Transitioning Users from the Franklin XT4 System to the Hopper XE6 System

Katie Antypas *and* **Yun (Helen) He**, *National Energy Research Scientific Computing Center*

ABSTRACT: *The Hopper XE6 system, NERSC's first peta-flop system with over 153,000 cores has increased the computing hours available to the Department of Energy's Office of Science users by more than a factor of 4. As NERSC users transition from the Franklin XT4 system with 4 cores per node to the Hopper XE6 system with 24 cores per node, they have had to adapt to a lower amount of memory per core and on-node I/O performance which does not scale up linearly with the number of cores per node. This paper will discuss Hopper's usage during the "early user period" and examine the practical implications of running on a system with 24 cores per node, exploring advanced aprun and memory affinity options for typical NERSC applications as well as strategies to improve I/O performance.*

KEYWORDS: XE6, I/O performance, hybrid programming models

1. Introduction

The Hopper system [1], a Cray XE6 with over 153,000 cores installed at the National Energy Research Scientific Computing (NERSC) Center is the first peta-flop system available to the general Department of Energy (DOE) Office of Science research community. NERSC is the primary high performance computing facility for the Department of Energy's Office of Science and serves the broad DOE community providing computational and storage resources and services to a range of researchers in disciplines spanning astrophysics, climate, combustion, biology, material science, chemistry, fusion and physics. NERSC supports over 4,000 users representing roughly 400 different research projects.

The addition of the Hopper system with a peak performance of 1.28 peta-flops to the NERSC facility increases the computing time available to NERSC users by a factor of 4 over NERSC's previous generation large scale system, Franklin, a Cray XT4 system with over 38,000 quad-core processors and a peak performance of 352 tera-flops.

Hopper

The Hopper system was deployed in two phases. The first phase was an XT5 system that was delivered to NERSC in the fall of 2009. This paper focuses on the second, larger, and final configuration of the Hopper system, a Cray XE6 named after Computer Scientist and Rear Admiral Grace Murray Hopper, consisting of 3,284 compute nodes, each containing two 12 core AMD 2.1 GHz MagnyCours processors. Each compute core has a peak performance of 8.4 Gflops/sec resulting in a system with a peak performance of 1.28 PFlops. All but 384 compute nodes have 32 GB of DDR3 1333 MHz memory while the remaining larger nodes have 64 GB of DDR3 1333 MHz memory creating a system with over 217 TB of memory. The nodes are interconnected in a 3D torus network with the Gemini interconnect which has significantly improved reliability features compared to the earlier generation Seastar2 network used on the Franklin system. Whereas a failed or down link in the Seastar2 interconnect will cause an outage on the entire machine, the Gemini interconnect has the capability of routing around down links. The Gemini interconnect also has greatly reduced latency and higher bandwidth compared to the Seastar2 network. Table 1 shows the MPI latency and MPI bandwidth of the Gemini and Seastar2 networks.

	Franklin (Seastar2)	Hopper (Gemini)
MPI Latency	6.5 microseconds	1.6 microseconds
MPI Bandwidth	1.6 GB/sec/node	6.0 GB/sec/node

Table 1: Comparison of MPI latency and bandwidth on the Seastar2 and Gemini networks.

NERSC has configured the Hopper system with five different parallel file systems: a GPFS based global home file system mounted across all NERSC systems, two identical local Lustre scratch file systems which have a peak performance of 35 GB/sec and a capacity of 1 PB each, a global scratch GPFS file system, which is mounted on most NERSC systems as well as a GPFS based project file system intended for collaborative data sharing.

As with Franklin, the previous generation machine, Hopper runs two different operating systems, a full featured SuSE Linux is run on the login nodes and service nodes while the compute nodes run a light-weight operating system called the Cray Linux Environment (CLE). A new feature of the Hopper system is the external login nodes that exist outside of the main torus network. The 8 external login nodes have 16 cores each with 128 GB of memory and increase the usability of the system by allowing users to do more computationally and memory intensive processing than an internal login node would allow. The external login nodes also allow users to access the five user file systems installed on Hopper when the compute portion of the machine is unavailable. For a more through discussion of the external services on Cray systems please refer to last year's CUG paper [2].

Hopper Timeline and Usage

The second phase of the Hopper system was delivered in four shipments at the end of the 2010 summer. By September 17th, all 68 cabinets had arrived at NERSC's Oakland Scientific Facility. During the fall, the system was installed and integrated into the NERSC environment and the earliest NERSC users accounts were enabled November 15, 2010. These users were targeted because their applications would stress various parts of the systems. For example, users who ran large concurrency jobs were invited on the Hopper system early to test the network and scalability of the system. Users whose codes contained both MPI and OpenMP were invited to test the hybrid programming abilities of the system, and heavy I/O users were invited to stress the I/O sub-systems. In exchange for providing feedback to NERSC staff about the performance of their codes and usability of the system, all early users ran their applications on the system free of charge. By the winter holiday, all user accounts had been enabled on Hopper, though the time on the system was still being shared with Cray engineers who continued to test the system. On February 4th, the official acceptance test and then, 30 day availability test began, giving users uninterrupted system access. After successful testing, the Hopper system was accepted by NERSC on April 19, 2011 and the system went into production shortly afterward on May 1st. The time from November 2010 to May 1st 2011 is referred to as the "Early User Period" during which time, user accounts are not charged. It is a time when scientists and researchers can experiment with larger concurrency jobs or try new experiments they might not be able to do with their normal production allocation. During the early user period, over 350 million hours were delivered to the Department of Energy's, Office of Science researchers. Two hundred eighty of NERSC's over 400 projects used time on the Hopper system and over 1000 users accessed the system. Figure 1 shows the breakdown of the early science hours by science category.

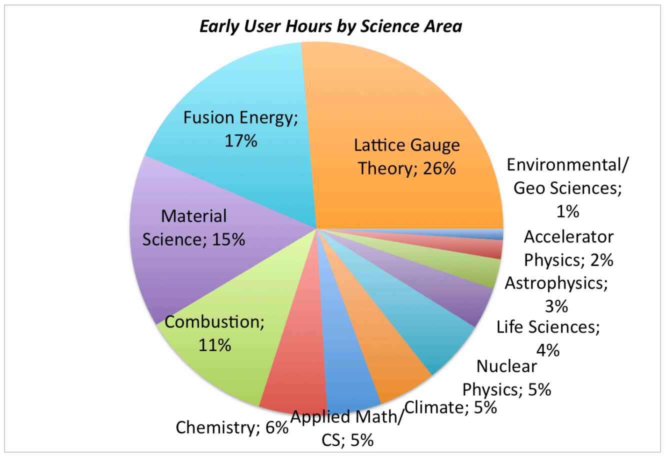

Figure 1: Breakdown of CPU core hours on the Hopper system from November 15, 2010 to April 30, 2011.

The largest science area represented during the early user period was lattice gauge theory followed by fusion energy, material science and combustion.

From the start of the availability test on February 5th until May 1st, the system ran at a utilization of 81.7%. The calculation for utilization is simply hours delivered to science projects divided by 24 hours per day. All benchmarking by NERSC or Cray staff is not included in the hours to science projects. Since it also includes downtimes and system maintenances, we are pleased with a utilization of close to 82% for the first few months of NERSC's first peta-flop XE6 system; however, we will continue to work to improve utilization on the system. During the early user period, we noticed that the Hopper system was able to recover and ride through problems better than the previous generation XT4 Franklin system. To illustrate, Lustre fail-over has been implemented and tested on the Hopper system and thus when an OSS failed in February, the system was able to recover on its own. Some jobs were lost, but system utilization stayed above 60%, which for a system as large as Hopper, means a significant number of cores were still running user applications. A major source of lost utilization is system maintenances and upgrades. Maintenances are necessary to keep a machine healthy and patched; however, with Hopper delivering 3.6 million CPU core hours per day, the opportunity cost of taking a system down is high. NERSC always tries to do as many tasks during a single maintenance in order to avoid frequently taking the system up and down. Finally, scheduling problems have been a source of lost utilization on the system. NERSC has a number of different queues configured for users to submit jobs. These queues have different priorities, and it was found that some of the jobs submitted to the high priority debug queue were inhibiting the largest jobs on the system from starting appropriately. NERSC has since adjusted the queue structures so this type of interference is less likely to happen. Figure 2 shows the utilization of the system from February 5th, through April 30th.

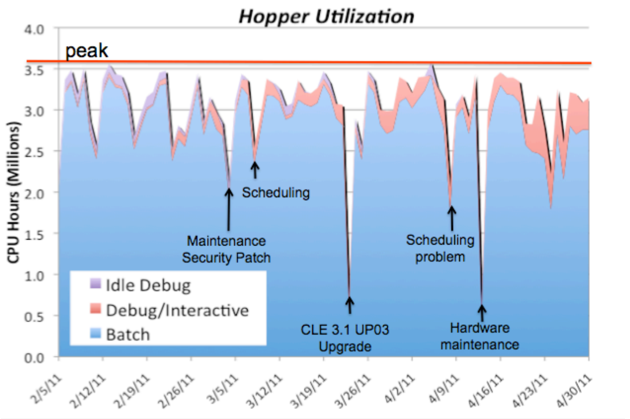

Figure 2: Utilization of Hopper system from the start of acceptance testing February 5, 2011 to April 30, 2011.

The NERSC workload is not only diverse in terms of the science areas and algorithms, but also in terms of the size of jobs run on the system. Some applications can utilize the entire machine, while other applications run at more modest sizes of a few hundred or few thousand cores. Figure 3 shows the breakdown of job sizes on the system during the early user period. When the system started acceptance testing February 4th, 2011 the job sizes became more stable with roughly 50% of raw hours going for jobs run at 16,384 cores or higher and over 20% of jobs run at 65,536 cores or higher.

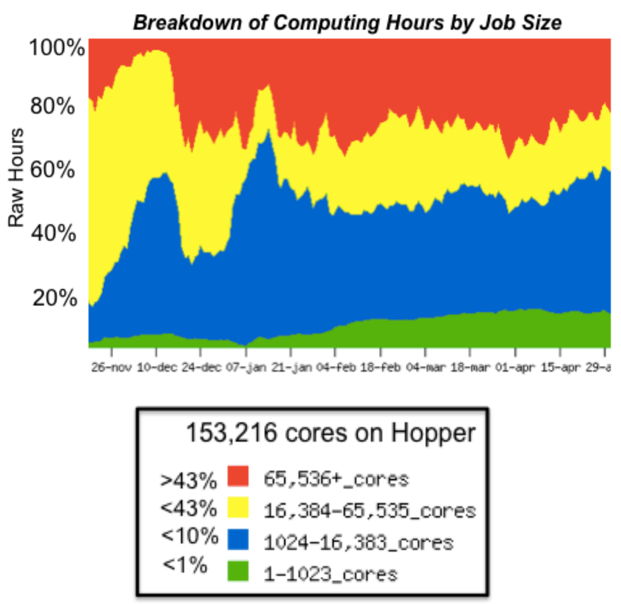

Figure 3. Breakdown of CPU core hours by job size on the Hopper system

User Feedback

User feedback from the early user period on Hopper was generally positive. Because of the similarity of the programming environments between Franklin and Hopper, most applications were fairly easy to port to the new system. At least at the beginning of the early user period, many users were able to take advantage of the relatively fast turn-around time for large number of cores. One user states, “The best part of Hopper is the ability to put previously unavailable computing resources toward investigations that would otherwise be unapproachable.” Another states, “During the "free" period Hopper provided very good turnaround for my jobs, which were in the 5,000 - 10,000 processor range. This was very important for finding errors, scaling up my code and generating new results”

2. Transitioning Users from Franklin to Hopper

User Environment

While the software environment between Franklin and Hopper has changed little, one of the largest challenges transitioning users from the Franklin system to the Hopper system has been teaching scientists to effectively use all 24 cores of the Hopper system. Figure 4 illustrates a Hopper compute node.

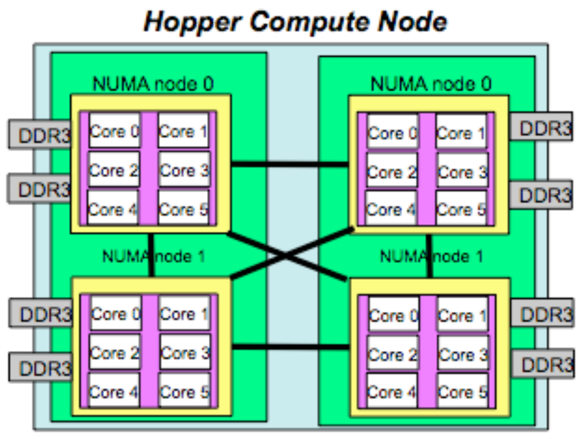

Figure 4. Schematic of Hopper compute node showing non-uniform memory access (NUMA) nodes.

A major difference between Franklin and Hopper is the available memory per core. On Franklin, each quad-core compute node has 8 GB of physical memory, while most Hopper compute nodes have 32 GB of physical memory. However, not all that memory is available to user programs. Compute Node Linux (the kernel), the Lustre file system software, and message passing library buffers all consume memory, as does loading the executable into memory. Thus the precise memory available to an application varies.

Approximately 31 GB of memory can be allocated from within an MPI program using all 24 cores per node, i.e., 1.29 GB per MPI task on average. If an application uses 12 MPI tasks per node on Hopper (half packed), then each MPI task can use about 2.58 GB of memory. In comparison, the average memory available for applications on Franklin is about 1.88 GB using all 4 cores per node and 3.77 GB half packed, using only 2 cores per node.

Using 24 cores per node has been a challenge for some users. As shown in Figure 4, each Hopper compute node consists of 24-cores grouped into 4 NUMA nodes, each with 6 cores. Any of the 24 cores can access the memory of any NUMA node, however, memory access time from a core to a local NUMA node’s memory will be faster than to memory on a remote NUMA node. Put a different way, compared to the Franklin compute node that had uniform memory access for all cores, the Hopper compute node is not flat, task placement and locality matter. Users need to be particularly careful if they are trying to run their application on fewer than 24 cores per node. For example, for pure MPI problems, some users may only want to run on 8 or 16 cores on the nodes in order to increase the amount of memory available per MPI task. Because of the non-uniform memory access on a Hopper node, how those MPI tasks are placed on a node can have a large impact on performance. The simplest default options to launch a job on the compute nodes with the ALPS aprun command, will assign tasks to the first numa node before moving on to the next NUMA node. This can result in sub-optimal performance, as ideally, a user would want to spread MPI tasks evenly across the node. Figure 5A below shows the naive way to launch an application requesting only 4 cores per node compared to a more optimal way shown in Figure 5B.

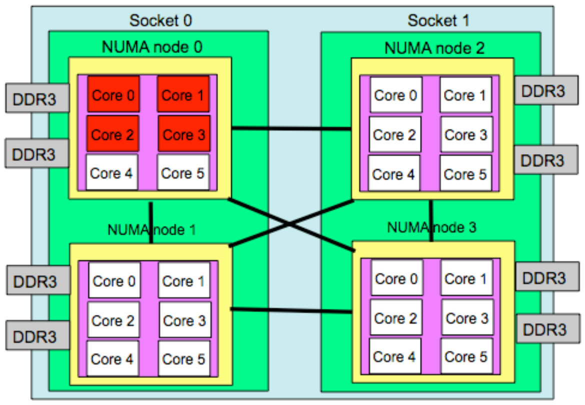

Figure 5A: Un-optimal placement of MPI tasks

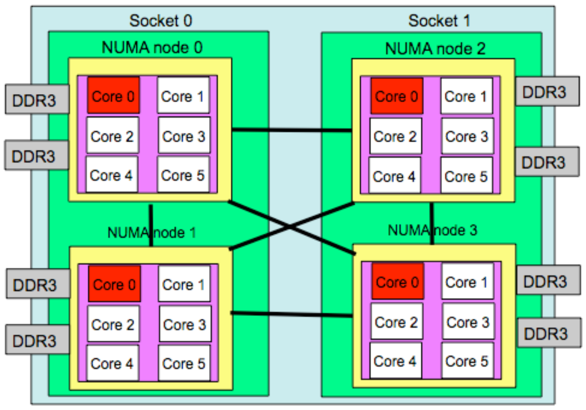

Figure 5B: More optimal placement of MPI tasks

One way to spread MPI tasks evenly across a node is to add the "-S" option to the aprun command line that is used to specify how many MPI tasks to assign per NUMA node. The example below shows the performance of a real user application running the VASP code [3] in a 660 atom system with 384 MPI tasks on 32 codes, but one simulation runs with the cores optimally placed across NUMA nodes while the other does not.

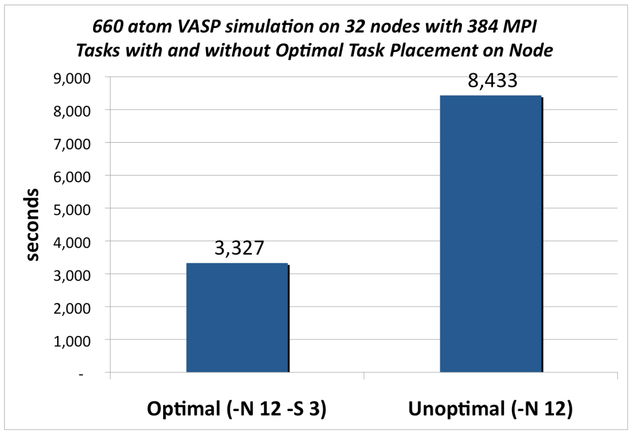

Figure 6: Runtimes for VASP simulation run with and without optimal MPI task placement.

Figure 6 shows the dramatic effect task placement can have on performance. For the VASP simulation the aprun line in the optimal case is: "aprun –n 384 –N 12 –S 3 vasp" where "-N 12" means to use 12 tasks per node, "-S 3" means to assign 3 MPI tasks per numa node. Without "-S 3", all 12 cores will be allocated on the first 2 NUMA nodes, creating contention for memory bandwidth on the first two NUMA nodes. Additionally, there will be sub-optimal memory access to the final two NUMA nodes. More advanced aprun options on Hopper such as "-sn" (number of NUMA nodes to use per compute node), "-ss" (specify strict memory containment per NUMA node) and "-cc" (controls how tasks are bound to cores and NUMA node) are also available for users to tune application performance.

Because of the larger number of cores available per node, 24, and the larger amount of shared memory available per node, 32 GB, NERSC encourages users to try a new programming paradigm, hybrid MPI/OpenMP, instead of pure MPI. While most pure MPI applications that run successfully on Franklin will also run successfully on Hopper, some applications may fail due to out of memory conditions. One strategy, as discussed above, is for users to run on fewer than 24 cores per node, leaving some cores idle. This is an acceptable strategy, though, the user will be charged by the node, and so leaving cores idle will result in a more costly run. An alternative is for users to try a hybrid MPI/OpenMP model. Although this paradigm may not necessarily outperform pure MPI performance, it can lower memory usage by requiring fewer instances of the same program, fewer MPI buffers and the storage of fewer ghost cells. It takes advantage of the natural two layers of parallelism of a Hopper compute node: using MPI across nodes or NUMA nodes, and using OpenMP within nodes or NUMA nodes. Fewer MPI tasks per node avoid extra communication costs within a node and allow for larger messages sizes to be sent off node.

Memory allocation also uses a "first-touch" policy, which means memory is mapped on the NUMA node of the compute core by which it is first written. With hybrid MPI/OpenMP, it is recommended that each thread initialize its memory at the "first touch", so that it will only access memory in its local NUMA node. Since this is usually difficult to do, NERSC recommends using no more than 6 threads per node, even though the maximum number allowed is 24 per node [4]. We have seen that typical Hybrid MPI/OpenMP applications show performance improvements from 1 thread to 6 threads per node, but drop when 12 or 24 threads are used unless perfect "first touch" memory allocation can be accomplished. Our strong suggestions to use no more than 6 threads per node with MPI/OpenMP programming on Hopper is clearly delivered to NERSC users in web documentation, NERSC training sessions, and workshops.

Dynamic and Shared Libraries

The Hopper system supports the capability to run dynamic and shared object codes on the compute nodes. This functionality is not supported on the Franklin system that runs an earlier version of the CLE operating system, 2.2. On Hopper with OS level CLE3.1, static binaries are still the default and suggested way to launch jobs on the system; however, dynamically linked applications with shared libraries can be run if a user compiles with the -dynamic flag and sets an environment variable at runtime. Dynamic and shared library applications are run on compute nodes via a software layer called DVS (Data Virtualization Service) [5] which forwards the service node environment and system libraries to the compute nodes. Many Cray supported libraries such as SciLab, MPI, pthreads, are available as dynamic shared objects (DSOs), and users can define and use their own DSOs. This feature also enables other executables (non-ELF binaries) such as shell scripts, perl and python scripts, that need shared libraries at run time, to run on the compute nodes. Most system calls are also now supported. The ability to run shared and dynamic object codes on the Hopper system gives users an environment more similar to the full Linux environment of traditional clusters and users appreciate the convenience of this support.

We have discovered however, that loading shared libraries on compute nodes can be slow, especially for large core count jobs. It appears that the DVS software layer is not currently configured to efficiently read shared objects in parallel, which results in serialization and a slowdown for user applications. We are actively working with Cray for better tuning of DVS parameters. Meanwhile, we advised NERSC users to try to combine multiple shared libraries into one or for python applications, reduce the number of python import calls if possible. Some users had to build static executables to avoid the performance hit caused by the shared libraries slowness in the DVS layer [6].

I/O Performance

An area users continue to find challenging is scaling application I/O and getting optimal I/O performance out of the system. Writing and reading large amounts of data from an HPC system is often a challenge for large applications because of the great difference between the compute node's ability to calculate data compared to the speed and performance of the magnetic spinning disks used to store data. This forces scientists needing to read or write large amounts of data to come up with a host of techniques for increasing bandwidth, including striping data across disks, aggregating data to a few nodes and using various buffering techniques. One challenge for users has been that although the number of cores on a node has increased from 4 cores per node on the Franklin system to 24 cores per node on the Hopper system, I/O performance has not scaled with the number of cores. IOR [7, 8] is a benchmark developed at Lawrence Livermore National Laboratory and has a flexible interface and can be run with various user I/O APIs such as POSIX, MPIIO or HDF5. It can also be set to output different block sizes. Figure 7 shows the difference in I/O performance on the Franklin and Hopper systems using the IOR benchmark on up to 24 MPI tasks where each MPI task writes out 2GB of data. On Hopper this represents one node while on Franklin this represents 6 nodes. The numbers are closely aligned for the first 8 MPI tasks and then start to diverge thereafter. Franklin performance continues to increase, while Hopper performance levels out at ~1700 MB/second. A question that needs to be resolved is why Hopper write performance peaks at 1700 GB/sec. Franklin's peak node performance achieves two-thirds the performance of the Seastar2 network's injection bandwidth of 1.6 GB/sec whereas Hopper only achieves 28% of the bandwidth of Gemini's 6 GB/sec injection bandwidth rate. It is interesting to note that using direct I/O that disables caching and buffering at the operating system does not improve performance. This leads us to believe that throttling is occurring at the Lustre client level.

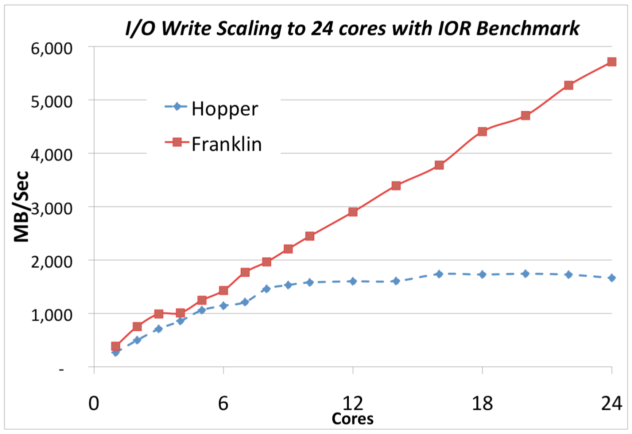

Figure 7: I/O write scalability comparison between Franklin and Hopper to 24 cores.

To try to improve I/O performance for user applications, Cray resurrected a library called iobuf that was used in the earlier Catamount systems. The purpose of the iobuf library is to buffer I/O requests on a compute node by intercepting read and write calls and thus allowing fewer, larger blocks of I/O to be read and written at a time, improving I/O performance. Cray provides the iobuf library in a module and the user needs only to load the module and re-link the application. Additionally, the environment variable IOBUF_PARAMS needs to be set to indicate which user files should be buffered. The user also has the option to set other parameters like the size of the buffers and the number of buffers to use per file. The defaults are 4 buffers per file each with a size of 1MB. There are other more advanced settings for shared files and shared buffers, prefetch and flush settings. Additionally, the iobuf module can be used to gather I/O statistics that are appended to the user's standard out file.

The iobuf library's capabilities were tested on 1560 cores with the IOR benchmark. This particular IOR test at NERSC uses the IOR POSIX file-per-processor interface to measure the aggregate I/O performance on the system. Each Lustre scratch file system has 156 Object Storage Targets (OSTs), and this test is intended to write 10 files on each OST. Each processor core writes and subsequently reads 2GB for a total output of over 3TB. Figure 8 shows the difference in performance between runs with and without the iobuf library for three different transfer sizes, one with a transfer size of 10,000 bytes, a second with 1 million bytes and a third with 1,024,576 bytes. These three transfer sizes were chosen because performance differences have been seen between small and large transfer sizes as well as transfer sizes which were not a power of two. It is important that NERSC understand the performance implications of different transfer sizes since real users applications do not always output data in large power of two sized chunks. In all cases, the IOR runs with I/O buffering enabled outperformed the runs that did not and again in all three cases, reads show significantly more benefit from I/O buffering than writes. The read performance is between 30-40% better when I/O buffering is enabled. For writes, performance only increases between 1 and 11%. I/O performance increases the most for the 10k write cases and the least for the power of two MiB case. This is an expected result as smaller writes would benefit from aggregation the most. For these runs, the IO buffering parameters have been set to two buffers each of size 2 MB. It is clear that the parameter space could be explored further with experiments to test larger and more buffers.

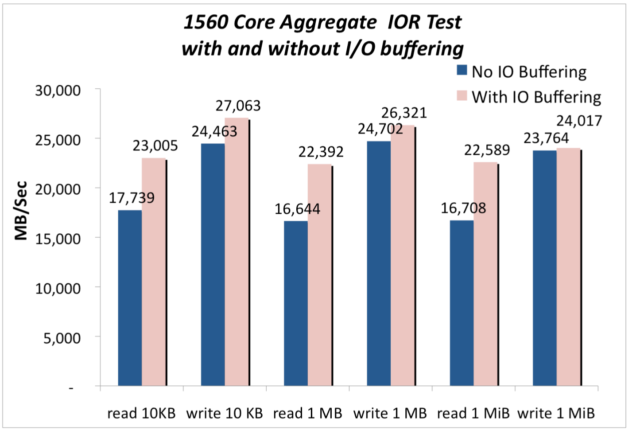

Figure 8: Comparison of I/O performance for different sets of transfer sizes using the IOR benchmark on 1560 cores.

In addition to testing the iobuf module on a synthetic benchmark like IOR, we linked a full scientific application, MAESTRO [9] against the iobuf library. MAESTRO is a low mach number astrophysics code used to simulate the evolution leading up to a type 1a supernova explosion. This particular simulation examines convection in a white dwarf and includes both explicit and implicit solvers. The code uses the BoxLib [10] grid framework, although this particular problem does not use an adaptive mesh, but rather evolves the time-steps on a uniform grid. The MAESTRO benchmark includes an I/O component and represents applications that have small or 'bursty' I/O patterns. This is often the case for adaptive mesh applications that write data one grid element, or box, at a time. These applications often suffer from low I/O performance due to the overhead of many write transactions. The iobuf library provided by Cray could improve I/O performance for MAESTRO and similar applications by buffering I/O on the node and subsequently writing fewer, larger chunks of data at a later time. The MAESTRO benchmark runs on 2048 processors and writes out 3 checkpoint-restart files using a one-file-per-processor model. Each set of checkpoint-restart files produces 153GB of data across 10240 files with most file sizes roughly 10 MBs. Figure 9 shows that linking with the iobuf library, with the default settings improves I/O performance by about 5%. Changing the iobuf parameter settings to use 2 buffers of size 2MB each rather than 4 buffers of 1 MB each improved performance slightly more. (Each of the tests were run multiple times, with the minimum number provided in each case to avoid the variability on the system.) The MAESTRO results are consistent with the performance improvements seen with the iobuf library on the IOR benchmark.

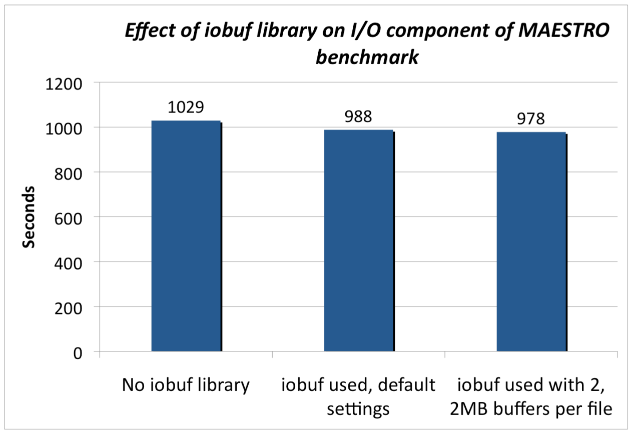

Figure 9: Performance of MAESTRO I/O with iobuf library

We conclude that the iobuf library does improve performance for some user applications. The largest improvement comes from reading data. For writing data, applications that send small amounts of data seem to benefit the most, though this area certainly needs wider parameter exploration. Since the iobuf module is simple to use and requires no code changes, users are encouraged to try it to see if I/O buffering will improve the performance of their application. Users should be aware though, that adding I/O buffers will reduce the amount of memory available to a user application.

In addition to the challenges of getting optimal performance from Hopper's I/O sub-system, on March 24th, 2011 NERSC upgraded the Hopper system to CLE 3.1 UP03 which fixed a number of critical bugs. Unfortunately, the upgrade to CLE 3.1 UP03 decreased aggregate I/O performance across the system. NERSC ran tests with the IOR benchmark on 1560 cores with the POSIX interface with each processor writing 2GB of data to 156 OSTs, (the same setup as in the previous IOR benchmark.) This aggregate benchmark test is intended to saturate the I/O bandwidth of the system and create a sustained I/O rate so the aggregate performance of the system can be measured. Figure 10 shows the performance of the IOR benchmark on a single Lustre scratch file system before and after the CLE 3.1 UP03. Performance dropped by 30-40%. At this time, Cray has a submitted a fix for problem, though it has not yet been put on the Hopper system.

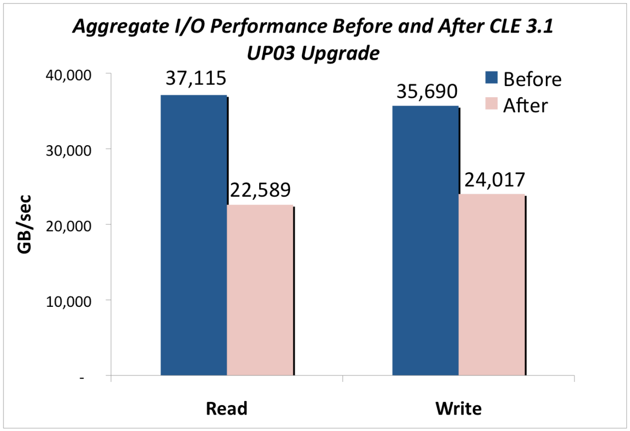

Figure 10: Performance degradation after the CLE3.1UP03 upgrade

3. Problem reports and ongoing issues

Throughout the early user period, NERSC staff and users have exposed bugs on the system, which is to be expected on a new system with a young software stack. So far, over 100 bugs have been opened against the Hopper system and about half of them have been fixed. Figure 11 shows the open bugs by month during the early user period. As expected, the most recent months have the lowest ratio of fixed to open bugs compared to bugs reported in the fall.

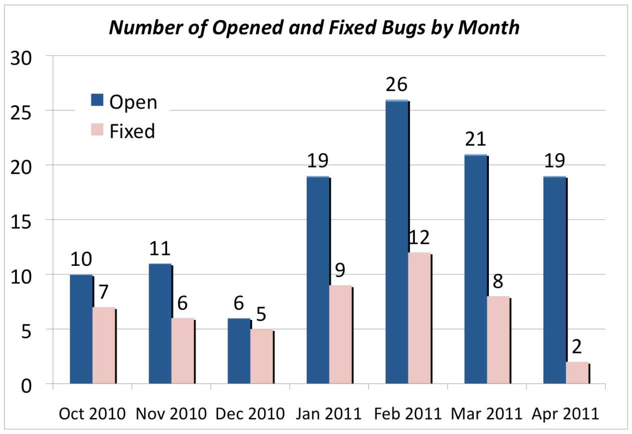

Figure 11: Bugs opened by NERSC and fixed by Cray each month.

The following sections describe some of the specific problems that were exposed on the system.

Low MPI Bandwidth with Small Memory Pages

Through the joint NERSC-Cray Center of Excellence collaboration low level benchmark testing exposed the MPI bandwidth for small pages was 50% the bandwidth for large pages. With the default 4KB pages, the MPI bandwidth between nodes measured 3.5 GB/sec whereas the bandwidth when large memory pages were used was 6 GB/sec. Since most users were running with the default memory pages, this performance problem was affecting a large number of users. The issue was raised with Cray and fixed in a subsequent patch.

Scheduling Problems

In February 2011, there were a few episodes where compute nodes appeared to be available, but jobs were not being scheduled on the nodes causing low system utilization. In some cases the utilization dropped to 50%, a sure sign of a problem as the Hopper queues were regularly packed with jobs waiting to run. The log files showed MOAB warning messages such as:
"WARNING: excessive memory in use (8597 MB) -- restart Moab?" It was found that there were orphan reservations on the system, meaning that the ALPS scheduler believed a job was utilizing a node when it really was idle. This meant that the errant reservations needed to be cleared up manually before the system could return to normal utilization.

The problem was reported to Cray as bug 770068 on February 28, 2011. NERSC discovered that the issues were related to users submitting many one-node jobs using job arrays. Cray's investigation found that the job arrays overwhelmed the ALPS scheduler causing an invalid request to be forwarded to the scheduling agent resulting in many TCP connections being opened and closed repeatedly. Cray added additional error checking, and provided the patch to NERSC a week after the bug was submitted. Meanwhile, NERSC worked with users to submit these types of job arrays more efficiently.

NID Ordering

Job placement on a system as large as Hopper can have an effect on the performance of an application, as nodes belonging to the same job but widely spread across the system can suffer greater latencies and higher communication costs. Cray's original node allocation algorithm for placing an application on the torus was based on NID ordering, where a particular job gets allocated nodes that are physically adjacent to each other, without any awareness of network topology. An improvement was made to the job placement algorithm to make it aware of the interleaving nature of the topology, and is known as xyz ordering. However, the allocation of nodes with either the original NID ordering or xyz ordering was only two-dimensional. In order to allocate nodes in three-dimensional manner that can take advantage of the full torus bisection bandwidth, a new algorithm was introduced on Hopper known as "xyz-by2" [11]. This ordering helps to minimize job run time variation by lowering latencies between nodes within a job and maximizing global bandwidth (network capacity) and bisection bandwidth fore each job.

The "xyz-by2" ordering was used to meet the run time requirements for the set of application benchmarks used for the Hopper procurement. After the OS upgrade to CLE3.1UP03 on March 24, 2011, the Hopper benchmarks were run to check the system and we discovered that the performance of CAM, GTC, Paratec, Impact, NPB, Stream, and Chombo were comparable to those before the upgrade under the production environment. However, we saw a significant slow down for MILC, Maestro, and GAMESS, as much as 30% as shown in Figure 12.

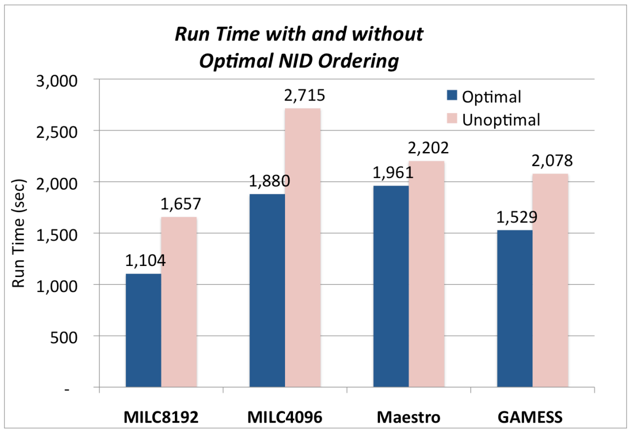

Figure 12: Effect job placement on torus has on runtime for NERSC applications.

The problem was reported to Cray, and it was discovered that the optimal NID ordering algorithm was lost during the OS upgrade. After it was put back on Hopper, running the applications again confirmed that application performance had returned to their previous performance levels.

Mixing C++ and Fortran in the GNU Programming environment

Another problem on the system reported by half a dozen users was found when mixed language C++ and Fortran applications executed in the GNU programming environment. It was found if a code had both C/C++ and Fortran routines, used I/O in a Fortran code, and was built with the GNU compiler, the executable would generate a segmentation fault during run time. This occurred with xt-asyncpe/4.7 and was reported as bug 769872 on February 14, 2011. Cray discovered a symbol from libgfortran.a was not getting resolved, resulting in a faulty memory address. Cray provided a workaround and then the bug was fixed in xt-asyncpe.4.9 released on March 17, 2011.

OpenMP Compiler Flag Options

A problem was found with the ftn, cc, and CC compiler wrappers to the PGI compiler such that the standard -mp=nonuma compiler option did not enable OpenMP. The problem did not occur with the other compilers on the system, Pathscale, GNU and Cray, and the standard flag also worked fine using the native PGI compiler without the compiler wrappers. It was also noticed that the issue was only associated with xt-libsci/10.4.5 and not earlier versions. The bug was reported to Cray on October 19, 2010 as bug 765956 and Cray provided a fix in release xt-libsci/10.5.0, released on December 16th 2010 by replacing -mp with -mp=nonuma in the wrapper for ftn and cc with PGI. However, this fix caused another problem that caused other options to the -mp flag to fail, (such as -mp=trace, or -mp=align). We reported this as bug 768025. Cray provided another fix for the wrappers in xt-asyncpe/4.7 released on January 20th, 2011, but it was found that the problem was only fixed for the FORTRAN compiler wrapper ftn, but not for the C and C++ wrappers. We re-iterated with Cray, and finally got all the PGI compiler wrappers fixed for passing the correct OpenMP compiler flags in xt-asyncpe/4.8 released on February 17th, 2011. This exchange underscores the importance of close communication with the vendor and also the challenges testing all possible use cases all the while mostly communicating through a bug tracking system and through our local Cray onsite staff.

Libsci Dynamic Linking

Originally found on the Franklin system, but even more relevant on the Hopper system which supports dynamically loaded libraries, users reported errors linking codes with the -dynamic compiler flag which resulted in numerous errors about undefined references to FFTW3 functions when xt-libsci/10.4.3 and later were used. Explicitly loading the FFTW3 module solved the compiler errors. Also without the -dynamic flag, the compilations worked successfully. The problem was not seen with xt-libsci/10.4.0. (This problem was originally found and filed as bug 761427 on June 15, 2010 on Franklin.)

Cray's initial suggestion was to change NERSC's default FFTW library from FFTW2 to FFTW3. NERSC was opposed to this suggestion since FFTW2 is the default library on all of NERSC's other systems and this could cause confusion with users. Additionally, FFTW2 and FFTW3 are essentially two products, which could be made into distinct packages instead of using version numbers to distinguish them. Because most users prefer FFTW2 since it supports distributed memory MPI, it is the default module on the system. To address the issue, Cray introduced a new environment variable CRAY_LIBSCI_FFTW_PATH to define the FFTW3 path used by xt-libsci, in versions 10.4.8 and on. NERSC is satisfied with this solution.

Complications Between xt-mpich2 and xt-shmem Modules

Another complication with dynamic linking has been with the interactions between the xt-mpich2 and xt-shmem modules. Cray has deprecated the xt-mpt module, and introduced two new separate modules xt-mpich2 for MPI applications, and xt-shmem for SHMEM applications. According to the MPI man page, "when the xt-mpt module is loaded, the compiler drivers automatically link code using both the -lmpich and -lsma options. This can introduce undesirable dependencies if you are using dynamic or shared libraries." These undesirable dependencies have been observed in some of the dynamically linked user applications with a distinct run time error message of: dmapp_dreg.c:391: _dmappi_dreg_register: Assertion `reg_cache_initialized' failed. Explicitly unloading xt-shmem module and rebuilding the application fixes the problem.

Because NERSC has such a large number of users, choosing an appropriate default environment is critical for reducing problem reports. NERSC had a number of choices for setting the default modules environment at login: A) load xt-mpt only; B) load xt-mpich2 only; C) load both xt-mpich2 and xt-shmem. At NERSC, we chose option C. The advantages of choosing option A are the continuity of the default user environment, and no need to contact users who had an explicit version of xt-mpt loaded in their scripts (as some applications are only validated to run with a certain MPI version.) The disadvantage with option A is that every single compilation with "-dynamic" option turned on would get an error message saying xt-mpt is deprecated, and to load xt-mpich2 or xt-shmem instead. This would affect all codes using dynamic libraries. The advantage of choosing option B is that all MPI applications will compile successfully either statically or dynamically. The disadvantages of choosing option B are that all SHMEM applications will fail. The advantage of choosing option C is that both MPI and SHMEM applications will compile successfully either statically or dynamically. The disadvantage of choosing option C is that some dynamically linked applications may run into the dependency issues between -lmpich and -lsma. Since option C affects the least number of users, this was chosen as the default login environment for all users on Hopper. We documented the need for users to unload xt-shmem to build dynamically linked MPI applications on the NERSC web pages.

Mysterious error messages

Simple easy to understand error messages help users determine reasons why a job or compilation failed. With the transition from Franklin to Hopper, there are a number of new job error messages that are specific to the XE6 system, many of them stemming from the new Gemini interconnect and surrounding software.

One example is:
ERROR - nem_gni_error_handler(): a transaction error was detected, error category 0x4 error code 0xb2e
Rank 0 [Mon Mar 7 03:46:10 2011] [c6-3c1s5n1] GNI transaction error detected

This error message was puzzling as it was accompanied by wide variety of other error messages, including: Fatal MPI error, ALPS error, PGFIO/stdio error, segmentation fault and it seemed that these other error messages were better indications of the true cause of the job failures. Checking with Cray developers proved our thought: the nem_gni_error_handler message is a generic indication of a failure that gives little insight into the problem. We have suggested to Cray that this error message be improved to avoid user confusion.

Another example of a difficult to parse user error is:
ERROR - MPID_nem_gni_check_localCQ(): Replaying failed network transaction
Many of these error messages in one job is usually followed by:
[NID 03782] 2011-04-20 18:45:43 Apid 1925046 killed. Received node failed or halted event for nid xxxx
which indicated the node xxx used by this job went down.

Another common mistake users make is trying to run an executable on Hopper that has been built for Franklin. Binaries for one system are not compatible with the other as an executable which is built for Franklin uses the Portals message interface on the Seastar interconnect while Hopper uses the Gemini interconnect. A distinct error message such as "PtlNIInit failed : PTL_NOT_REGISTERED" helps us to pinpoint the problem and be able to help users easily.

Multiple users reported a similar error message as follows on the same day:
[NID 01083] 2011-04-20 09:39:07 distributeControlMsg: Apid 1919514 write
failure to node 449, 10.128.1.196, port 607, Connection reset by peer

We reported the problem to Cray and it was discovered that the node 449 was purposely downed on the previous day, and accidentally did not get warm booted, though it remained available for scheduling. Warm booting this node brought it back to normal.

Some other error messages users sometimes see on Hopper (and Franklin) are:

- "error while loading shared libraries: libxxxx.so not found". The cause may be CRAY_ROOTFS is not set, or LD_LIBRARY_PATH is not updated with user's own shared objects.
- "OOM killer terminated this process". Suggests user to reduce memory usage or to use fewer cores per node.
- "node count exceeds reservation claim". Check PBS keywords to be compatible with aprun keywords. Do not use more nodes than reserved.
- "segmentation fault". Usually an error in user code.
- "compute nodes initiated termination". Usually an error in user code, and more information in stderr.

We have documented common error messages and our recommendations for troubleshooting the problems on the Hopper web pages. A job completion analysis team has been working on understanding the causes of the job failures and providing the job completion reports on Franklin and Hopper. Still, a clear error messages from the system are a great benefit to NERSC users and staff members.

4. Conclusions

In summary, the early user period on the Hopper system has been successful, delivering over 350 million hours to the Department of Energy Office of Science research community and has enabled users to do research they would not otherwise have been able to do. As soon as the Hopper system came online, immediately it was highly utilized, running applications at large scales. With the new resiliency features of the Gemini interconnect and the added redundancy to the file systems, the Hopper system is significantly more stable than the earlier generation Franklin XT4 machine. We expect to continue to report system and software bugs against the Hopper system as we stress the system and expose new problems going into the production period. We will continue to work with Cray on areas where we can improve the system such as the as I/O performance, dynamic shared library scaling and the usability of the login nodes.

References

1. NERSC web pages for Hopper. http://www.nersc.gov/users/computational-systems/hopper/
2. K. Antypas, T. Butler, J. Carter. *External Services on the Cray XT5 System Hopper*. CUG Proceedings 2010, Edinburgh Scotland.
3. VASP code: http://cms.mpi.univie.ac.at/vasp/
4. N.J. Wright et al. *The NERSC-Cray Center of Excellence: Performance Optimization for the Multicore Era*. CUG Proceedings 2011, Fairbanks, Alaska.
5. D. Wallace and J. Rogers. *DVS*. CUG Proceedings 2008. Helsinki, Finland.
6. T. Butler, G. Butler, R.C. Lee. *DVS, GPFS and External Lustre at NERSC - How It's Working on Hopper*. CUG Proceedings 2011, Fairbanks, Alaska.
7. IOR code: http://computing.llnl.gov. Scalable I/O Benchmark Project. Download: http://sourceforge.net/projects/ior-sio/
8. H. Shan, K. Antypas, J.Shalf. *Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark*. IBM Journal of Research and Development, Supercomputing 2008.
9. MAESTRO code: https://ccse.lbl.gov/Software/index.html
10. A. Nonaka, A.S. Almgren, J. B. Bell, M. J. Lijewski, C. M. Malone, and M. Zingale, "*MAESTRO: An Adaptive Low Mach Number Hydrodynamics Algorithm for Stellar Flows*", Astrophysical Journal Supplement Series, 188, 358-383, June 2010
11. S. Whalen. *"XE6 Job Placement: Node Allocation in an Anisotropic Torus."* NERSC/Cray quarterly meeting presentation. January 27, 2011.

Acknowledgments

We would like to thank Zhengji Zhao for providing VASP code performance data, and Nick Wright for helpful suggestions.

We would like to thank Cray teams (remote and on site support staff), the joint NERSC/Cray Center of Excellence staff, and also our NERSC colleagues for their hard work on Hopper. We would also like to thank NERSC users for their valuable feedbacks.

The authors are supported by the Director, Office of Science, Advanced Scientific Computing Research, U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This work used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

About the Authors

Katie Antypas is the Group Leader of the User Services Group at NERSC. Helen He is a High Performance Computing consultant in the User Services Group. Email: kantypas@lbl.gov, yhe@lbl.gov.