Trinity: Opportunities and Challenges of a Heterogeneous System

K. Scott Hemmert, Stan G. Moore, Michael A. Gallis
Sandia National Laboratories
Albuquerque, NM, USA

Mike E. Davis, John Levesque
Cray, Inc.

Nathan Hjelm, James Lujan, David Morton, Hai Ah Nam, Alex Parga, Paul Peltz Jr., Galen Shipman, Alfred Torrez
Los Alamos National Laboratory
Los Alamos, NM, USA

***Abstract*—This paper describes the high-level architecture, success and challenges with Trinity, the first DOE ASC Advanced Technology System. Trinity is a Cray XC40 supercomputer that was delivered in two phases: a Haswell first phase and Knights Landing second phase. We focus on the installation and acceptance of the KNL partition, as well as the merge of the system into a single heterogeneous supercomputer.**

Keywords-HPC

I. Introduction

Trinity is the U.S. Department of Energy's (DOE) Advanced Simulation and Computing's (ASC) first Advanced Technology System (ATS). The system was procured 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 was installed at LANL.

Trinity supports the largest, most demanding applications and will be used by all three nuclear weapons laboratories. The Advanced Technology Systems (ATS) are first-of-a-kind systems that identify and foster technical capabilities that are beneficial to ASC applications. Trinity is a heterogeneous Cray XC40 supercomputer, which was delivered over two phases. Phase 1, comprised of Intel Xeon E5-2698v3 (Haswell) compute nodes was delivered in 2015, and Phase 2, which added Intel Xeon Phi 7250, Knights Landing (KNL) compute nodes, was delivered in 2016.

Our previous work describes Trinity including architecture and early experience [1], focusing on the Haswell partition delivered in Phase 1. Here we describe experiences on Trinity's KNL partition, merging the two partitions, and Trinity as a unified heterogeneous platform. As a first-of-a-kind system, Trinity presented many unique challenges and opportunities, which will be discussed in this paper.

II. Architecture Overview

The Trinity platform operated as two independent systems until June 2017, when the two partitions were merged into a single heterogeneous platform. The high-level Trinity architecture is shown in Figure 1. Table 1 presents the specifications for each partition.

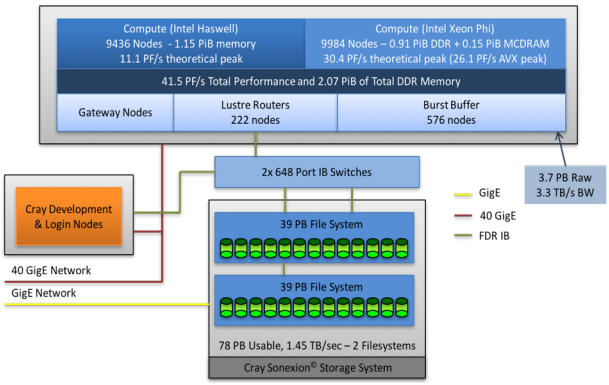

Figure 1: Trinity high-level architecture

Table 1: Trinity Phase 1 and Phase 2 specifications

Phase 1 (Haswell)	Phase 2 (KNL)
9436 nodes	9984 nodes
Intel Xeon E5-2698v3	Intel Xeon Phi 7250
Dual socket 16 cores/socket 2.3 GHz	Single socket 68 cores/socket 1.4 GHz, > 3TF/KNL
128 GB DDR4	96 GB DDR4 + 16 GB HBM
1.2 PB total on-node mem	1.11 PB total on-node mem

The system is made up of nearly 20,000 nodes, split roughly evenly between the two node types. The 9436 Haswell nodes each consist of two processors, each with 4 memory channels and 64 GiB of memory. The Haswell CPUs each have 16 cores, which support the AVX2 vector instruction set and two hyperthreads per core. The 9984 KNL nodes each have a single

socket processor and 96 GiB of DDR4 memory across 6 memory channels and 16 GB of MCDRAM high-bandwidth memory (HBM). The KNLs each have 68 cores with 4 hardware threads per core and support the AVX-512 vector instruction set. The system has a peak theoretical performance of 41.5 PF/s and 2.16 PiB of total DDR memory, plus an additional 0.16 PiB of MCDRAM on the KNL sockets.

Trinity also presents a two-tier storage system to the user: a fast burst buffer tier enabled by solid state storage (SSD) and a traditional high capacity tier based on Lustre. The burst buffer tier is implemented using 576 Cray DataWarp nodes, which, in aggregate, provide 3.7 PB of raw storage and a peak 3.3 TB/s of bandwidth. The Cray Sonexion Lustre filesystem has 222 Lustre router nodes and is backed by 78 PB storage, with a peak of 1.6 TB/s aggregate bandwidth divided evenly across two filesystems.

A. Phase 2 KNL Acceptance

The Phase 2 Trinity KNL acceptance occurred separately from Phase 1 Haswell acceptance. An extensive description of Trinity acceptance testing is described in [1] with results from Phase 1 presented in [2] and Phase 2 presented in [3].

Application performance at scale is paramount and is a key Trinity acceptance requirement. A key figure used to gauge performance at full scale is the Capability Improvement (CI) metric, computed as an average improvement in performance of three ASC applications: PARTISN, Nalu, and Qbox. For Phase 2, Trinity achieved a CI metric of 9.36, exceeding the 8x improvement required for acceptance.

B. Early KNL Open Science Results

After the KNL partition was accepted at the end of 2016, eight application teams were selected as early users and were given access to the system from February through May of 2017 to run 'Open Science' applications for scientific research and to further stabilize the system before it was moved into a more secure environment. Prior to entering the Open Science period, code teams worked to optimize their applications for the KNL architecture and participated in Trinity Center of Excellence activities described in Section 5.

Despite the short Open Science period, teams from LANL, SNL and LLNL were able to demonstrate performance and scalability of their codes and produce scientific results. In addition, two code teams submitted 2017 Gordon Bell submissions.

Applications and referenced results include; Vector Particle-In-Cell (VPIC) [4], used to model kinetic plasmas in 3D; CTH [5], an Eulerian hydrocode developed at SNL for studying shock wave propagation and material deformation; Mercury [6], a Monte Carlo particle transport code used to simulate radiation transport through urban environments; and LAMMPS [7], a classical molecular dynamics code.

III. System Merge

Following the Open Science Period, the two partitions were taken offline on June 1, 2017 and merged into a single heterogeneous system. The merging of the system introduced new capabilities in the areas of system software and I/O, along with introducing heterogeneity across the compute nodes with all nodes now connected with one Aries network fabric. The absolute number of LNET routers and DataWarp nodes on the single fabric was also increased as these nodes were distributed equally across each of the phases before the merge. The merge effectively doubled the size of the two individual Aries networks, which made the complete Trinity system the largest Aries network that Cray has deployed.

The effort to do this required a significant amount of coordination between Cray and ACES to prepare for and plan the strategy for hardware and software merging, and testing of the entire platform after the system was in its final configuration. The demand to put the system into production resulted in an aggressive schedule. Merge steps included re-cabling, system reconfiguration, system booting, initial functionality testing of major subsystems, followed by application performance testing. A short system reliability test was then performed and the Trinity merged system and returned to users on July 6, 2017. In addition to merging the two halves, the job scheduler was also changed from Adaptive's Moab-Cray's ALPS to a native Slurm implementation and Cray's *hugepages* feature was enabled as default.

A. System Software

The Phase 2 portion of Trinity was initially delivered and deployed as a standalone system, with a separate network and dedicated filesystem. Since the two systems were running independently, they both had separate management systems and configurations that needed to be merged into one. The Phase 2 management system was detached from Phase 2 and the Phase 1 management system took control of the

entire merged platform. This required a great deal of effort, but a large portion of this work was staged and completed before taking the systems offline. The ACES integration and production teams were able to stage most of the changes to their version control system and have the changes ready for integration into the system while the hardware teams were performing initial re-cabling of the network to extend the Aries across the entire platform.

To further complicate the joining of the two systems, there was a desire from the user community to convert the job scheduler from Moab to Slurm. There were many reasons for this decision, but one of the largest factors was to converge upon a common scheduling environment across the DOE Tri-Labs. At the time of the merge, LANL was early in the process of deploying Slurm on the commodity clusters and on the smaller testbed Cray systems. Implementing Slurm on Trinity compared to smaller commodity clusters required development of highly specific configurations.

In order to reduce risk, we initially tested these configurations on the Phase 2 Trinity KNL partition before the actual merge. Since the KNL partition was in the open environment, we could easily send logs and diagnose issues more rapidly with Cray and SchedMD. To minimize downtime of the Phase 2 system while the Open Science period was ongoing, the integration team devised a method to rapidly swap between Moab and Slurm, which did require a system reboot to change configuration.

A dual-boot configuration was developed that allowed ACES to do multiple full-scale tests of the system running Slurm within a scheduled downtime while other maintenance activities could still be performed in parallel. This approach allowed Slurm Trinity configurations to be developed and tested in parallel with the Open Science production use and proved very useful in minimizing merge time dealing with Slurm issues.

1) Scheduler challenges

The transition from Moab-ALPS to Slurm created new challenges for users, and highlighted the tight integration between ALPS and Cray platforms. Challenges included:

Loss of transparent process placement functionality on Haswell and KNL nodes provided by ALPS: Although equivalent functionality is provided by Slurm, it is less transparent. Significant effort was required to help users find the right Slurm options to provide the same functionality as ALPS. These issues were catalogued and documented and continue to be a pain-point for novice users.

PMI_TIMEOUT error for large-scale application launches: In the case of ALPS, the user's binary is distributed to every node in the allocation before execution and job launch does not occur until all nodes are ready; this action is transparent to the user. The time for Slurm to prepare the nodes was not aligned with the Cray system application launch resulting in PMI_TIMEOUT errors that perplexed users. Slurm does provide a tool called *sbcast*, which the user can explicitly use to perform a fast distribution of specified files to nodes. The use of this tool and increasing the PMI_TIMEOUT setting helped mitigate the problem, however a long-term solution is still under discussion with Cray and SchedMD.

2) Scale and memory challenges

In addition to scheduler challenges, other issues manifested after the merge that required in-depth analysis and debugging to address. These problems were not deemed critical enough deny users access to the system; however, they did create some system usability restrictions, such as reducing the allowable job size to 2K nodes and not allowing large KNL mode changes.

One issue that highly impacted the schedule was a repeatable large-scale application crash at 8K nodes that required rebooting all of the associated compute nodes. This failure scenario was extremely disruptive to the entire system and could have negative consequences to the performance of other applications or bring down the entire system. Only recently was root cause attributed to the interaction of disabling the kernel's *vm.overcommit* behavior and Cray's *hugepages* implementation, which was not accurately accounted for by the kernel. This improper tracking of memory allows an application to over-allocate memory and crash the node; therefore, vm.overcommit was enabled until a fix was delivered from Cray to correct this behavior. Identifying out-of-memory as the issue for this was further complicated by the change to Slurm, which was not reporting kernel error messages to the user as previously provided through ALPS.

B. Parallel File System (PFS)

Trinity has two scratch Lustre-based Cray Sonexion 2000 file systems that were delivered with Phase 1, but could not be fully tested until the merge. Each scratch file system has 39 PB of capacity. The merge incorporated 108 additional LNET routers that

provided the necessary network bandwidth to maximize throughput to the file system. Highly tuned benchmarks demonstrated file system performance that met requirements. All KNL results were obtained while the KNL nodes were configured in quad-cache mode.

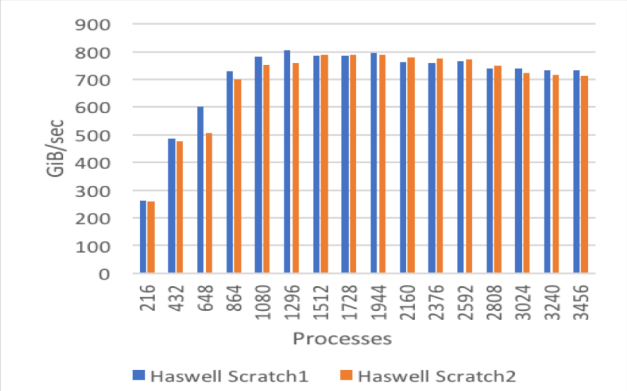

Figure 2: PFS Optimal Write Performance, Haswell 2 PPN

Figure 2 shows I/O write performance for an IOR N-N benchmark tuned for maximum performance on Haswell processors gathered during merge testing. This benchmark was run to show file system performance capabilities using a timed run and large transfer sizes (64MiB) to each of the scratch file systems. The benchmark was run on the Haswell partition with 2 procs per node, and the node counts were specified based on streams per Lustre OST, therefore 216 processes imply 108 nodes on Haswell (2 ppn) for 1 stream per OST (216 OSTs per file system). Write performance peaks at approximately 806 GiB/sec

Figure 3 compares the benchmark results running on both Haswell and KNL following the software upgrade of Trinity (UP05, SU26) that accompanied the merge. As mentioned above, this optimal benchmark was run with 2 procs per node on Haswell. KNL procs per node was set to 4. A process count of 3456 implies 1728 nodes for Haswell and 864 nodes of KNL with 16 streams per OST (3456/216). This graph shows Haswell performance peaking at 772 GiB/sec at 1944 processes (972 nodes) and KNL peaking at 738 GiB/sec at 3456 processes (864 nodes).

The most recent results indicate lower Haswell performance than the baseline result shown in Figure 2. As of this writing, an investigation is underway to try and pinpoint the 7% degradation in performance.

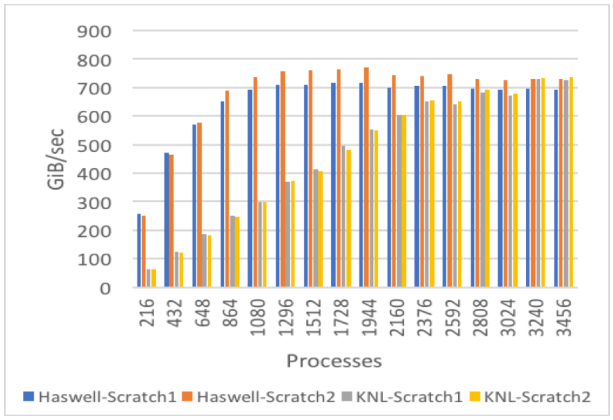

Figure 3: PFS Optimal Write Performance, Haswell 2 PPN, KNL 4 PPN

Figure 4 compares Haswell and KNL metadata performance from *mdtest* file create, stat, and delete rates. Similar performance was observed on previous Haswell metadata test runs. Haswell and KNL performed within 5%-10% of each other with the exception of delete at 5 metadata servers, which had a 20% difference.

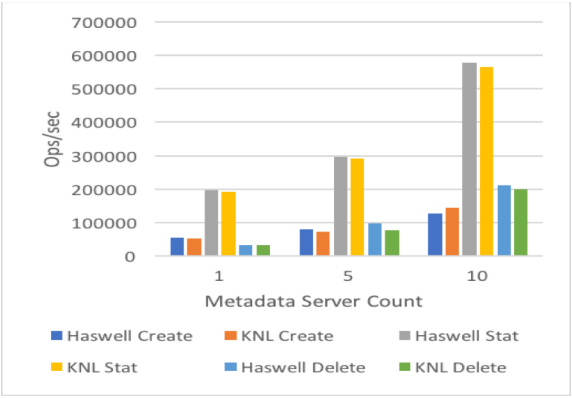

Figure 4: Haswell and KNL Metedata Performance

Additional IOR benchmarks were run against the file system to show both KNL N-N and N-1 write performance. Figure 5 shows N-N and N-1 performance for both Haswell and KNL using proc per node counts of 2 for Haswell and 4 for KNL. For N-N, IOR was run using 1MiB transfer sizes with each process writing 2 GiB in total. For N-1, Each process wrote utilizing a block size of 8MiB in a strided fashion for a total of 1GiB per process. The target directory was configured with a stripe size of 8MiB and a stripe count of 206 in order to optimize shared file performance.

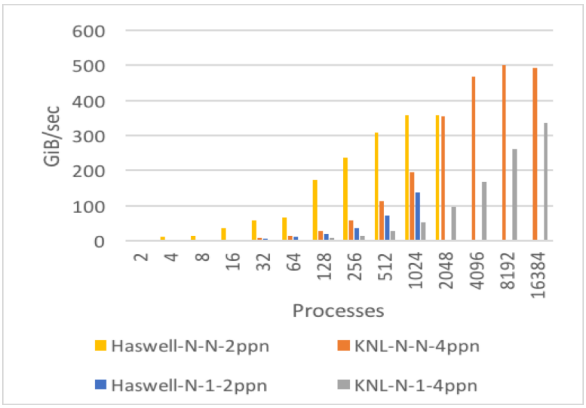

Figure 5: Haswell and KNL PFS Write Performance

Although the graph is incomplete it does show performance scaling for Haswell and KNL N-N up to 2048 processes for Haswell and 16384 processes for KNL. Haswell and KNL N-1 performance is increasing with KNL reaching 300 GiB/sec at 16384 processes (4096) nodes. Further testing is required to determine peak values.

C. Burst Buffer

An extensive discussion of Trinity's Burst Buffer design, integration and performance across 300 DataWarp nodes from Haswell nodes is presented in [1]. The KNL partition provided an additional 276 DataWarp nodes to complete the 576 DataWarp nodes available on the heterogeneous Trinity system today. As noted in [1], DataWarp functionality is tightly coupled to the Work Load Manager (WLM), which was changed from Moab to Slurm during the merge. Although most functionality has been re-established after the merge to stage-in and stage-out data, support for several complex workflows involving chained jobs are still unavailable under Slurm and are actively being developed.

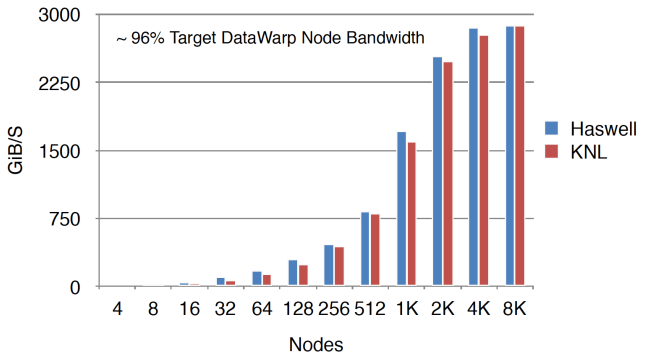

Figure 6: DataWarp Write, N-N
4 PPN, 128 GiB/node, 1MiB xfer

Performance of the Trinity burst buffer from both Haswell and KNL nodes is presented in Figures 6-9 using the IOR benchmark. Figures 6 and 7 show file per process (N-N) read and write performance. Figures 8 and 9 show shared file (N-1) read and write performance. A particularly attractive feature of the burst buffer seen in each Figure is the nearly identical I/O performance from Haswell and KNL nodes for both N-N and N-1 I/O, which was not seen from the parallel file system.

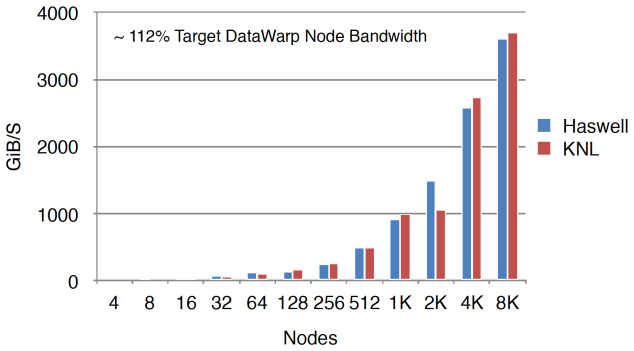

Figure 7: DataWarp, Read, N-N
4 PPN, 8GiB/node, 1MiB xfer

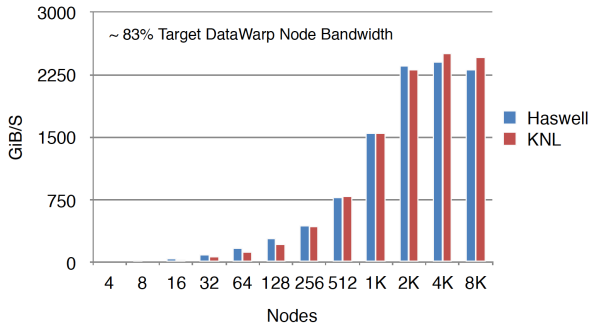

Figure 8: DataWarp, Write, N-1
4 PPN, 128 GiB/node, 1MiB xfer

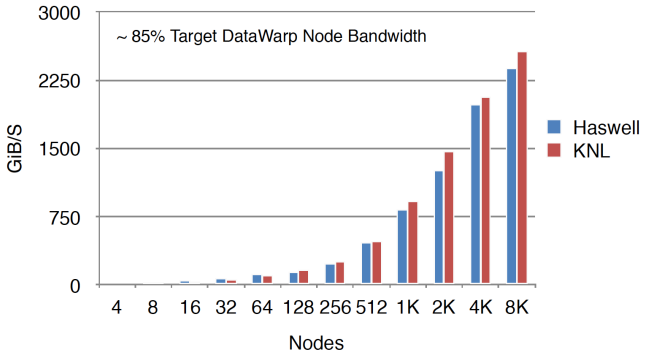

Figure 9: DataWarp, Read, N-1
4 PPN, 8 GiB/node, 1 MiB xfer

IV. Application Performance

Trinity's heterogeneous system architecture, with substantial Haswell and KNL partitions, creates new opportunities for code teams to create complex workflows or run at large scales. However, exploiting this opportunity also comes with challenges described in this section.

A. Hybrid HPL/HPCG

Trinity is the first system to reach the top three on the HPCG Top500 benchmark on a heterogeneous system using different types of compute nodes. The run that achieved this standing was conducted in October 2017 and reported a performance of 0.546124 Petaflops.

The version of HPCG used was a variant of the Intel MKL 2016 HPCG provided by Alexander Kalinkin of Intel. It was based on a variant of HPCG 2.0 provided by Mike Heroux of Sandia National Laboratories. This variant of HPCG included a feature called "2-region split." With this feature, the global grid is split into two regions along the Z dimension, governed by runtime parameters pz, zl, and zu, such that the npx*npy*pz ranks in region 0 have a local process grid of size nx*ny*zl and the npx*npy*(npz-pz) ranks in region 1 have a local process grid of size nx*ny*zu. The values of zl and zu are chosen to balance the computation between the two processor types hosting the ranks in each region.

The benchmark was run on 9372 Haswell nodes and 9906 KNL nodes. The two regions were formed from these nodes as follows: in region 0 were 9372 Haswell nodes with 2 ranks per node, each with 16 OpenMP threads, and 267 of the 9906 KNL nodes with 2 ranks per node, each with 34 OpenMP threads; in region 1 were 9639 of the 9906 KNL nodes with 4 ranks per node, each with 34 OpenMP threads. The HPCG parameters used for the run were npx=27, npy=42,npz=51, nx=160, ny=160, zl=88, zu=136, pz=17. The local process grids were sized to fit optimally within the HBM of the KNL nodes. See Figure 10, where an HPCG global grid with npz=3 and pz=1 is exploded in the Z dimension.

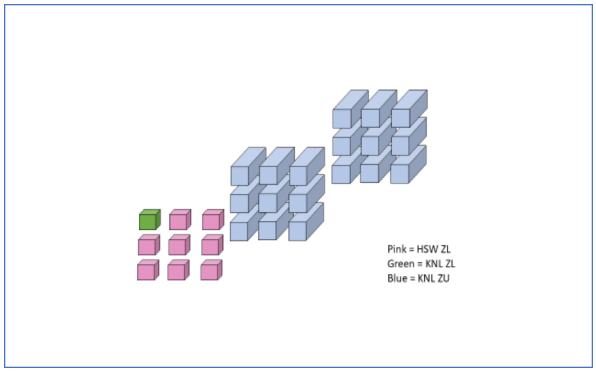

Figure 10: HPCG Global Grid

A special driver program (called *multiprog_driver*) and an MPI_Init wrapper function were developed to facilitate, respectively, the launching of the benchmark across the heterogeneous nodes and the initialization of the processes for their respective environments. To optimize performance, a different HPCG executable had to be run on each of the two different node types: *xhpcg_avx2* for the Haswell architecture and *xhpcg_avx512* for the KNL architecture. Also for the KNL architecture, it was beneficial to invoke the HPCG executable from within the numactl command, to bind the memory images for the processes to the proximal quad-flat NUMA regions. The multiprog_driver handled these launching details.

Because (this version of) the HPCG 2-region split feature did not properly derive nz from either zl or zu as appropriate, each process had to tailor its nz argument to its region assignment; for this run, region-0 processes used nz=zl=88 and region-1 processes used nz=zu=136. In addition, the number of OpenMP threads created by each process depended on the type of processor hosting the process. Finally, different rank/thread placement strategies had to be employed on the different node types, including strategies (such as different rank-per-node values for different nodes) that are not supported by some workload management systems. The MPI_Init wrapper function handled these details.

HPL was also run across both the Haswell and KNL nodes, but was unable to complete execution.

B. Hybrid SPARTA

A production application heterogeneous run on the full Trinity system was performed using Sandia's SPARTA code during March 2018. Over 19,000 nodes (9200+ Haswell and 9900+ KNL) and 1.2 million MPI processes were used. While several challenges were encountered, the run was successful, with SPARTA running for several hours.

SPARTA is a Direct-Simulation Monte-Carlo (DSMC) code [8,9]. The DSMC method resolves fluid length scales at the particle level and accurately models high-altitude hypersonic re-entry flows. DSMC can be used for non-equilibrium, non-

continuum conditions that cannot be simulated with traditional CFD or reproduced experimentally. In addition to re-entry, SPARTA has also been used to model spacecraft, fluid instabilities, turbulence, and porous media.

The turbulent energy evolution for an argon flow at a Reynolds number 500 and a Mach number 0.3 (at the limits of compressibility), was studied during the run. A similar flow has been studied previously by Gallis et al [10] using the Sequoia supercomputer at LLNL. In the incompressible limit, the DSMC simulations were found to agree with corresponding Navier-Stokes Direct Numerical Simulation (DNS) results. The computational power of Trinity offers the possibility of further extending molecular calculations in the compressible regime, where DNS simulations may lose accuracy due to sharp gradients. Thus, molecular simulations will allow new insights into turbulence by directly linking molecular relaxation processes to macroscopic transport processes.

For the full Trinity run, the same parameters as were run on Sequoia were used to verify correctness and compare performance between the two machines. A 3D grid with 8 billion grid cells (2000 x 2000 x 2000) was used, with approximately 45 particles/cell, giving at total of 360 billion particles. The particular case studied was a very well load balanced case. The nominal (excluding statistical fluctuations in the number of simulators) workload between cores did not vary more than 3%. Figure 11 presents the energy and energy dissipation as a function of time from the Sequoia and the Trinity runs. Both time and energy have been normalized. We note that the results for this particularly challenging problem (for a DSMC code), are practically identical. The performance on full Trinity was approximately 8 timesteps/s, while the performance on one-third (32K nodes) of Sequoia using 16 MPI tasks/node (no OpenMP or MPI running on the hyperthreads) was approximately 1.6 timesteps/s, or a projected 4.7 timesteps/s on the entire Sequoia machine (ignoring additional MPI overhead from scaling 32K nodes to 96K nodes). Including OpenMP on the hyperthreads is expected to speed up the Sequoia results by around 1.5x to 2x, but exact numbers are not currently available.

To run on both types of nodes simultaneously, separate executables were compiled for each node type. The KNL executable used Sandia's Kokkos performance portability library [11] to support OpenMP multithreading, along with AVX-512 instructions. The Haswell executable used the Kokkos

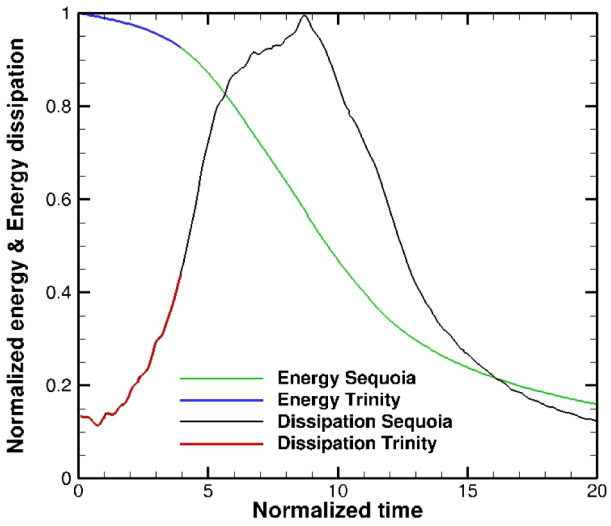

Figure 11: Energy and energy dissipation for a Re=500, Ma=0 Taylor Green Argon flow

Serial backend (no OpenMP support), along with AVX2 instructions. The multiprog driver program (described previously) was then used to determine the node type and launch either the Haswell or KNL executable, based on node type. A total of 64 MPI ranks were used on both node types. For Haswell, MPI ranks were placed on the hyperthreads, while for KNL, 4 OpenMP threads per MPI rank were placed on the hyperthreads to fill the node with tasks. Using 64 MPI ranks on Haswell and 64 MPI x 4 OpenMP threads on KNL has been found to give reasonable performance with SPARTA.

Unfortunately, the version of the Slurm workload manager on Trinity did not support running separate srun commands on the different nodes, so the affinity options to srun were shared between Haswell and KNL. The option used was "--cpu_bind=core". This led to slightly suboptimal affinity for one node type but was a necessary compromise.

Initially, SPARTA performance on full Trinity was much slower than expected. Built-in timers showed that most of the time was spent outside of normal computation (such as the particle move, sort, and collide algorithms) and was instead in the "other" category. This suggested a severe load imbalance issue. Using static load balancing showed better performance when most of the work was shifted from the Haswell nodes to the KNL nodes, but prior benchmarking on the Mutrino Cray testbed showed that the KNL and Haswell nodes were less than 2x

imbalanced. This suggested a very slow Haswell node in the system. Timers were added to the code to find the MPI process with the largest time spent in particle move. The *nid* of the slow node was determined, and the node was excluded from the allocation. This process was repeated until two slow Haswell nodes were eliminated, and then the simulation ran over 20x faster, giving the expected performance. If even one node is running slow, the performance of the entire simulation will significantly degrade due to MPI barriers or other synchronization points in the program. A small standalone MPI program has been written to easily determine slow nodes and will be used in future runs prior to full application launch.

V. 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 applications to Trinity. COE activities begin roughly 2 years prior to system acceptance and continue 2 years after acceptance for maximum impact to code performance on a new platform.

An extensive description of COE activities and benefits is described in [12]. Activities include training and workshops, application deep-dives, and SME integration into daily code team development activities. After Trinity was put into production in a secure environment, it was essential for our vendor COE partners to integrate with the code teams to address scaling and optimization issues on the full system with the problems of interest. This section describes several performance successes through COE activities in the LANL xRAGE application, a radiation-hydrodynamics code.

A. Scalability improvements of AMR

xRAGE uses adaptive mesh refinement (AMR) and makes extensive use of AlltoAll and Allreduce communications to build communication pairs during mesh reconfiguration. This occurs at every cycle since neighbors can change. Working closely with the COE, we were able to identify this significant scaling bottleneck using the CrayPAT profiling tool. After identifying the source of the bottleneck, we were able to implement an RMA-based solution that takes advantage of the sparsity and caching, and incorporated an asynchronous barrier optimization allowing for overlap of synchronization and use of the RMA window.

The use of RMA-based communications was inspired early in the COE lifecycle through a COE seminar focused on disseminating best practices and new optimization approaches.

Communication between vendor COE members to Cray software developers of the proposed solution also resulted in an optimized Cray MPICH for this use case that was incorporated into their December programming environment release. This resulted in a 3x performance improvement at scale to the total run time. Figure 12 shows weak scaling of xRAGE before and after the code improvements.

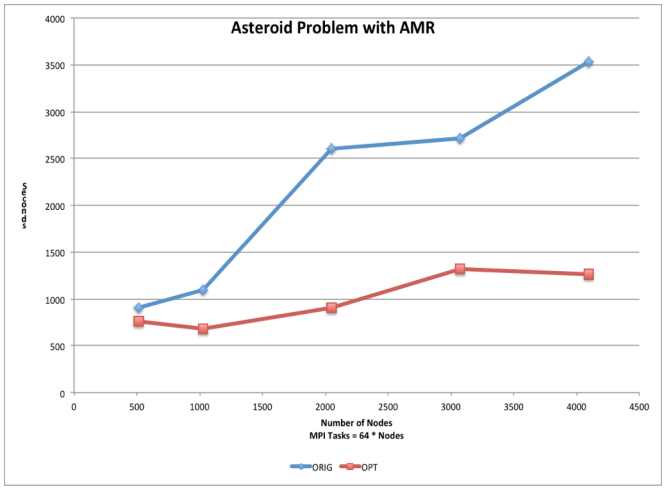

Figure 12: Weak scaling for an xRAGE AMR problem before and after RMA-based communication improvements.

B. Load imbalance

Profiling is a valuable activity that should be repeated, despite achieving performance successes. Improvements to one area of the code reveal other inefficiencies previously hidden. In a multi-physics code such as xRAGE, mapping between mesh types is required. xRAGE generates mappings between the AMR mesh and a structured mesh. At each cycle the AMR mesh changes and the mapping must be rebuilt. With improvements to the communications, the mapping now took up to 60% of runtime.

Identifying the performance issue was challenging, particularly due to the scale of the problem. Working with the COE and using CrayPAT we identified a load imbalance in the calculation of the topology intersections where a handful of ranks were taking a degenerate code path. The code team then implemented an in-place heap sort of the mesh topology and a binary search to significantly reduce the load imbalance. This

resulted in a reduction of this section of the code to only 8% of the runtime.

C. HIO & Burst Buffer

Checkpoint files and analysis/visualization files are integral to long-running simulations such as xRAGE to recover from node failures and track progress. xRAGE checkpoint files can grow to 24 TB each and sometimes larger. Using Trinity's burst buffer and the HIO high performance I/O library, xRAGE's read/write times were significantly improved. Table 2 shows measured read and write speeds for xRAGE 24TB checkpoint files to the parallel file system (Lustre) and the burst buffer (BB) with HIO. Read performance improved by over 4x and write performance improved by over 16x.

Table 2: xRAGE measured read/write bandwidth for 24 TB checkpoint files

	24 TB Read	**24 TB Write**
Lustre	16 Gbytes/sec	21 Gbytes/sec
BB w/ HIO	69 Gbytes/sec	350 Gbytes/sec

VI. Conclusions

The complete heterogeneous Trinity supercomputer, with both Haswell and KNL nodes, is delivering valuable production computing time to the NNSA to solve significant problems and provide new opportunities for scientific investigation. The journey of deploying Trinity and the challenges presented by the scale and complexity of the system are becoming the norm as each generation of supercomputer pushes the limits. Beyond the initial deployment, each software upgrade to Trinity requires dedicated efforts to ensure performance expectations persist through its lifetime. It is only through close collaborations with our vendor partners that we are able to stand up these systems that are essential to pushing the limits of our scientific knowledge. The works that follow this will focus on the science enabled and less about the system itself.

VII. Acknowledgements

The authors would like to thank the EAP code team for their contributions to the Trinity Center of Excellence. The authors would also like to thank the many dedicated staff from LANL, SNL, LLNL, NERSC, Cray, Intel and SchedMD who spent many long hours making Trinity a success.

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. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

VIII. References

[1] K. S. Hemmert, M. W. Glass, S. D. Hammond, R. Hoekstra, M. Rajan, S. Dawson, M. Vigil, D. Grunau, J. Lujan, D. Morton, H. Nam, P. Peltz Jr., A. Torrez, C. Wright, "Trinity: Architecture and early experience", *Cray Users Group CUG*, May 2016.

[2] 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.

[3] Agelastos, A.M., Rajan, M., Wichmann, N., Baker, R., Domino, S., Draeger, E.W., Anderson, S., Balma, J., Behling, S., Berry, M., Carrier, P., Davis, M., McMahon, K., Sandness, D., Thomas, K., Warren, S., Zhu, T.: Performance on Trinity phase 2 (a Cray XC40 utilizing Intel Xeon Phi processors) with acceptance applications and benchmarks. In: Cray User Group CUG, May 2017.

[4] WD Nystrom, B Bergen, RF Bird, KJ Bowers, WS Daughton, F Guo, A Le, H Li, H Nam, X Pang, DJ Stark, WN Rust III, L Yin, BJ Albright, "Recent Performance Results of VPIC on Trinity," APS Conference poster, 2017

[5] K. Ruggirello and T. Vogler, "Trinity Phase 2 Open Science: CTH", SNLTechnical Report, 2017 http://prod.sandia.gov/techlib/access-control.cgi/2017/178216r.pdf

[6] S. Dawson, "LLNL Mercury Project Trinity Open Science Final Report," LLNL Technical Report, 2017, DOI: 10.2172/1389985

[7] S. Moore, M. Wood, A. Thompson, "LAMMPS Project Report for the Trinity KNL Open Science Period," SNL Technical Report, 2017, http://prod.sandia.gov/techlib/access-control.cgi/2017/178701r.pdf

[8] G. A. Bird, Molecular Gas Dynamics and the Direct Simulation of Gas Flows, Oxford University Press, Oxford, UK (1994)

[9] M. A. Gallis, J. R. Torczynski, S. J. Plimpton, D. J. Rader, and T. Koehler, Direct Simulation Monte Carlo: The Quest for Speed, Proceedings of the 29th Rarefied Gas Dynamics (RGD)

[10] M. A. Gallis, T. P. Koehler, J. R. Torczynski, S. J. Plimpton, G. Papadakis, Molecular-Level Simulations of Turbulence and its Decay, Phys. Rev. Lett. **118**, 064501 (2017).

[11] Carter Edwards, H., Trott, C. R. and Sunderland, D. J., Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Parallel Distr. Com., 74, 12 (2014), 3202-3216.

[12] H. A. Nam, G. Rockefeller, M. Glass, S. Dawson, J. Levesque and V. Lee, "The Trinity Center of Excellence Co-Design Best Practices," in *Computing in Science & Engineering*, vol. 19, no. 5, pp. 19-26, 2017. doi: 10.1109/MCSE.2017.3421542