Preparing NERSC users for Cori, a Cray XC40 system with Intel Many Integrated Cores

Yun (Helen) He, Brandon Cook, Jack Deslippe, Brian Friesen, Richard Gerber, Rebecca Hartman-Baker,
Alice Koniges, Thorsten Kurth, Stephen Leak, Woo-Sun Yang, Zhengji Zhao
National Energy Research Scientific Computing Center (NERSC)
Lawrence Berkeley National Laboratory, Berkeley, CA, USA
Eddie Baron
University of Oklahoma, Norman, OK, USA
Peter Hauschildt
Hamburger Sternwarte, Hamburg, Germany

***Abstract*[1]—The newest NERSC supercomputer Cori is a Cray XC40 system consisting of 2,388 Intel Xeon Haswell nodes and 9,688 Intel Xeon-Phi "Knights Landing" (KNL) nodes. Compared to the Xeon-based clusters NERSC users are familiar with, optimal performance on Cori requires consideration of KNL mode settings; process, thread, and memory affinity; fine-grain parallelization; vectorization; and use of the high-bandwidth MCDRAM memory. This paper describes our efforts preparing NERSC users for KNL through the NERSC Exascale Science Application Program (NESAP), web documentation, and user training. We discuss how we configured the Cori system for usability and productivity, addressing programming concerns, batch system configurations, and default KNL cluster and memory modes. System usage data, job completion analysis, programming and running jobs issues, and a few successful user stories on KNL are presented.**

Keywords-Intel Xeon Phi; KNL; heterogeneous; cluster and memory modes; NESAP; performance optimization; cross compilation; job scripts; process and thread affinity; user support; training; web documentation

I. Introduction

A. Motivation

As the High Performance Computing (HPC) community moves toward exascale, it is being challenged to effectively exploit the energy-efficient manycore processor architectures that will comprise exascale systems. For NERSC [1] users this poses a significant challenge because adapting codes to run efficiently on manycore processors will require a large investment in code optimization and tuning. Codes will have to identify and use additional fine-grained thread parallelism, take advantage of wide SIMD units, and carefully manage data structures to optimally fit into a deep memory hierarchy.

NERSC's newest supercomputer, named Cori [2] after Nobel prize-winning biologist Gerty Cori, introduces the NERSC user community to its manycore future. Cori is a Cray XC40 system consisting of more 9,688 single-socket nodes with Intel Xeon Phi "Knight's Landing" (KNL) manycore processors [3]. The KNL Many Integrated Cores (MIC) architecture features 68 cores per node with 4 hardware threads each, 512-bit vector units, and 16 GB on-package MCDRAM (Multi-Channel DRAM) high bandwidth memory. The KNL nodes offer many configuration options: MCDRAM can be configured for explicit management as one or more NUMA nodes ("flat" mode) or as a transparent cache ("cache" mode), simultaneously the cores and DDR memory can be managed as a single mesh ("quadrant" et al) or as two or four NUMA nodes ("snc2", "snc4") [4]. Cori also has 2,388 dual-socket Intel Xeon "Haswell" nodes on the same Cray Aries high-speed interconnect network. This mix of node types makes Cori a rather novel system at this scale in HPC.

When Cori was conceptualized, NERSC foresaw – through a series of requirements reviews with DOE Office of Science computational scientists and conversations with its users – that users would need assistance to prepare their codes for manycore. The NERSC Exascale Science Applications Program (NESAP) [5], in which code teams were connected to experts at NERSC, Intel, and Cray, was developed to help enable a significant fraction of the NERSC workload to run well on Cori. In addition to in-depth help from NERSC staff and postdocs, NESAP teams were given access to tools, early hardware, user training programs, intensive "dungeon session" tuning sessions with Cray and Intel, and extensive NERSC documentation on best practices for KNL optimization. NESAP and its results are described in Section 2.

In this paper, we discuss how we configured the Cori system for usability and productivity, addressing programming concerns, batch system configurations, and default KNL cluster and memory modes. System usage data, job completion analysis, programming and running jobs issues, and a few successful user stories on KNL are presented. The main focus is on using the Cori KNL architecture more effectively. Using DataWarp and optimizing IO are not part of the scope for this paper.

[1] This paper has been submitted as an article in a special issue of Concurrency and Computation Practice and Experience on the Cray User Group 2017.

B. Cori KNL System Timeline

Preparation for Cori began with NESAP in the Fall of 2014, well before the Cori hardware arrived. NESAP teams used NERSC system Edison [6] (a Cray XC30 with Intel Xeon IvyBridge compute nodes) and various test and development systems to analyze performance and begin optimization.

The first phase of Cori, 1,630 nodes with dual-socket 16-core Intel Xeon Haswell processors, arrived in late 2015. The operating system underwent a major upgrade to CLE6, required to support KNL, in June 2016 as the first phase-two cabinets arrived.

The second phase, consisting of more than 9,300 single-socket nodes with Intel Xeon Phi KNL processors, arrived at NERSC in mid 2016. Following standalone testing, the two phases were integrated over a period of six weeks beginning mid-September 2016. A subset of service nodes in Haswell cabinets were repositioned into KNL cabinets. The regained Haswell slots were filled with more compute nodes, bringing up the total number of Haswell nodes to be 2,004.

NESAP users were given early access to the KNL nodes in November 2016. In January 2017 access to a subset of the KNL nodes was granted to all users for code development, debugging, and optimization. Feedback from the NESAP early access users fed into the Recommendations for Jobs described in section 4 and the KNL Default Mode Selection described in section 5A.

An additional two cabinets (384 nodes total) of KNL were integrated in February 2017, bringing up the total number of KNL compute nodes to be 9,688. An additional two cabinets (384 nodes total) of Haswell were integrated in April 2017, bringing up the total number of Haswell compute nodes to be 2,388.

The initially-limited access for non-NESAP users was prompted in part by the substantial differences between KNL and traditional Xeon multicore processors: without significant investment in code readiness, applications are unlikely to make effective use of the KNL hardware. Beginning in March 2017 we allowed all users full access to the KNL nodes once they had demonstrated some degree of application-readiness, this gating procedure is described in section 5C.

Finally, the machine is scheduled to go into full production mode with none of these restrictions in July 2017.

C. KNL Usage Data

NESAP teams and codes were chosen to give broad coverage to the NERSC workload, representing the spectrum of science fields within the DOE Office of Science mission.

Figure 1 shows the breakdown of hours used on Cori KNL nodes in 2017 through April. Lattice QCD codes were ready to use KNL before many others, VASP and Quantum Espresso Early User Program started in mid January to allow longer wall limit, thus explaining their large usage.

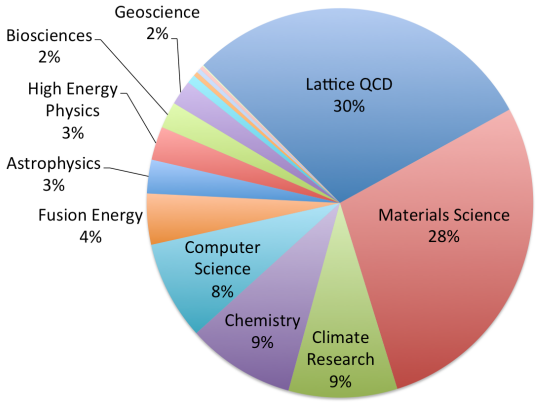

Figure 1. Breakdown of hours used on Cori KNL, January to April 2017.

The workload is highly parallel as shown in Figure 2. About half the hours from January to April 2017 were used by jobs using more than 500 nodes and a quarter of the hours were used by jobs running on more than 1,280 nodes.

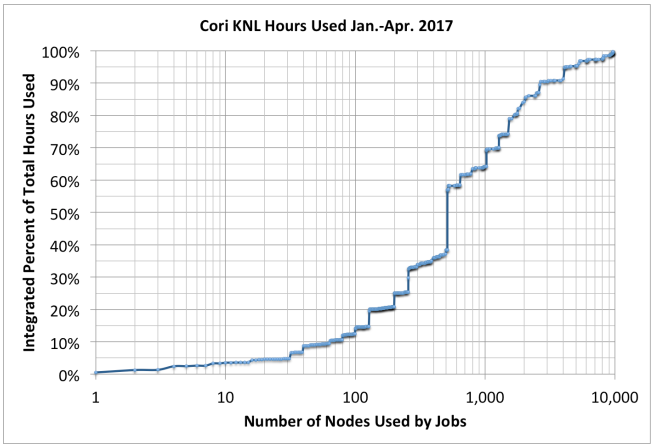

Figure 2. Cori KNL hours used in January to April 2017.

II. NESAP

A. Introduction

To assist users with the transition to Xeon Phi, NERSC established the NERSC Exascale Science Applications Program (NESAP) [5] concurrently with the procurement of Cori. NESAP is a collaboration among NERSC staff and postdocs, application developers, and third-party library and tool developers, with the goal of preparing science applications for the many-core architecture on Cori.

Because NERSC hosts 6,000 users running 700 different applications, devoting staff resources to assist all users with tuning their applications for Xeon Phi is not feasible; instead, NERSC selected approximately twenty application teams [7] to participate in the NESAP program, with domains spanning all six programs within the Department of Energy Office of Science. Approximately 60% of NERSC hours are represented by these NESAP applications, either directly or as proxy codes.

The investment in these application teams has been significant: they have received advanced training in the form of on-site visits from Intel and Cray staff, visits to Intel campuses for intensive hackathons, and on-going collaborations with NERSC and Cray staff as part of a Center of Excellence. In addition, these teams were provided access to pre-production Xeon Phi hardware prior to its delivery with the Cori system. Furthermore, eight of these application teams were paired with NERSC postdoctoral researchers, whose focus would be to profile and optimize these codes for the Intel Xeon Phi architecture on Cori. By selecting a broad subset of applications running at NERSC which run a wide variety of algorithms, and documenting extensively their efforts to port their applications to Cori, the information gleaned from this subset of users would then be disseminated to the rest of the NERSC user base via extensive web documentation and user training.

B. Optimization Strategy and Tools

It is the within-node architecture of Cori KNL nodes that is most disruptive to application performance: Cori uses the same Aries interconnect and dragonfly topology as Edison [6], its predecessor at NERSC.

The KNL processor has different performance constraints and several new or enhanced features that can benefit applications compared to Xeon, such as many cores, more hyperthreads, wider vector units, added instructions and on-package MCDRAM.

To help users navigate this complex optimization space, NERSC developed an optimization strategy that heavily utilizes the roofline performance model [8][9][10][11] developed at Berkeley lab to frame our conversations with code teams. With Cray and Intel staff we developed a methodology for computing roofline ceilings for the Cori system as well as roofline positions for codes and kernels using Intel's VTune [12], SDE [13], and Vector Advisor [14] products.

The novelty of the roofline modeling approach is that a kernel's position on the roofline indicates whether optimizing for effective memory bandwidth (eg via cache blocking or explicit use of MCDRAM) or CPU related improvements (vectorization, improved code generation) will be more profitable. The model also allows us to visualize the effect of code optimization in terms of the theoretical limits of memory bandwidth and compute speed - the principle components which constrain performance.

Figure 3 shows the performance of a specific kernel in the BerkeleyGW [15] application after a series of optimization steps (2 - addition of OpenMP, 3 - loop reordering for vector code generation, 4 - cache blocking, 5,6 - hyperthreading and refined vectorization).

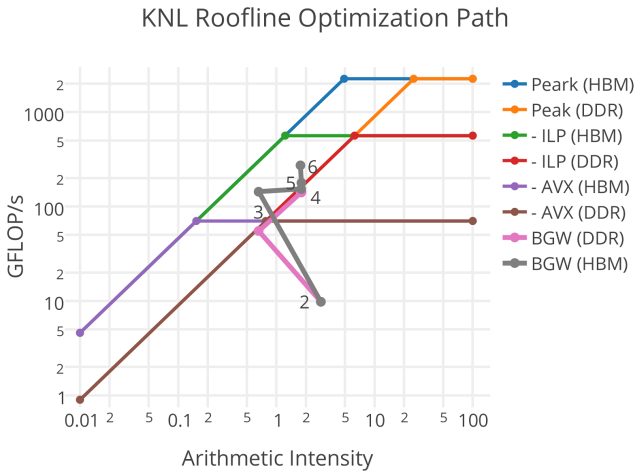

Figure 3. BerkeleyGW kernel optimization path after a series of optimization steps.

C. NESAP Results

A summary of before-and-after NESAP code performance on Edison and Cori is shown in Figure 4. The numbers are collected from multi-node benchmarks representing larger scale science runs being performed by the teams. The runs compare performance on Cori KNL with performance on the same number of Edison (dual socket Ivy Bridge) nodes.

In Figure 4, we set the performance of the baseline code on Edison to 1. The number in parenthesis for each application in X-axis is the number of nodes used. We see an approximate 3x performance increase on average due to optimizations on KNL and more than 1.5x average speedup node-to-node on Cori KNL *vs.* Edison for optimized codes.

We have also looked at the impact of various aspects of the Xeon-Phi hardware towards NESAP application performance. Figure 5 shows the relative speedups when utilizing MCDRAM (in either cache or flat mode depending on the application preference) *vs.* ignoring the MCDRAM (running in flat mode utilizing only the traditional DRAM). In addition, we compare performance when compiling with full-optimization (-xMIC-AVX512) *vs.* disabling vectorization (-no-vec -no-simd). It should be noted that the latter test does not include libraries or AVX intrinsic codes where such compiler flags are ignored.

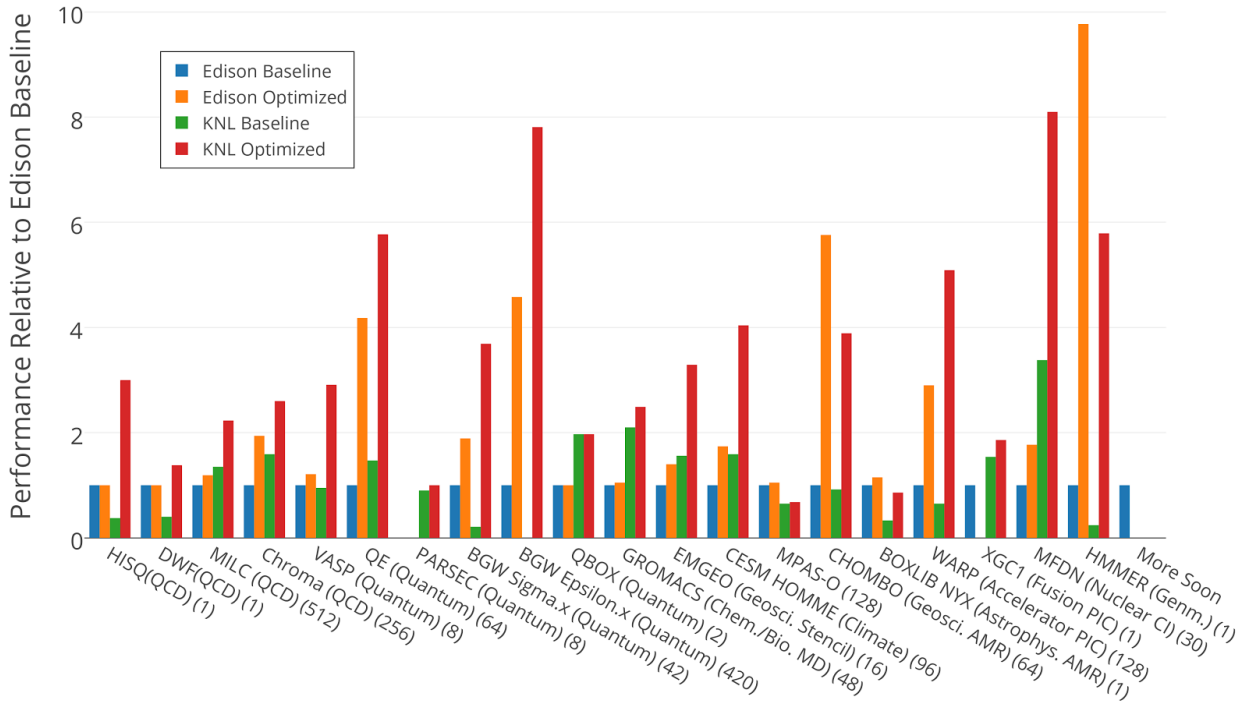

Figure 4. Applications performance relative to Edison baseline on multiple nodes.

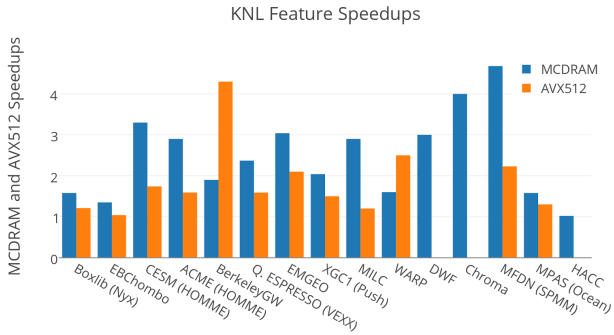

Figure 5. Blue: Applications performance speedup using MCDRAM *vs.* using DRAM. Orange: Applications performance speedup using AVX512 *vs.* using AVX2.

The NESAP optimization efforts and results are documented in more detail in [16][17]

III. Heterogeneous Programming Environment

The heterogeneous Haswell/KNL system is considerably more complex than a homogeneous Xeon cluster. We describe some challenges encountered and the recommendations we formed around building applications with cross-compilation and binary compatibility.

A. Building Software for a Heterogeneous System

When Cori's OS was upgraded in mid June 2016 from CLE5 to CLE6, which was required to support KNL, all NERSC installed software had to be rebuilt.

Since binaries built for Haswell can run on KNL, but not vice versa, Cray builds its software to target Haswell and only builds KNL optimized versions of key numerical libraries that can take advantage of the KNL architecture.

NERSC adopted the same strategy for the many software packages we install for the users. The materials science codes VASP and Quantum Espresso were among the first scientific applications NERSC installed that target KNL architecture. We plan to install more KNL-specific builds of applications and libraries as appropriate.

NERSC is actively working on streamlining the software build process by using the Spack software package manager for building supported packages on all the Cray systems at NERSC, including Edison Ivy Bridge, Cori Haswell, and Cori KNL [18].

B. Cross Compilation

Compute nodes have no local storage and to speed up launch times for large jobs, NERSC configured the system to hold the OS image in memory on the KNL compute nodes. In order to limit the memory used by the OS image, our compute nodes do not have a full build environment, so applications targeting either Haswell or KNL must be built on the Haswell login nodes.

For executables that will run on KNL, this means they should be cross-compiled on Haswell to target KNL. While compiling for Haswell has been set to be the default on Cori, switching the target to KNL is straightforward if the Cray compiler wrappers (ftn, cc, CC) are used: simply swap the "craype-haswell" module with the "cray-mic-knl" module and compile as normal. Although a Haswell binary will run on KNL (but not vice-versa), that binary and the libraries it calls will use compiler optimizations targeting Haswell rather than KNL, so NERSC recommends that the appropriate module be loaded when building applications.

It is also possible to build a merged binary that contains instructions appropriate to both Haswell and KNL by including a special compiler flag "-axMIC-AVX512,CORE-AVX2" for Intel compilers. The overhead in terms of binary size depends on how many extra KNL instruction sets it includes, and the overhead in terms of performance is just the startup check for the run architecture.

A frequently-encountered difficulty when building software targeting the KNL nodes is that in some build systems (such as autoconf or cmake), certain steps need to run a small test program in order to generate a Makefile. This will not work when building on Haswell for KNL since the binary targeted for KNL (with the -xMIC-AVX512 flag) cannot run on Haswell. Our workaround is as follows:

```
% module load craype-haswell
% ./configure CC=cc FTN=ftn CXX=CC ...
% module swap craype-haswell craype-mic-knl
% make
```

This allows the configure script to build for Haswell and test against Haswell and then has the actual build target KNL.

Cray has worked with Kitware to increase the compatibility of CMake with the programming environment on the Cray systems [19]. Beginning with version 3.5.0, CMake has been enhanced to work with the Cray Programming Environment. Cross-compiling can be enabled in CMake via the following steps:

```
% export CRAYOS_VERSION=6
% cmake
-DCMAKE_SYSTEM_NAME=CrayLinuxEnvironment ...
```

In CMake scripts that invoke the *try_run* command, errors occur when cross compiling. The user must edit the TryRunResults.cmake file and replace placeholder values, or else use the CMAKE_CROSSCOMPILING_EMULATOR property (available starting with CMake version 3.6).

IV. Recommendations For Jobs

A KNL node is comprised of 34 dual-core tiles, which in some cluster modes are arranged into 4 non-equal NUMA domains [4]. This complicates task placement and management especially for hybrid MPI/OpenMP applications so we have developed a batch script generator to help users navigate such additional complexity. In many cases, jobs use only 64 of the 68 cores per node. We discuss the reasoning leading to our recommendations and our defaults for NUMA and cluster modes for achieving optimal thread affinity and memory binding, managing hyperthreading.

We use Slurm [20] in "native" mode (i.e., NERSC does not use the aprun job launcher from Cray ALPS) [21][22] to manage batch jobs on the Haswell and KNL nodes as a single system. Slurm features including core specialization, affinity support and sbcast (pre-execution broadcast of key files to compute nodes) are more important for Cori than for previous systems and we discuss our usage of these features.

A. Job Script Recommendations

On previous NERSC systems, a simple job script was sufficient to run a simple job: specify node count and time required then start the desired number of tasks across those nodes. Our Haswell nodes have 32 cores - a convenient power of two - and while hyperthreading is supported, most of our workload does not use it [23]. OpenMP was used in a minority of the NERSC workload [23][24], so 32 MPI tasks per node, evenly distributed, generally gave optimal performance.

Xeon Phi nodes can be configured - on a job-by-job basis - with CPUs distributed over one, two (equal) or four (non-equal) NUMA nodes and with MCDRAM configured as one or more NUMA nodes or a transparent cache [4]. The selected configuration is often significant for performance. Neighboring Xeon Phi cores share an L2 cache and hyperthreading within a Xeon Phi core often *does* improve performance. More applications can benefit from OpenMP parallelism - thanks to NESAP efforts - but Xeon Phi's core count of 68 doesn't conveniently divide over MPI tasks and OpenMP threads, especially when configured for unequally-sized NUMA nodes. Consequently, the distribution and placement of tasks has become an important consideration for performance, and a naive job script is no longer sufficient.

For additional complexity Linux, Slurm and the OpenMP runtimes of each compiler have overlapping

features and limitations around task placement and binding. Feedback from our users was that they would prefer a simple and reliable recipe for correct affinity.

Our strategy for mitigating this complexity is a mixture of recommendations, training and provision of tools to help users prepare an optimal job script.

As discussed in section 5A of choosing default KNL mode, we recommend “quad,cache” configuration (all cores share a single NUMA node with the DDR memory, MCDRAM is used as a transparent cache). For most users, this adds the least complexity at a usually-minimal cost to performance. We are recommending that jobs use "-c" and "--cpu_bind" options for srun (Slurm’s task launcher) and the environment variables OMP_NUM_THREADS, OMP_PROC_BIND and OMP_PLACES to control affinity in a consistent manner across OpenMP runtimes and node types. On our web pages [25][26] we provide recipes for users to calculate suitable values for each of these parameters.

Through training and documentation, we aim to give users the ability to deviate from our recommendations when it would be advantageous to do so. We quickly learned that verbally describing the effect of affinity settings left users in confusion, but diagrams help to clarify. Using consistent nomenclature is also important: what most users think of as a "cpu" corresponds to a "core" in Slurm and OpenMP, while "cpu" in Slurm means "hyperthread" (or “logical cores”). We use diagrams such as Figure 6 [27] to illustrate the nomenclature and relationship between node components, environment variables, and Slurm options.

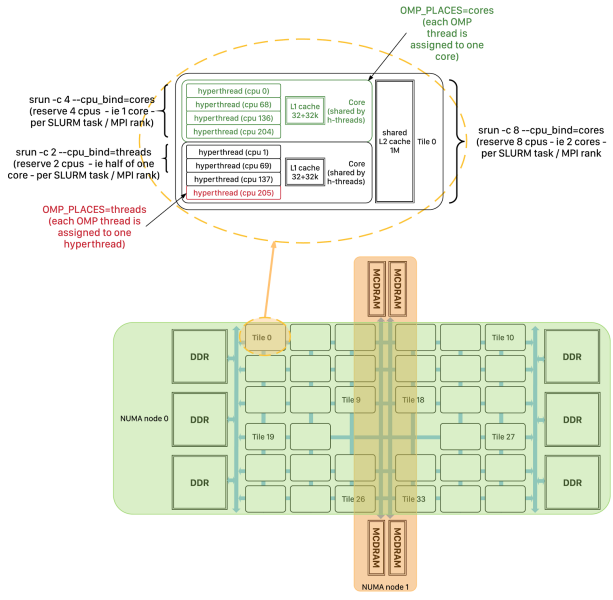

Figure 6. A diagram to illustrate Cori compute node processor components, OpenMP environment variable, and Slurm srun task launching options.

Figures 7 and 8 show slides from our user training, illustrating the effect of key runtime settings for affinity [26].

More advanced users can request nodes in a “flat” MCDRAM configuration, in which the MCDRAM is presented to the OS as one or more secondary NUMA nodes. Linux will allocate memory on the DDR NUMA nodes first, so users must explicitly manage memory placement via directives in source code or, at run time, via “numactl” or the “--mem_bind” option for srun. Our web pages [25] have example batch scripts demonstrating these.

Navigating the options for affinity has proved challenging, so we also provide a tool to generate a job script template [28] and a set of executables (based on Cray’s xthi.c [29]) to report the affinity they experience. The job script generator tool is described in more detail in section 4B, users can use this as a starting point for their script. Users could also replace their application launch with the launch of an xthi executable to verify the task placement that actually occurs [26].

Utilizing the core specialization feature of CLE and Slurm, it is also possible to reserve a specified number of cores for system operations. The application will not use these cores which will reduce any contention for core resources from the operating system. On a system with many smaller cores like Xeon Phi such a setting may be beneficial depending on the performance profile of an application.

B. Job Script Generator

As discussed in the previous section, developing job scripts that achieve a good process and thread binding has become a byzantine task. It becomes even more complicated with the addition of advanced Slurm features such as file broadcasting and core specialization, etc. However, most users are unconcerned about the details of how Slurm performs bindings behind the scenes but rather seek a quick submit script which will work for their case. For that purpose, we provide an online job script generator [28] on our user portal my.nersc.gov. The job script generator is a small javascript application that guides the user to select from typical requirements and produces a job script with the directives and commands that will satisfy those requirements.

The user can select between Edison, Cori-Haswell and Cori-KNL and based on that selection, the maximum number of allocatable nodes, cores per CPU and sockets per node are set internally. Users then walk through the form and can enter details about their application, i.e., assign a job name, an executable string or email address. Most importantly, they can select how many nodes they would like to use and how many MPI processes per node as well as threads per process they want to use. The Cori-KNL tab

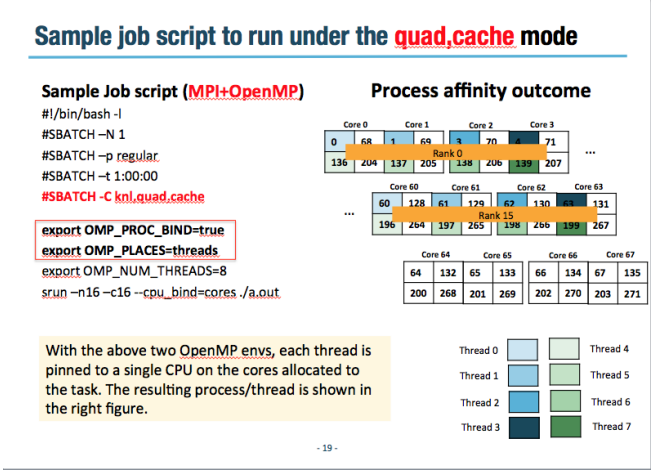

Figure 7. Sample user training slide on running jobs on Cori.

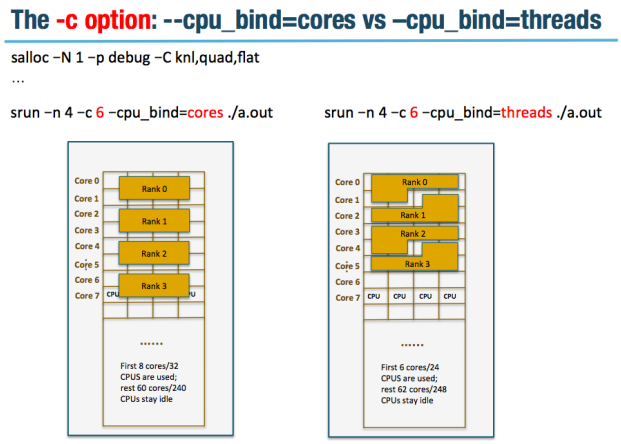

Figure 8. Sample user training slide on running jobs on Cori.

offers additional KNL-specific selections such as the NUMA and memory configuration.

The user can select between Edison, Cori-Haswell and Cori-KNL and based on that selection, the maximum number of allocatable nodes, cores per CPU and sockets per node are set internally. Users then walk through the form and can enter details about their application, i.e., assign a job name, an executable string or email address. Most importantly, they can select how many nodes they would like to use and how many MPI processes per node as well as threads per process they want to use. The Cori-KNL tab offers additional KNL-specific selections such as the NUMA and memory configuration.

When the user is ready and clicks on the button *Generate Script*, an error checking mechanism will highlight problematic sections in yellow (warning) or red (error). Those are issued when allowed but probably sub-optimal or unallowed configurations were selected by the user respectively.

Concerning thread and MPI binding, the generator automatically selects a good binding for the given configuration. The generator also offers an advanced mode in which the user can influence the thread binding to some degree. For jobs which ask for more than a certain number of MPI ranks, the generator automatically adds the Slurm option *sbcast* (copy executables to compute nodes beforehand to reduce large scale job startup time). Figure 9 is a sample screenshot with user choices and generated batch script.

C. *Launching Large Jobs*

The high processor density of KNL and the scale of Cori introduces some challenges for the Lustre file system.

One example is that when launching an executable, each MPI process requests the appropriate pages for the executable from Lustre via a remote procedure call (RPC). The Lustre metadata server (MDS) services these RPCs, but cannot efficiently service a large volume of simultaneous RPCs all requesting the same pages. If a user tries to run an application with a large number of MPI processes per node, this can lead to a long delay (up to several minutes) between the first and last processes on each node starting the job. Users can mitigate this startup delay by using Slurm's "sbcast" feature, which broadcasts the executable from Lustre to each node's RAM disk, thereby funneling each MPI process' RPCs to its own local memory.

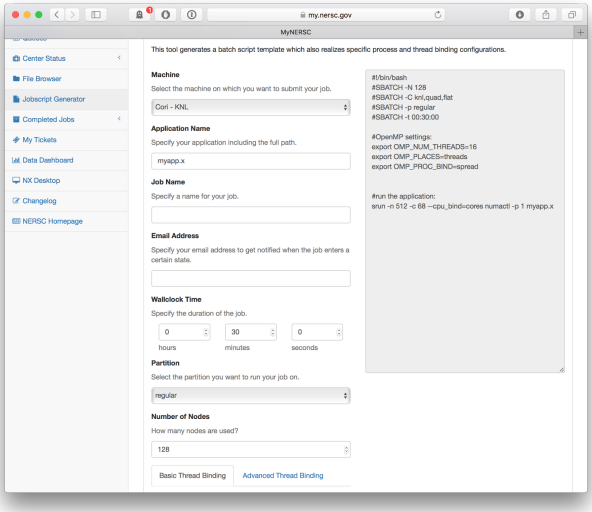

Figure 9. Sample screenshot of the NERSC job script generator on my.nersc.gov with user choices and generated batch script.

D. Launching Large Jobs

The high processor density of KNL and the scale of Cori introduces some challenges for the Lustre file system.

One example is that when launching an executable, each MPI process requests the appropriate pages for the executable from Lustre via a remote procedure call (RPC). The Lustre metadata server (MDS) services these RPCs, but cannot efficiently service a large volume of simultaneous RPCs all requesting the same pages. If a user tries to run an application with a large number of MPI processes per node, this can lead to a long delay (up to several minutes) between the first and last processes on each node starting the job. Users can mitigate this startup delay by using Slurm's "sbcast" feature, which broadcasts the executable from Lustre to each node's RAM disk, thereby funneling each MPI process' RPCs to its own local memory.

E. Performance Tuning

Compared to Haswell or Ivy Bridge, KNL requires more cores to drive the network. We recommend that users with hybrid MPI/OpenMP codes run with at least 8 processes per node to get the best communication performance from the Aries high speed network.

The use of hugepages [30] to get the best communication performance is also recommended because the Aries NIC has resources to track a limited number of pages of memory. With bigger pages more memory addresses can be utilized simultaneously, especially for codes that have many small point-to-point messages or MPI_Alltoall calls with small buffer sizes. We have found that the 2M pages provided by craype-hugepages2M module is sufficient in many cases. Figure 10 shows hugepages effect for the VASP performance running on 1 to 16 nodes [31]. Other NESAP applications which are known to see significant benefit from hugepages are MILC, MFDn and HipMer. Since the hugepages availability may decrease the longer a node is up, it is recommended for system admins to monitor this via Cray Node Health Check (NHC) tool and reboot nodes from time to time.

In addition to uGNI Cray provides an alternative low level communication library DMAPP. For some collective and RDMA operations Cray offers optimizations based on the DMAPP interface. The interface can be enabled by linking libdmapp and setting MPICH_USE_DMAPP_COLL=1. This setting also enables some hardware acceleration for some operations such as barriers and alltoall.

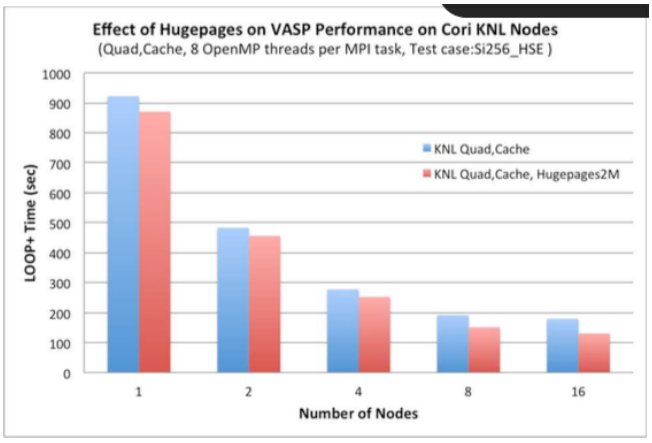

Figure 10. Hugepages effects on VASP performance running on 1 to 16 KNL nodes.

By default, MKL uses multi-threaded libraries in a single threaded region and single threaded libraries within an OpenMP region. We worked with Intel to come up with recommendations to our users to turn this default off, and allow the MKL to use multiple threads, as if it is a nested OpenMP region under the outer layer of OpenMP.

V. Other Considerations

A. KNL Default Mode Selection

The Xeon Phi has the option of being configured at boot-time in a variety of sub-NUMA clustering (SNC) modes which control the degree of memory locality within the on-chip mesh network. Depending on the mode, the chip can appear to the OS as having 1 ("quadrant", or "quad" mode), 2 ("snc2") or 4 ("snc4") NUMA domains for the cores. It is also possible to configure the MCDRAM as a direct-map cache ("cache" mode) or to expose it as a separate NUMA domain from the DDR ("flat" mode).

In our testing it was found that the SNC modes introduce significant complexity into user job scripts, especially due to additional NUMA issues and the uneven number of cores in each quadrant in snc4 mode. Our initial testing of modes was done with a variety of NERSC-8 and APEX [32] benchmark applications including MILC, GTC-P, AMG, MiniDFT, SNAP and PENNANT in additional to other applications. In our testing, when we were able to utilize modes other than quadrant + cache we found at most 5% differences, with not all codes seeing any benefit from specific combinations.

The MCDRAM configured in flat mode outperforms cache mode by a few percent in memory bandwidth bound benchmarks such as stream provided the working set size is less than 16 GB. In real applications the performance difference is often minimal with cache mode sometimes outperforming flat mode. Effectively utilizing a node

configured in flat mode with data sizes larger than 16 GB requires additional programming effort. Utilizing only flat mode also introduces more settings into the standard batch script.

For these reasons we chose a default of quad,cache mode for our KNL nodes. However, for those users who are able to take advantage of other modes we allow approximately 3,000 nodes to be rebooted by the job scheduler. We introduced this limit to avoid overhead related to rebooting nodes, which could last 40 min or longer for each reboot.

B. Using MCDRAM

As discussed in the above section 5A, most of the KNL compute nodes will be used in either quad,cache mode or quad,flat mode. In quad, cache mode there is no need to explicitly use MCDRAM since it is treated as the cache for the DDR memory.

In flat mode Linux sees the MCDRAM as farther from the cores than DDR (*eg* in quad mode DDR is NUMA domain 0 and MCDRAM is domain 1), so memory will be allocated preferentially in DDR. To use the MCDRAM in flat mode users must allocate to it explicitly either at job launch or with NUMA-aware memory allocation in the code.

At job launch a user can specify mandatory or preferred placement, either with srun options or by starting processes via numactl. With mandatory placement ("srun --mem_bind=map_mem:1" or "numactl -m 1", for domain 1), a malloc that cannot be satisfied when the selected NUMA domain has no sufficient free memory will fail, typically resulting in the application terminating. Preferred placement ("srun --mem_bind=preferred,map_mem:1" or "numactl -p 1") allows the memory to be allocated in some other NUMA domain if it cannot be allocated in the preferred one.

Before version 17.02 Slurm did not support preferred placement with the native mem_bind option so we were recommending the numactl method. At NERSC's request to SchedMD this capability was added in 17.02 and installed here in March 2017. Since then we are recommending srun options.

For an application that will not fit entirely within MCDRAM, specifying memory-bandwidth-critical variables to allocate to MCDRAM allows the code to benefit from MCDRAM bandwidth without the constraints on size. The Intel VTune memory access tool [12] can help to identify these memory-bandwidth-critical variables.

To explicitly allocate these variables into MCDRAM source code modifications are needed. Cray and Intel Fortran compilers support directive-based hints such as: “!DIR$ ATTRIBUTES FASTMEM :: a,b,c”. In C, “hbw_malloc(size)” instead of the usual “malloc” function can be used. The Cray compiler wrappers automatically add “*-lmemkind -lnuma*” to the link line and switch to dynamic linking. This may not be a desired behavior for some applications so NERSC installed a custom memkind package with both the static and shared libraries. When loaded, compiler wrappers will add “*-lmemkind -ljemalloc -lnuma*” automatically, and the default build is static.

Another method requiring no source code modification is to use MCDRAM via AutoHBW. At runtime, users specify a range of array sizes in an environment variable. Arrays meeting the size criteria will be allocated to MCDRAM. When the AutoHBW module is loaded, compiler wrappers will add “*-lmemkind -ljemalloc -lnuma*” automatically, and the default build is static.

C. KNLEAP Gating

The NESAP program ensured the bulk of the NERSC workload was ready for KNL, for our remaining users we implemented a gating procedure to gain full KNL access before the system goes into full production mode in July 2017. Using a small number of KNL nodes, the procedure guides users through a list of experiments such as thread-scaling, comparisons between cache and flat-mode performance, vectorization efficiency and strong and/or weak multi-node scaling. This demonstrates readiness for KNL while helping users to get familiar with its distinct features, to think about their application performance and to test different process and thread topologies.

When Cori KNL nodes first became available for benchmarking and production jobs, only NERSC staff and developers of NESAP applications were granted access to the whole machine. After the first few weeks, we opened the “debug” queue access to all users, allowing 30-minute jobs (and later in “regular” partition for up to 2 hours) and on up to 512 nodes on Cori-KNL. The restricted access was because KNL is substantially different from regular multi-core CPUs, a non negligible development effort has to be made to utilize this new architecture efficiently. NESAP teams had access to early hardware revisions and support from Intel and Cray staff and thus could prepare their codes for KNL, while many other NERSC users did not have that opportunity. However, those users were able to use the debug queue to optimize their application and to make sure that their code runs well on KNL before graduating to the full machine.

To facilitate this graduation, we created an application form for the KNL Early Access Program (KNLEAP) [33] on my.nersc.gov web portal. After logging in to the myNERSC service, users can submit an application for full Cori-KNL access on behalf of one of their repos (allocations). The applicant is asked general information about the application, e.g. a short science description, specify the programming languages and kernels used, etc. Subsequently, the applicant has to provide time-to-solution for each of a list of experiments. The results are entered into

the web form which provides examples and live plotting functionality to help the user to achieve the proper formatting. Finally, if the form passes a simple validation test, the user can submit the application. This action will write the application data into a MySQL database backend and automatically generate a ticket in ServiceNow (the NERSC ticketing system) which contains some brief information about the application as well as the database index. This ticket can then be assigned to the consultant on duty who will review the application or forward it to a more suitable NERSC staff member. Once the application is accepted, we notify users via ticket that they have full Cori-KNL access.

Figure 11 shows a sample of live plotting functionality on the KNLEAP application form.

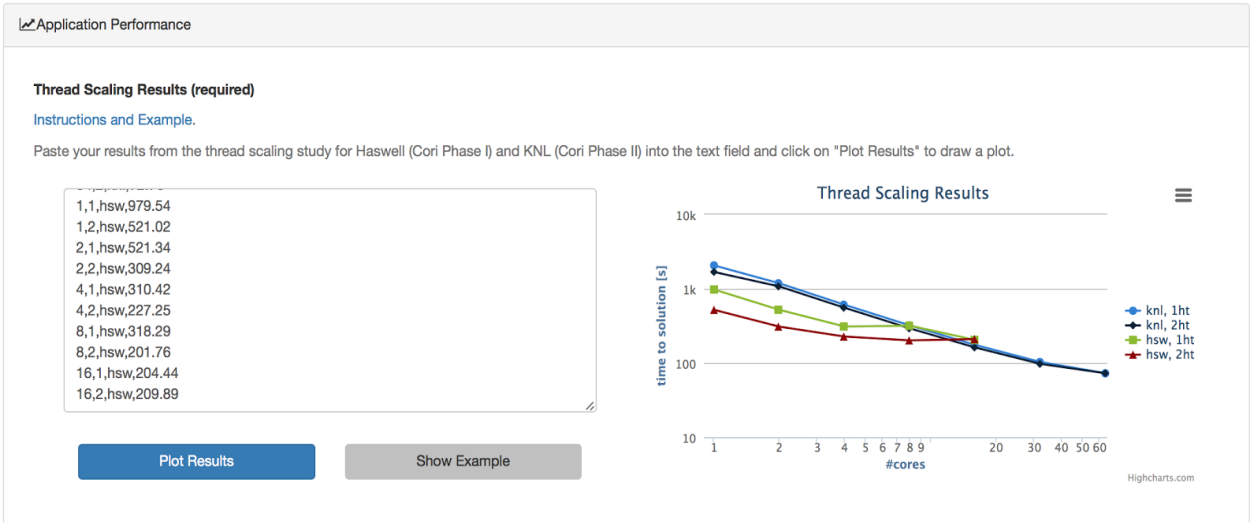

Figure 11. Live plotting functionality on the NERSC KNL Early Access Program (KNLEAP) application form.

D. Job Completion

Job completion statistics are useful indications of how successfully user applications are running on the system.

Figure 12 shows a sample of the percentage of KNL node-hours spent by jobs with each exit state. To provide a quick visual aid regarding how big percentages of jobs failed on a certain day due to system and user issues, we arrange the failures whose causes can be attributed to system problems in the top area of the plot, job terminations due to a user issue or successful runs at the bottom, and job states whose termination causes are difficult to attribute to either in the middle.

Around two-thirds of our KNL workload completes either successfully or with a timeout (many NERSC workflows are based on regular checkpointing, running until the job is killed by timeout and then continuing from the latest checkpoint in a new job). A smaller percentage is cancelled by the user or fails, this likely reflects the debug workload as users prepare their codes for larger-scale KNL use. A small percentage at the top of the chart fail due to system issues.

The Slurm job accounting database is valuable for job completion analysis, and we use this and system usage data to analyze the rate, categories, and causes of job failures.

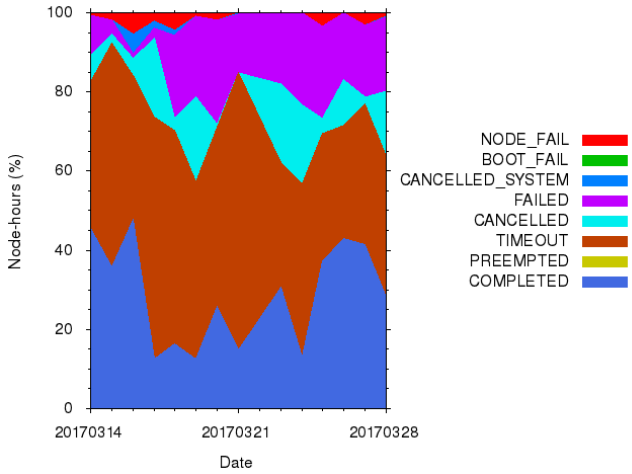

Figure 12. Cori job completion breakdown in node hours.

Slurm records the exit code [34] of an application using an 8-bit integer for each job step, and a signal, if it is responsible for a termination of the job step. Slurm also summarizes the job or step's state with one of the 14 pre-defined state values. Among these, 7 states (BOOT_FAIL, CANCELLED, COMPLETED, FAILED, NODE_FAIL, PREEMPTED and TIMEOUT) are relevant to completed jobs, and they can be used for a job

completion analysis. For our analysis, we add one more category, 'CANCELLED_SYSTEM', for cancelled jobs whose job steps ran well past their batch jobs due to a system issue. The exit code, a signal and the state value for a job can be queried using the sacct command.

The exit code for the entire job is from the exit code of the last process executed in the batch job. Slurm provides another exit code, called the derived exit code, which is the highest exit code value among all the job steps. Considering the exit codes, the derived exit codes and the state values, we have determined with some confidence the true job states, and we are able to see daily job completion statistics.

NERSC has traditionally adopted an approach for a job completion analysis per batch job, instead of per individual application run, mainly because identifying the main cause of a job failure could often be correctly determined after parsing through stdout and stderr messages saved for the batch job. Following this approach, our preliminary job completion analysis uses a batch job based analysis although we have not started capturing job stdout and stderr messages and using them for a more correct analysis, yet.

VI. System Issues Affecting Users

During the KNL early access period, we have seen jobs perform inconsistently or large jobs fail to start; unexplained node failures; timeouts caused by high speed network (HSN) quiesces associated with warm swapping of compute blades; and job or application errors.

A. KNL Node Reboot Time

Reconfiguring the MCDRAM or NUMA mode of a KNL node requires a time-consuming reboot of the node.

Slurm does not consider the current mode of the KNL nodes before it schedules and allocates nodes to next highest priority jobs. KNL node reboot (also called reprovisioning) occurs after the number of requested nodes have been allocated to a user's batch job, but one or more nodes are not in the requested cluster mode. The reboot could take from 40 minutes to a full hour. Slurm shows the job as in the "CF" state, which stands for "CONFIGURING", meaning resources are now allocated to the job, but the job itself could not start until after all allocated resources are ready for use (e.g. reboot for each node has completed).

Slurm recognizes the reboot time and counts the wallclock time used by the job as starting from when the job actually starts to run - after node reboots complete.

The long reboot time confuses users and wastes computing resources. We try to minimize node rebooting as much as possible by keeping the majority of the KNL nodes fixed in quad,cache mode and allowing only the remaining 3,000 nodes to be rebooted by users at job run time. This is a compromise between statically partitioning the cluster by mode (which would disallow full system jobs) and having a significant percentage of the cluster always unavailable due to reboots-in-progress.

Slurm version 17.02 we just installed in March 2017 allows users to specify the maximum additional time (we set the default value of 2 days) they are willing to leave their job queued in order to avoid the node reboot, so that groups of nodes can be allocated to subsequent jobs requesting same cluster modes.

B. High Speed Network (HSN) Quiesces

We have observed Cori Aries HSN quiesces, link failures and throttling during warm swap operations. This impacts user applications performing MPI communication and IO operations enough that user jobs may time out or experience significant run time variation.

Unexpected link failures were seen following warm swapping adjacent compute blades. A patch was provided for NERSC in March 2017, and a Request for Enhancement (RFE) was filed with Cray to allow multiple cables to be swapped.

When a network quiesce was in progress, large-job startup failures were observed. We worked with SchedMD and applied a workaround of increasing a couple of Slurm timeout parameter which stabilized the large job startup in February 2017.

Link recovery following a link failure or network throttle sometimes triggers an NFS failure at the boot node, resulting in failures of service nodes providing DVS, Slurm, DataWarp or Lustre LNET routing.

We are mitigating these issues by performing warm swaps of compute nodes only at prearranged times and with the Slurm batch scheduler paused to prevent jobs from attempting to start during the network interruption.

C. Runtime Variations

Even in applications with perfect load balancing, run time variability of 30% has been observed on both Haswell and KNL. There are several possible sources of variability. We believe the main contributing factors are network interference from other user jobs, the direct map cache of the KNL when MCDRAM is configured in cache mode, turbo mode being enabled on the compute nodes, HSN quiesces, file systems IO load variations, and more synchronization requirements when an equivalent problem is decomposed over many more cores.

We are investigating different packet routing strategies and topology-aware scheduling to mitigate the impact of network performance on user jobs. The routing of packets is controlled by environment variables described in the MPI man pages, however the effects from outside mean that changing this setting for all user jobs is needed for a full study. Slurm has some topology awareness through the --switches=N@HH:MM:SS option where a switch

corresponds to a group in the Aries network and the time value tells Slurm how long to delay a job in order for it to get the requested number of groups. Running on fewer groups may mitigate the influence of other jobs on communication performance.

D. Zone Sort

The MCDRAM in cache-mode is a direct-mapped cache which is physically addressed and tagged. This means that cache lines with the same physical address modulo the size of the cache cannot be in the cache at the same time. In Linux, the list of free memory pages is not kept contiguous in address space, which means that over time it is likely that even when using less than the size of MCDRAM conflicts will exist in the cache and performance will suffer. This is highly dependent on the memory usage of the application and how long a compute node has been booted, in some cases a 30% or higher loss of performance is observed.

To mitigate the impact of the direct map cache on KNL we have enabled a kernel module called "zonesort". This kernel module is made available by Intel and functions by sorting the Linux kernel's list of free pages. We run the zonesort operation in the prolog of each Slurm job step and optionally periodically during a user job. The module is still under development by Intel, but our testing has shown it to be effective at significantly reducing the impact of direct map cache conflicts.

E. Compute Node Failures

We have encountered a higher than expected number of compute node failures from various hardware and software concerns. This especially affects large jobs. A few of the failure modes are as follows:

We have observed that large scale KNL mode changes (reboots) are unreliable. Either the node reboot may fail leaving the node unresponsive, or the boot node managing the reboot can become overloaded and de-stablize the system.

We had a reproducible user-application-triggered node failure "graceful" node power-off during runtime from a Lattice QCD application. This has been traced by Intel to an L2 cache queue control issue, and a combined BIOS change that mitigates this by reducing the XQWB Queue depth slightly (and to fix some other issues related to this problem) was installed on Cori. The patch was confirmed to have fixed this specific node failure mode, but since it had other bad effects causing system unstable, we had to pull out this patch. A standalone patch will be installed when received.

Another reproducible node failure from user application was seen when using Intel VTune stack sampling. The cause was identified as an interaction between the Intel VTSSPP driver and CLE kernel, and has been fixed in VTune version 2017.up2, installed on Cori in March 2017.

VII. Selected User Stories

Here we highlight some of our successful endeavors to get applications and users from different science domains and programming model categories ready for the Cori KNL system.

A. VASP

Vienna Ab Initio Simulation Package (VASP) [35], a computational materials science code, has been the top-ranking code at NERSC for many years. It alone consumes more than 10% of the computing cycles each year and has more than 850 registered VASP users at NERSC. Getting VASP code and users ready for KNL is therefore critical. NERSC has worked with Intel and VASP on this optimization effort and performed extensive performance analysis with a representative VASP workload. NERSC has hosted a beta-testing program for the resulting KNL-optimized hybrid MPI+OpenMP VASP code, provides comprehensive performance guidance for users and with the help of VASP developers held a workshop for NERSC VASP users. These efforts have resulted in three times better performance on KNL in comparison to that on Haswell with representative workloads. Details of this effort are described in Zhao *et. al.* [30].

B. PHOENIX

PHOENIX is a multi-physics simulation code designed to compute synthetic spectra and images for both terrestrial and astronomical objects [36][37][38]. It includes equations of state for atoms, ions, molecules, and solids; modules for various opacity sources; a detailed multi-level treatment of non-equilibrium rate equations for atoms, ions and molecules and a detailed solution of the 3D radiative transfer problem.

In optimizing PHOENIX/3D for KNL, the developers first concentrated on the 3D radiative transfer (3DRT) module, which uses about 80% of the time in the unmodified PHOENIX/3D version. The 3DRT inherently (via the physics of the problem) shows a complex/chaotic memory access pattern as characteristics (rays) are tracked from all solid angles through the voxel grid. The solution of the transfer problem (complex via the presence of scattering) is found with an operator splitting method using a non-local operator, leading to a sparse linear system with a rank equal to the number of voxels.

To improve performance on KNL, a new "tracker" was designed to use a multi-pass method (which needs more RAM) that can more efficiently use OpenMP than the original version. In addition, explicit vectorization with SIMD statements was required as the compiler was not

auto-vectorizing many loops due to (incorrect) cost estimates.

Close inspection of the compiler messages showed an important (time-consuming) loop was not correctly vectorized, and only after including a (vectorized) subroutine internally ("contained") to the tracker did the loop vectorize. In addition, loops outside of the tracker were threaded and vectorized with OpenMP directives, as they started to consume a larger fraction of the overall time after the tracker was optimized. As a result of these optimizations, not only is the KNL version now 2.2 times faster than the original version and slightly faster than a Cori Haswell node (which contains two Haswell CPUs), the Haswell version is now 1.4 times faster than the original code as well. A summary of the solver performance gain is shown in Figure 13.

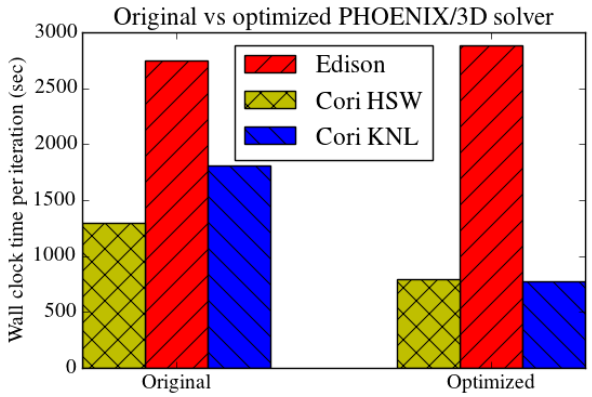

Figure 13. Phoenix/3D solver performance before and after optimization

C. HPX

We are looking into new programming models for the KNL architecture. One of these is HPX (High Performance ParalleX) [39], a general purpose C++ runtime system for parallel and distributed applications. Rather than combining MPI with OpenMP on a node, the intent of HPX is to provide a unified programming model which transparently utilizes the available resources to achieve unprecedented levels of scalability. The HPX library conforms to the C++11 Standard and leverages the Boost C++ Libraries. We have installed HPX on Cori and tested it with the HPX STREAM benchmark and several other applications. Improvements to the runtime are ongoing, so the data here shows just a snapshot in time. Ongoing also are first tests on a full astrophysics application with HPX. These will be presented in an upcoming publication. Currently the full application is running well on 2,048 KNL nodes with the roughly same code that has been optimized for Edison, demonstrating the portability of the HPX model.

A full discussion of the STREAM benchmark in the HPX version and the C++ standard is given in Heller, et. al. [40] and a STREAM itself is introduced in [41]. In [40], the portability of HPX is stressed as the same benchmark gives comparable performance to native CUDA on GPU architectures and native OpenMP on a two socket, 12 cores per socket, Intel NUMA system with Xeon CPU E5-2650v2 processors. STREAM times the performance of copy, scale, add, and a triad calculation measuring sustainable memory bandwidth (in MB/s) and the corresponding computation rate for these simple vector kernels[41]. The take-away from the HPX STREAM results is in being able to provide a single source, generic, and extensible abstraction for expressing parallelism, with no loss in performance compared to optimized CUDA and OpenMP implementations, while at the same time adhering to the C++ standard.

Figures 14 and 15 show the performance of the currently achievable bandwidth with the HPX version of the STREAM benchmark. When running in quad mode, the performance is as expected when placing the memory in DRAM, that is, we achieve close to the nominal peak. When the memory is placed in the High Bandwidth Memory, we see a significant increase in performance for using up to 100 or so worker threads. We see a significant drop with more threads which is mostly due to the fact that the grain size of the different chunks to perform the respective benchmark is too little compared to the costs of synchronizing once the parallel region has been executed. That is, in order to exploit the maximal performance of the KNL, one has to consider considerable larger grain sizes than on regular platforms like Haswell. In its current implementation, the benchmark can not fully utilize the sub NUMA clustering modes, however, we expect it to result in much better performance compared to the "quad" mode, since the scheduling (that is task stealing) is confined to one sub NUMA domain, as well as the synchronization. Further investigation is needed on that front, however, since we believe that a runtime system like HPX should be able to utilize the SNC modes in a similar fashion as we have shown when using GPUs in [40].

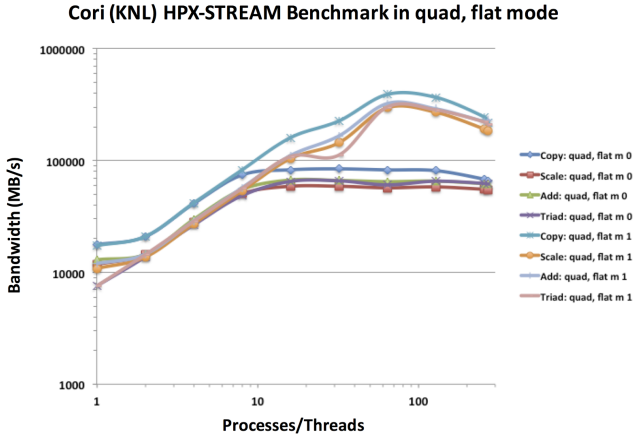

Figure 14. First HPX results of STREAM on KNL for quad, flat mode

using only DRAM (m=0) and using High Bandwidth Memory (m=1).

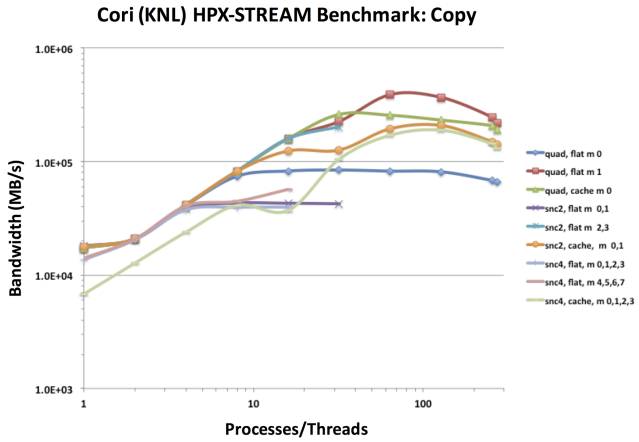

Figure 15. Individual components of the HPX STREAM benchmark (Copy is shown) show some variations in the different available modes.

VIII. Summary

Although most applications can run unmodified on the newest Cori system with the Intel Xeon-Phi KNL nodes, users must explore fine-grained parallelization, vectorization and use of high bandwidth memory to achieve optimized performance on KNL. The KNL architecture is complex but affords a great deal of flexibility for experienced users. There are multiple memory modes, and changing from one to another requires a time-consuming reboot of the node.

There are many aspects an HPC center should consider to achieve best system utilization and efficiency for a KNL system. This paper summarizes NERSC experiences in preparing and transitioning users from the traditional Xeon architecture to the Xeon Phi KNL architecture. Practical aspects of system configuration, programming, and user-facing runtime issues are presented in detail. We hope that the information shared in this paper is useful to other centers when deploying a large KNL system on their computer floors.

Acknowledgments

We would like to thank Cray teams (onsite and remote staff), Intel teams (training and dungeon session staff), and also our NERSC colleagues (especially CSG, ATG, DAS staff and NESAP postdocs) for working with us on preparing users for Cori KNL. We would also like to thank NERSC users for valuable feedback and for helping us investigating system issues.

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] NERSC: https://www.nersc.gov

[2] Cori: https://www.nersc.gov/users/computational-systems/cori/

[3] Intel Xeon Phi products: http://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html

[4] Cori KNL Processor Modes: https://www.nersc.gov/users/computational-systems/cori/configuration/knl-processor-modes/

[5] NESAP: http://www.nersc.gov/users/computational-systems/cori/nesap/

[6] Edison: https://www.nersc.gov/users/computational-systems/edison/

[7] NESAP Projects: http://www.nersc.gov/users/computational-systems/cori/nesap/nesap-projects/

[8] Douglas Doerfler, et. al. Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor. IXPUG 2016.

[9] Samuel Williams. Auto-tuning Performance on Multicore Computers. Ph.D. thesis, EECS Department, University of California, Berkeley (December 2008)

[10] Williams, S., Watterman, A., Patterson, D.: Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Communications of the ACM (April 2009)

[11] Williams, S. Roofline performance model, http://crd.lbl.gov/departments/computerscience/PAR/research/roofline

[12] Intel VTune Amplifier: https://software.intel.com/en-us/intel-vtune-amplifier-xe

[13] Intel Software Development Emulator (SDE): https://software.intel.com/en-us/articles/intel-software-development-emulator:

[14] Intel Advisor: https://software.intel.com/en-us/intel-advisor-xe

[15] Jack Deslippe, Georgy Samsonidze, David A. Strubbe, Manish Jain, Marvin L. Cohen, and Steven G. Louie. "BerkeleyGW: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures." Computer Physics Communications 183, no. 6 (2012): 1269-1289.

[16] T. Barnes, B. Cook, J. Deslippe, D. Doerfler, B. Friesen, Y. He, T. Kurth, T. Koskela, M. Lobet, T. Malas, A. Ovsyannikov, C. Kerr, J. Dennis. Evaluating and Optimizing the NERSC Workload on Knights Landing. 2016 7th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computer Systems (PMBS). DOI 10.1109/PMBS.2016.010.

[17] Thorsten Kurth, William Arndt, Taylor Barnes, Brandon Cook, Jack Deslippe, Doug Doerfler, Brian Friesen, Yun (Helen) He, Tuomas Koskela, Mathieu Lobet, Tareq Malas, Leonid Oliker, Andrey Ovsyannikov, Samuel Williams, Woo-Sun Yang, and Zhengji Zhao. Analyzing Performance of Selected Applications on the Cori HPC System. IXPUG Workshop "Experiences on Intel Knights Landing at the One Year Mark"" at ISC 2017. Accepted.

[18] Mario Melara *et al.* Using Spack to Manage Software on Cray Supercomputers. Cray User Group 2017.

[19] Tips for Using CMake and GNU Autotools on Cray Heterogeneous Systems. http://docs.cray.com/books/S-2801-1608//S-2801-1608.pdf

[20] Slurm: https://Slurm.schedmd.com

[21] Michael Karo1, Richard Lagerstrom1, Marlys Kohnke1, Carl Albing. The Application Level Placement Scheduler. Cray User Group 2006.

[22] Using aprun to launch applications: http://docs.cray.com/books/S-2496-4101/html-S-2496-4101/cnl_apps.

html

[23] Austin, B., Bhimji, W., Butler, T., Deslippe, J., French, S., Gerber, R., Jacobsen, D., & Wright, N. (2015). 2014 NERSC Workload Analysis - Slides of 2015 presentation by Brian Austin.

[24] Austin, B., Butler, T., Gerber, R., Whitney, C., Wright, N., Yang, W. -S., & Zhao, Z. (2014). Hopper Workload Analysis. - Report Number: LBNL-6804E

[25] Cori Running Jobs on KNL page: https://www.nersc.gov/users/computational-systems/cori/running-jobs/example-batch-scripts-for-knl/

[26] Cori Running Jobs General Recommendations: http://www.nersc.gov/users/computational-systems/cori/running-jobs/general-running-jobs-recommendations

[27] Steve Leak and Zhengji Zhao, Using Cori. Presented at the 2016 NESAP Workshop and Hackathon: https://www.nersc.gov/assets/Uploads/Using-Cori-20161129-NESAP-HACKATHON.pdf

[28] NERSC batch script generator: https://my.nersc.gov/script_generator.php

[29] Cray code xthi.c: http://docs.cray.com/books/S-2496-4101/html-S-2496-4101/cnlexamples.html

[30] Cray Programming Environment Users Guide. http://docs.cray.com/books/S-2529-116/S-2529-116.pdf

[31] Zhengji Zhao *et al.* Performance of MPI/OpenMP Hybrid VASP on Cray XC40 Based on Intel Knights Landing Many Integrated Core Architecture. Cray User Group 2017.

[32] APEX: http://www.nersc.gov/research-and-development/apex/

[33] NERSC KNL Early Access Program (KNLEAP): https://my.nersc.gov/knleap.php

[34] Slurm exit code: https://Slurm.schedmd.com/job_exit_code.html

[35] G. Kresse and J. Furthm_ller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. Comput. Mat. Sci., 6:15, 1996; http://www.vasp.at/

[36] Baron, E., Chen, B., & Hauschildt, P. H. 2010, Astrophysics Source Code Library, ascl:1010.056

[37] Hauschildt, P. H., & Baron, E. 2010, *Astronomy & Astrophysics*, 509, A36

[38] De Gennaro Aquino, I., Hauschildt, P. H., & Wedemeyer, S. 2016, 19th Cambridge Workshop on Cool Stars, Stellar Systems, and the Sun (CS19), 149

[39] http://stellar-group.org/libraries/hpx/

[40] Thomas Heller, Hartmut Kaiser, Patrick Diehl, Dietmar Fey, and Marc Alexander Schweitzer, Closing the Performance Gap with Modern C++, n International Conference on High Performance Computing, pp. 18-31. Springer International Publishing, 2016.

[41] McCalpin, J.D.: Stream: sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia (1991–2007), a continually updated Technical report. http://www.cs.virginia.edu/ stream/