and see the impact of real-time code changes with feedback and collaboration with tools and performance experts.

The vendor SMEs bring the domain knowledge of the hardware, compilers, and performance analysis tools. The DOE developers bring the domain knowledge about the physics in the source code, as well as the architecture, design, constraints, and portability goals of the application.

Deep dive activities with Cray SMEs focused on larger simulation studies using Craypat and Cray performance tools. Deep dives used testbed systems with roughly 100 nodes and eventually utilized the Trinity Phase 1 Haswell partition as it became available. Since testbed systems are installed at NNSA facilities, this opened the code exploration activity to export controlled codes, which was extremely valuable to LANL code development activities.

The Intel-led discovery and dungeon sessions focused on on-node performance, with discovery sessions hosted at NNSA sites and dungeons hosted at Intel sites. A discovery session is a one or two day working session with a couple of Intel SMEs and DOE code developers. The dungeon session is typically a week long focused time, where application or library developers (from the DOE) are partnered with performance analysts, library and tool developers and compiler specialists from Intel. The time is used to engage in intensive behavior and performance profiling of existing codes using tools and a variety of performance counters. The analysis results of these activities are then used to drive exploratory code modification to address bottlenecks.

As part of the COE effort, LLNL developers participated with Intel developers in two Discovery sessions. The goal of these Discovery sessions was to broadly identify bottlenecks in certain areas of a code. Both sessions provided useful insight to the developers. In the first session, one of the limiting factors was related to cross section data lookup – which was expected. However, another hot spot was related to pipeline flushes. By design, Monte Carlo code has a lot of branching, and branch mis-prediction resulted in a large number of stalls, which limits single core performance.

In the second Discovery session focused on different applications, code teams specifically wanted to analyze the efficiency of a certain abstraction of the OpenMP layer. They quickly identified a series of print statements that were slowing down the code unnecessarily using the Intel tools and removed them in real-time to improve performance. During the session, they observed an unexpected performance in the abstraction layer. It was unclear if this was 1) a real issue with the abstraction, 2) an issue with Intel Vtune itself in how it collects data, or 3) an issue with the OpenMP run time library. A dungeon session is scheduled to further investigate these issues.

The SNL developers participated in both discovery and dungeon sessions as a natural evolution of the code exploration process. Prior to the dungeon, discovery sessions were held to gather performance profiles of the Sierra Mechanics engineering analysis code suite. From these performance profiles, key areas were identified to focus on for the dungeon session.

The focus of the SNL dungeon at Intel was on matrix assembly and scalable linear solvers (domain decomposition and multi-grid), and the underlying data structures of these algorithms. Rather than extracting small code kernels for this activity, which drastically shortens compile time and helps to improve feedback from code changes, the SNL developers brought a substantial portion of their software environment to expose Intel SMEs to the real challenges faced by DOE developers. These core components utilize the Kokkos parallel performance portability abstraction library [20]. Initial findings have shown slow dense-BLAS performance for very small matrix operations – something previously unseen by both Cray and Intel in many DOE benchmarks and mini-applications. In addition, OpenMP tasking was found to be very limited in performance for factorization kernels in the solvers. Performance issues found in these studies are now being worked into future versions of math libraries on the Trinity platform.

B. *Proxy Applications*

The development and use of proxy applications for Trinity and future ATS machines provides key advantages. The proxy application is designed to be freely distributable to vendors, collaborators, other laboratories, and universities. The design of the proxy application reflects the design of the larger application that it represents. It is therefore not written to be the most efficient source code for Trinity, but rather to represent the current coding of a larger, portable application.

Quicksilver is an LLNL developed proxy application, which reflects the physics and computer science of a larger Monte Carlo transport code. It was developed as part of the COE effort in preparation for porting code to Trinity and future ATS machines. Quicksilver, and other proxy applications, provide a manageable test bed to explore methods for refactoring a code which may provide for better performance of the larger code across a range of computer architectures. Approaches which prove beneficial will then be implemented in both the proxy application and the larger code.

Quicksilver is a freely distributable code, which solves a simplified dynamic neutron transport problem. At 7000 lines of code, it includes key computer science and physics features: It may be compiled as a pure MPI or a hybrid MPI and threaded code. It contains internal problem and mesh generation. Mesh elements have 14 nodes and 24 faces. Mesh elements are single material, and are assigned to domains using Voronoi decomposition. Any number of domains (MPI ranks) are supported. The default problem is a cubic source in the corner of a cubic domain, but user defined geometry may consist of any number of bricks and spheres, each with its own material properties.

Quicksilver supports three reaction types: elastic scattering, absorption, and fission. However as it is desired to simulate materials with any number of reactions to investigate Trinity hardware features and compilers, multiple copies of the three basic types may be replicated so that each material may have any number of reactions. Isotopes are used as an additional key to the cross section data. They have no physical meaning, but allow for the proxy app to simulate the larger code. Any number of isotopes are supported. Cross section data is completely synthetic, users may specify an arbitrary fourth degree polynomial (in energy) to define cross sections. The number of groups in the cross section tables and the source strengths may be set at run time.

There are potential disadvantages of proxy applications. As proxy applications are simplified from the original, they do not completely represent the physics and computer science of the original application. Memory access, cache usage, high-bandwidth memory usage, threading, and vectorization are difficult to exactly replicate in a smaller code. An approach which looks promising in Quicksilver, may not perform as well in the larger application. There is also a cost to develop and maintain the proxy application. Still, evidence from proxy applications used in other COE efforts show that the lessons learned will pay back this investment.

C. Early Emulation Environments to Improve Levels of Vectorization

A key determinant of performance for both the Haswell and Knights Landing parts of Trinity is ensuring high levels of vectorization (we note that a high proportion of compute performance comes from both processor types having dual-vector processor units). Since our users did not have access to KNL hardware, we utilized Intel's Software Development Environment (SDE) for early programming environment development [18]. While SDE does not provide accurate performance timing information, the analysis information provided does allow for detailed instruction breakdowns and memory access behavior. In Figure 13 we show early evaluations of the use of AVX-512 instructions in unmodified/unoptimized codes for a range of mini-applications and benchmarks compiled to run on Knights Landing. The use of wide FMA, gather and scatter instructions are new for both Haswell and KNL micro-architectures, AVX-512 is new for KNL only. We have provided the SDE information back to our development activities, ahead of having access to KNL, to ensure we focus on, and improve, the levels of vectorization being achieved.

D. Early Evaluation of Compiler Toolchains

During the last three years, research staff preparing codes for Trinity systems have been engaged with both Intel and Cray compiler teams to ensure that bugs in the compiler tool chains and resulting code performance are addressed earlier than in previous procurements. There has been a particular focus on OpenMP runtime performance for Haswell, and work is starting to perform similar assessments on Knights Landing test systems. We are pleased to report significant improvements in support for C++, particularly C++11, on the most recent edition of the Cray compiler environment. Using the full multi-physics applications have also helped to identify compiler issues that were only exposed when compiling very large and complex applications. Very lengthy (hours) compile and link times and reduced performance optimization are a common issue for the larger codes. Improvements in the compiler's ability to optimize

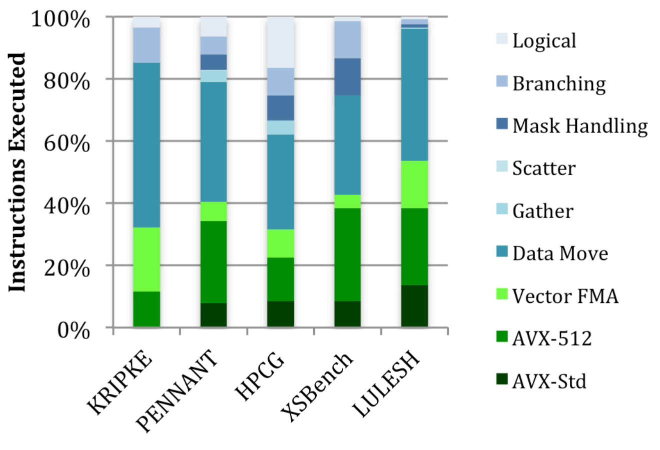

Figure 13 - AVX-512 Vectorization Levels in DOE Benchmarks and Mini-Apps

through multiple layers of abstractions have significantly benefited the overall performance.

E. Next Steps

Early COE activities created much needed collaborations between vendor SMEs and DOE application developers and paved the way for more rapid progress. COE activities will escalate over the next couple of years as access to Knights Landing processors becomes available, the programming environment and tools provide full KNL functionality and performance and become more stable, and Trinity is finally delivered and open for production simulations. Code teams are carefully balancing performance optimizations targeted for Trinity, while keeping an eye toward performance portability, so that codes can utilize ATS-2 'Sierra', a GPU-enabled system in the 2018 time frame [21], and future exascale systems in the 2023 time frame[22].

VII. ACKNOWLEDGEMENT

THIS WORK WAS CARRIED OUT UNDER THE AUSPICES OF THE NATIONAL NUCLEAR SECURITY ADMINISTRATION OF THE US DEPARTMENT OF ENERGY AT LOS ALAMOS NATIONAL LABORATORY SUPPORTED BY CONTRACT NO DE-AC52-06NA25396 AND SANDIA NATIONAL LABORATORIES SUPPORTED BY CONTRACT NO DE-AC04-94AL85000..

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] N. Wichmann, C. Nuss, P. Carrier, R. Olson, S. Anderson, M. Davis, R. Baker, E. Draeger, S. Domino, A. Agelastos, and M. Rajan, "Performance on Trinity (a Cray XC40) with Acceptance-Applications and Benchmarks," CUG 2016, May 8-11, 2016, London, UK.

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

[11] http://pubs.cray.com/#/Collaborate/00258285-FA/FA00258300/Distributed Namespace (DNE) Feature

[12] http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/enhanced-mca-logging-xeon-paper.pdf

[13] http://www.intel.com/content/dam/support/us/en/documents/ssdc/ssd-software/Intel_SSD_DCT_3_0_x_User_Guide.pdf

[14] https://github.com/trilinos

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

[16] http://pubs.cray.com/#/Collaborate/00258285-FA/FA00258300/Distributed Namespace (DNE) Feature

[17] http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/enhanced-mca-logging-xeon-paper.pdf

[18] http://www.intel.com/content/dam/support/us/en/documents/ssdc/ssd-software/Intel_SSD_DCT_3_0_x_User_Guide.pdf

[19] Bach, M., Charney, M., Cohn, R., Demikhovsky, E., Devor, T., Hazelwood, K., Jaleel, A., Luk, C.K., Lyons, G., Patil, H. and Tal, A., 2010. Analyzing parallel programs with pin. *Computer*, *43*(3), pp.34-41

[20] https://github.com/kokkos

[21] https://asc.llnl.gov/CORAL/

[22] [http://www.exascaleinitiative.org/pathforward