Cray Cluster Compatibility Mode on Hopper

Zhengji Zhao, Yun (Helen) He, and Katie Antypas
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, USA
e-mail: zzhao@lbl.gov; yhe@lbl.gov; kanytpas@lbl.gov

Abstract— **Cluster Compatibility Mode (CCM) is a Cray software solution that provides services needed to run most cluster-based independent software vendor (ISV) applications on the Cray XE6. CCM is of importance to NERSC because it can enable user applications that require the TCP/IP support, which are important parts of NERSC workloads, on NERSC's Cray XE6 machine Hopper. Gaussian and NAMD replica exchange simulations are two important application examples that cannot run on Hopper without CCM. In this paper, we will present our CCM performance evaluation results on Hopper, and will present how CCM has been explored and utilized at NERSC. We will also discuss the benefits and issues of enabling CCM on the petascale production Hopper system.**

Keywords-component; cluster compatibility mode; CCM; TCP/IP; ssh; Cray XE6; performance; ISV; IAA;G09; ccmrun

I. INTRODUCTION

NERSC as a primary production computing facility of Department of Energy Offices of Science supports a diverse workload. More than 5,000 users from all major science fields utilize the computing resources at NERSC and more than 500 code instances run on NERSC machines regularly. NERSC's Peta-flop Cray XE6 system, Hopper [1], is the main work horse to support these computational needs. While most of the NERSC users can conduct their day to day scientific computations on Hopper and get benefits from its extreme scalability and its large capacity (with 153,216 compute cores, 217 TB of memory and 2PB of disk), some of our users can not use Hopper because their codes use the TCP/IP services that are not supported in the native Hopper computing environment. Eg., Gaussian [2], and NAMD [3] replica exchange simulations are two important examples of the codes that can not run on Hopper native environment. In addition, the Cray's Application Level Placement Scheduler (ALPS) deployed on Hopper does not allow multiple processes to be launched on a single node, therefore some applications that need to launch multiple serial executables or multiple parallel executables (each running on a couple cores) on a single node can not run on Hopper without wasting most of the available cores on the node. WIEN2k [4], which is another widely used application at NERSC, is such an example, and has been excluded from running on Hopper due to this constraint. Currently these users who can not run their applications on Hopper have to run their jobs on NERSC's other machine Carver [5], an IBM iDataPlex Intel Nehalem based InfiniBand Linux cluster, which has a much smaller computing capacity (9,984 cores) compared to Hopper. As a result, they have to wait a few times longer to get to run their jobs on Carver. Eg., for the three month time period from 7/1/2011-10/1/2011, the average queue wait time for jobs requesting 3 nodes and 36 hours on Carver, which is the scale that average Gaussian jobs run at, is around four times longer than that on Hopper [6]. Of course, some users may run their jobs on Carver for other reasons. Eg., Carver processor cores have a much faster clock speed, 2.7GHz vs Hopper's 2.1GHz, which is an important factor for small concurrency jobs. In addition, Carver has more memory and queue options for small concurrency long running jobs. However, if we can migrate the TCP/IP workload on Carver to Hopper, then both the migrated and remaining users will get benefit from a shorter queue turnaround which results in greater scientific productivity directly.

The Cray Cluster Compatibility Mode (CCM) is a Cray software solution that provides the services needed to run most cluster-based independent software vendor (ISV) applications on the Cray XE6 [7-9]. It supports the standard Linux services, such as ssh, rsh, nscd, and ldap, and it provides complete root file systems on the compute nodes through the Dynamic Shared Library (DSL) environment using Data Virtualization Service (DVS). MPI runs over TCP/IP and the High Speed Network (HSN). CCM is implemented as a queue that coexists with other queues for the native environment on the system (hereafter we will use Extreme Scalability Mode (ESM) to refer to the native computing environment as contrast to CCM), meaning that CCM dynamically allocates and configures the compute nodes at job start, and releases those compute nodes to the main computing pool upon job exit to be ready for the next job, either an ESM or a CCM job. Therefore it does not require a static partition of the Hopper system, which is one of the most appealing features of CCM. The system can accommodate large CCM workloads up to the whole machine capacity and meanwhile avoid the resource waste when the CCM workload is low over the time.

The Cray CCM is a new feature recently enabled on our production Hopper system on Jan 18, 2012 with the Compute Linux Environment (CLE) upgrade to CLE4.. We have been testing this feature on Hopper's development machine Grace (288 cores) since it was made available on Grace last September. We have tested the CCM with TCP/IP support in CLE3 (CCM1) and the CCM with ISV Application Acceleration (IAA) in CLE4 (CCM2), which utilizes the OFED interconnect protocol over Gemini High Speed Network and therefore should improve performance over CCM1. The performance data we will present are

mostly for CCM2, and we will use the word CCM in the place of CCM2 in our discussion unless CCM1 needs to be mentioned explicitly. In this paper, we will present our performance evaluation results of CCM and how CCM is utilized at NERSC. The paper is organized as follows. After the introduction, we will give an overview on the CCM, focusing on how to use it under NERSC customization in Section 2. In Section 3, we will discuss three applications that use the TCP/IP services and that CCM has enabled on Hopper. We will compare the performance of the applications on Carver and Hopper with CCM an analayze whether performance is sufficient to encourage Carver TCP/IP users to migrate to Hopper CCM. To further evaluate the baseline CCM performance, we conducted the performance tests using a few selected NERSC application benchmarks. This work will be covered in Section 4. In Section 5, we will discuss how CCM is utilized at NERSC, and benefits and issues we have been observing with CCM on Hopper, followed by the Conclusions at the end.

II. Cluster Compatibility Mode on Hopper

Cray intends to provide CCM as an execution environment rather than a development environment. Therefore there is no programming environment support for CCM as there is with ESM from Cray. One can compile codes on any other x86_64 platforms, and then run them "out-of-the-box" under Hopper CCM. However, one can compile codes for CCM on Hopper as well without needing to depend on other platforms to compile codes. CCM can be an environment where the users can compile, run, debug, and profile, and also do data analysis by running visualization tools. Since CCM is open to client site customization, what we describe here is our deployment of CCM on Hopper. For more general and complete information about CCM, we refer to Ref. [9].

A. CCM Programming Environment

All compilers available to Hopper ESM environment can be used to compile codes to run under CCM, namely, the PGI, GNU, Intel, Pathscale and Cray compilers. To compile MPI codes, users have to link the codes to the third party MPI libraries because the Cray custom MPICH2 libraries for ESM (xt-mpich2 modules) don't work under CCM. We have installed the OpenMPI libraries for CCM (as modules). Users compile codes using either the native compiler calls, eg., pgif90, pgcc, pgCC, or the parallel compiler wrappers from OpenMPI, ie., mpif90, mpicc and mpiCC for Fortran, C, and C++ codes, respectively. In contrast to ESM where all binaries are linked statically by default, the executables built for CCM is linked dynamically by default as one would normally expect on a generic Linux cluster. In fact, the OpenMPI runs over ISV Application Acceleration software layer which utilizes the OFED software stack that is available only as shared libraries on Hopper, so MPI codes indeed have to be built dynamically to get the performance boost offered by the IAA. Currently we have provided the OpenMPI library builds using the PGI, GNU, Intel and Pathscale compilers on Hopper, but have not been successful building them with the Cray compiler even with fair amount of staff effort. Cray compiler appears to be difficult to use under CCM. Since there is no programming modules defined specifically for CCM, to switch programming environment from one to another, we still use the PrgEnv modules of ESM, eg., do module swap PrgEnv-pgi PrgEnv-gnu to switch to the GNU programming environment. We used the environment variable PE_ENV defined in the ESM PrgEnv modules in our OpenMPI modulefiles to avoid a separate openmpi module for each compiler. However, most of the software contained in the ESM PrgEnv modules are not relevant to the CCM environment, users can unload them by choice. Users can compile codes either on Hopper login nodes or MOM nodes.

We have provided the most commonly used libraries for CCM. Eg., LAPACK, ScaLAPACK, and FFT libraries. It should be emphasized that all parallel tools and libraries that use the Cray custom MPICH2 libraries for ESM do not work under CCM. However, all serial and threaded libraries built for ESM can still be used under CCM with caution. Eg., we have found that the ACML libraries built for ESM work fine under CCM, and so do the serial FFTW libraries. In addition to borrowing the libraries from ESM, we have also installed ScaLAPACK, and have made MKL available under CCM. To allow debugging and profiling under CCM, we have also provided DDT and the profiling tool IPM. Lastly we have also made MATLAB available under CCM in order to allow interactive data analysis under CCM. Since CCM allows "out-of-the-box" execution, some of the modules mentioned above, eg., MKL, DDT, and MATLAB, are simply pointers to the Carver (Intel Nehalem) builds which reside in a common file system which can be accessed from Hopper. This has saved the support staff's effort significantly from building all the commonly used libraries and tools for CCM separately. In principle, all the libraries, tools and applications built on Carver should run "out-of-the-box" under Hopper CCM (with a slight performance decrease at worst).

B. Running jobs under CCM and CCM runtime environment management

As mentioned earlier, CCM is implemented as a queue on Hopper. Therefore, users need to submit jobs to the ccm_queue to access CCM. Instead of using the ALPS job launcher aprun, CCM jobs are launched using the "ccmrun" [10] command. The ccmrun command places a single instance of the execution commands to the head node of the allocated compute nodes (hereafter, we will call the nodes allocated and configured for CCM jobs as CCM nodes), and then the head node is responsible for launching the executables to the rest of the CCM nodes (remote CCM nodes hereafter) through whatever mechanisms used by the execution commands, eg., mpirun (ssh). There is another

command "ccmlogin" [10] which allows the interactive access to the CCM nodes.

Fig. 1 illustrates how to run a CCM job on Hopper. The job lands on a MOM node as an ESM job would do. Notice a binary built for ESM can still run via aprun under the ccm_queue, but may incur a slight performance loss due to the system overhead from additional environment set for CCM. (However, for small concurrency jobs up to 288 cores, we haven't observe any obvious performance slowdown). In Fig. 1, the ccmrun command places the execution command

"/usr/common/usg/openmpi/1.4.5/pgi/bin/mpirun –np 2 –bynode –hostfile $PBS_NODEFILE hostname" on to the CCM head node, and then the mpirun command is invoked on the head node launching the executable "hostname" on the rest of the CCM nodes using ssh. The second part in Fig. 1 shows how to use the ccmlogin command to login to the CCM head node and then ssh to another CCM node that was allocated to the same job. Where the –V option passes environment on the mom node to the CCM nodes.

Please note the lengthy mpirun command line. Because Cray does not support the native torque launch mechanisms, the OpenMPI library has to be compiled without the batch system awareness (configured with --tm=disable). Therefore the mpirun command from OpenMPI cannot make use of the torque job launch mechanisms, users need to pass the runtime environment to the remote CCM nodes using other options available. Note for CCM jobs, the file $PBS_NODEFILE contains the CCM nodes allocated to the job instead of the MOM node as for an ESM job.

There are a few ways to pass the environment to the CCM nodes. The most reliable method is to add the environment variables to the shell start up files, eg., .bashrc.ext and .cshrc.ext (.ext is a NERSC convention, the .bashrc and .cshrc files are reserved for the system wide settings). These are the files that get sourced by the remote ssh execution. For jobs launching through mpirun, the runtime environment variables can be stored in the ~/.ssh/environment file which the mpirun command sources. Another way of passing environment variables is to use the mpirun's command line option –x and use the absolute path to the OpenMPI installation (or equivalently use the –prefix option). It should be emphasized that one should use the ~/.ssh/environment file with caution when running multiple jobs at the same time, because different jobs may overwrite the ~/.ssh/environment file by each other, and may cause unexpected results. In addition, the leftover ~/.ssh/environment file from a previous job may interfere with next job which does not intend to use ~/.ssh/environment file to pass environment.

```
zz217@hopper04:~/ccm> qsub -I -l mppwidth=48,walltime=1:00:00 -q ccm_queue
qsub: waiting for job 1494288.sdb to start
qsub: job 1494288.sdb ready

In CCM JOB: 1494288.sdb JID 1494288 USER zz217 GROUP zz217
Initializing CCM environment, Please Wait
CCM Start success, 2 of 2 responses
Directory: /global/homes/z/zz217
Sun Apr  8 00:57:04 PDT 2012
zz217@nid05620:~> cd $PBS_O_WORKDIR
zz217@nid05620:~/ccm> module load ccm
zz217@nid05620:~/ccm> export CRAY_ROOTFS=DSL
zz217@nid05620:~/ccm> module load openmpi_ccm
zz217@nid05620:~/ccm> mpicc xthi.c
zz217@nid05620:~/ccm> ccmrun mpirun -np 2 -bynode -hostfile $PBS_NODEFILE --prefix /usr/common/usg/openmpi/1.4.5_mom/pgi ./a.out
libibgni version RB-4.0UP02-4130-2011-11-16-07:27
Hello from rank 0, thread 0, on nid00873. (core affinity = 0-23)
Hello from rank 1, thread 0, on nid00950. (core affinity = 0-23)

zz217@nid05620:~/ccm> ccmlogin –V
zz217@nid00873:~> cd $PBS_O_WORKDIR
zz217@nid00873:~/ccm> mpirun -np 2 -bynode -hostfile $PBS_NODEFILE --prefix /usr/common/usg/openmpi/1.4.5_mom/pgi ./a.out
libibgni version RB-4.0UP02-4130-2011-11-16-07:27
Hello from rank 0, thread 0, on nid00873. (core affinity = 0-23)
Hello from rank 1, thread 0, on nid00950. (core affinity = 0-23)
zz217@nid00873:~> mpirun -np 2 -bynode -hostfile $PBS_NODEFILE --prefix /usr/common/usg/openmpi/1.4.5_mom/pgi hostname
nid00873
nid00950
zz217@nid00873:~/ccm> ssh nid00950
Last login: Sun Apr  8 01:03:36 2012 from nid00873
zz217@nid00950:~> hostname
nid00950
```

Figure 1. This figure illustrates the use of the command "ccmrun" to launch CCM jobs to compute nodes and the use of the command "ccmlogin" to login into compute nodes.

Managing CCM runtime can be tricky and inconvenient, especially when users need to run several executables simultaneously, which require the conflicting runtime environments.

The following are a few sample job scripts to run various jobs under CCM on Hopper. The job scripts 1) and 2) assume the openmpi_ccm module has been loaded in the user's shell start up file .bashrc.ext, otherwise the mpirun command should be replaced with, eg., mpirun –prefix /usr/common/usg/openmpi/default/pgi.

1) Sample job script to run an MPI job

```
#!/bin/bash -l
#PBS -N test_ccm
#PBS -q ccm_queue
#PBS -l mppwidth=48,walltime=00:45:00
#PBS -j oe

cd $PBS_O_WORKDIR
module load ccm

export CRAY_ROOTFS=DSL
mpicc xthi.c
ccmrun mpirun -np 48 -hostfile $PBS_NODEFILE
./a.out
```

2) Sample job script to run an MPI+OpenMP job

```
#!/bin/bash -l
#PBS -N test_ccm
#PBS -q ccm_queue
#PBS -l mppwidth=48,walltime=45:00
#PBS -j oe

cd $PBS_O_WORKDIR

module load ccm
export CRAY_ROOTFS=DSL

mpicc -mp xthi.c
export OMP_NUM_THREADS=6

ccmrun mpirun -np 8 -cpus-per-proc 6 -bind-to-core
-hostfile $PBS_NODEFILE –x
OMP_NUM_THREADS ./a.out
```

3) Sample job script to run an Gaussian job

```
#PBS -S /bin/tcsh
#PBS -N ccm_g09
#PBS -q ccm_queue
#PBS -l mppwidth=48,walltime=24:00:00
#PBS -j oe

module load ccm
setenv CRAY_ROOTFS DSL
set input=myinput
set output=myoutput.$PBS_JOBID
module load g09

mkdir -p $SCRATCH/g09/$PBS_JOBID
cd $SCRATCH/g09/$PBS_JOBID
ccmrun g09l < $PBS_O_WORKDIR/$input >
$PBS_O_WORKDIR/$output
```

where the script g09l is as follows:

```
% cat g09l
#!/bin/csh
setenv GAUSS_EXEDIR
/usr/common/usg/g09/c1/g09/linda-
exe:$GAUSS_EXEDIR
set nodelist="""`cat $PBS_NODEFILE | sort -u`"""
setenv GAUSS_LFLAGS "-vv +getload +kaon -
delay 500 -wait 1200 -workerwait 1800 -mp 24 -
nodelist $nodelist"
setenv GAUSS_SCRDIR `pwd`
g09 $argv
```

4) Sample job script to run multiple serial jobs on a single node

```
#!/bin/bash -l
#PBS -q ccm_queue
#PBS -l mppwidth=24
#PBS –l walltime=1:00:00

cd $PBS_O_WORKDIR
module load ccm
export CRAY_ROOTFS=DSL

ccmrun multiple_serial_jobs.sh
```

Where the script, multiple_serial_jobs.sh, looks like this:

```
% cat multiple_serial_jobs.sh
./a1.out &
./a2.out &
…
./a24.out &
wait
```

Note: The ccmrun wraps the aprun command. One can not launch multiple ccmrun commands on to the same CCM node, therefore the multiple serial binaries have to be launched inside the script multiple_serial_jobs.sh

III. APPLICATIONS THAT CCM EANBLES ON HOPPER

CCM enables applications on Hopper that could only run on Carver previously. In this section, we will compare the performance of three application codes under Hopper CCM with that on Carver. In fact, it is difficult to do any

fair comparison between Carver and Hopper, as two systems are different in every aspect that account for the code performance. However, our focus is to see if the performance of CCM is sufficient to give our users the incentive to migrate to Hopper, so we tried to choose the common practice of our users on two systems and compared the runtime of the codes between two systems. Eg., Carver users run jobs under a GPFS global scratch file system, while Hopper users run their jobs on the local scratch Luster file system which has a much better performance over the Carver global scratch file system. More detailed descriptions about the two machines are as follows.

Hopper, a Cray XE6, has a peak performance of 1.28 Petaflops/sec. Hopper consists of 6,384 compute nodes made up of two twelve-core AMD 'MagnyCours' 2.1 GHz processors. Each node has 24 cores, and two sockets. Each socket contains a Multichip Module with two six-core processors. Thus each node essentially is a four-chip node, and there are large NUMA penalties for crossing the chip boundaries. The majority (6,008) of the nodes have 32 GB DDR3 1.33 GHz memory per node, which is 1.33 GB per core. Hopper compute nodes are connected via the Gemini interconnect via a 3D-torus.

Carver, a liquid-cooled IBM iDataPlex system, has 1,202 compute nodes (9,984 processor cores). This represents a theoretical peak performance of 106.5 Teraflops/sec. All nodes are interconnected by 4X QDR InfiniBand technology, providing 32 Gb/s of point-to-point bandwidth for high-performance message passing and I/O. Note that the above node count includes hardware that is dedicated to various strategic projects and experimental testbeds (e.g., Hadoop). As such, not all 1,202 nodes will be available to all users at all times.

A. Gausssian 2009

Gaussian code (G09) [2] is a computational chemistry code that is one of the most widely used codes at NERSC (rank #23 in 2011). In 2011, it used 1% of total computing cycles [11] at NERSC even though it is usually run at an extremely small scale. We have 314 registered users, and 132 active users who run G09 jobs regularly. G09 consists of many component executables called Links, and the code is parallelized in the master/slave mode. G09 Links are parallelized with OpenMP threads intra-node and with ssh between nodes through the Linda [2] parallel library. While the most of the Links run on multiple nodes, some of them do not, therefore the code does not scale well to the number of processor cores. Most of the G09 jobs run at NERSC are long running jobs using a few nodes. Since Hopper does not support ssh between the compute nodes, G09 could not run on Hopper in the past. Now with CCM, G09 can run on Hopper.

Fig. 2 shows the performance comparison of G09 under CCM on Grace (the Hopper development machine) and on Carver. CCM was made available on Grace last September, and the results shown here are the G09 performance numbers under CCM on Grace. As we have mentioned, G09 consists of many component Links. Fig. 2 shows the sum of the runtime of 3 main component Links in a UHF calculation at two different core counts, 24 and 48, at which our G09 users most like to run their jobs. One can see that G09 runs at around half of the speed (100% slowdown) under CCM on Grace than that on Carver. The performance slowdown is more than what the slower processor speed can account for, which is around 30%. The lack of the process/memory affinity control over ssh could be an important cause of this performance slowdown. This is not CCM specific issue. The same problem exists for Carver as well, however we expect a lower performance hit by this on Carver as it has fewer number of NUMA domains (2 vs Hopper's 4). We ran 24 OpenMP threads per task on each node, instead of the optimal thread/task ratio (6/1) to mitigate the NUMA effects on Hopper (Grace).

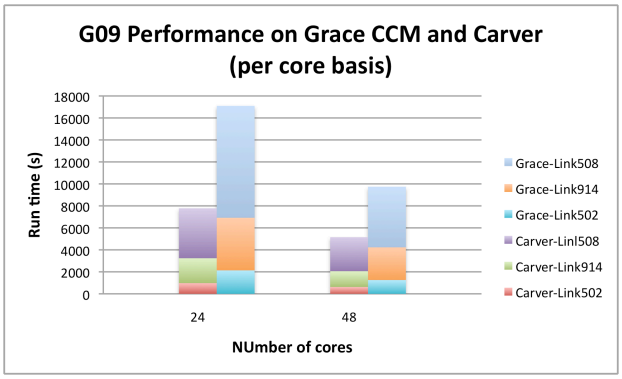

Figure 2. G09 performance comparison (per core basis) between Grace (Hopper's development machine) CCM and Carver. In the figure, the stacked bars show the sum of the runtime spent on the three most time consuming component executables (Links) in a UHF calculation in G09 that run on mulitple nodes (Linda parallelized). G09 were run with 24 and 8 threads per node on Grace and Carver, repectively. The test case (a NERSC user case) contains 61 atoms and 919 number of basis functions.

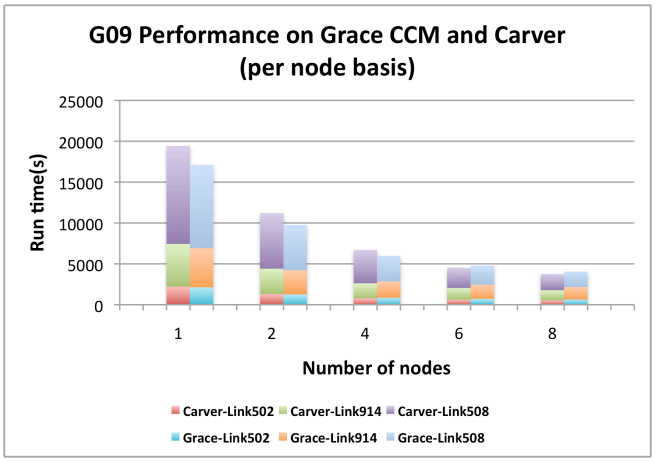

Figure 3. The same as in Fig. 2, but the comparison was done per node basis.

Fig. 3 shows the performance comparison on a per node basis. When 1 node is used, G09 is ~24% faster under Grace CCM than Carver. However, when the number of nodes increases, G09 slows down, and eventually becomes slower than that on Carver, even with 3 times more cores than Carver.

B. NAMD Replica Simulations

NAMD is a classical molecular dynamics code, and it is one of the most widely used codes at NERSC as well. It was ranked the #8 code at NERSC in 2011, and consumed 2.4% of total computing cycles in 2011. We have around 70 active users at NERSC. While the main code works on Hopper, its replica exchange jobs don't run on Hopper. The replica exchange simulation runs many similar job instances (replica) independently at the same time, and occasionally communicates between replicas through the socket operations. Since the socket operations are not supported on Hopper compute nodes, this job type could not run on Hopper in the past. Now CCM enables this simulation on Hopper as well.

Fig. 4 shows the performance comparison of NAMD replica exchange simulations on Hopper CCM and Carver at two different core counts, using a test case provided by a NERSC user. 12 replicas were calculated simultaneously, using 8 and 24 cores per replica, respectively. We run each case 3 times both on Hopper and Carver, and used the shortest run time in the figure.

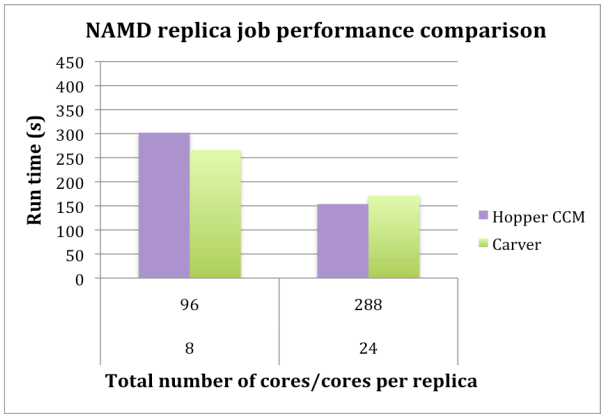

Figure 4. NAMD replica performance comparision between Hopper CCM and Carver. The figure shows that at two different core counts, the time used in the first 1000 MD step in this test case with a 95K atom system (a user case). In both cases, 12 replica jobs run at the same time, at 96 cores, 8 cores per replica (job instance) were used; while at 288 cores, 24 cores per replica were used.

The NAMD replica jobs run around 14% slower on Hopper CCM than on Carver when 8 cores are used per replica, but they run around 10% faster than on Carver when 24 cores used per replica. This could be the result of the slower file system on Carver, as the code writes many small files during the runs. Since the performance depends on how many cores used per replica, we tested the parallel scaling of the pure NAMD code under Hopper CCM and on Carver using a standard benchmark ApoA1 (92K atoms, PME), and showed the results in Fig. 5 and 6 with two different formats. And for reference we also provided the performance numbers for Hopper ESM. In Fig. 6, in the standard benchmark format of NAMD, the flatter the line is the better in parallel scaling.

One can see that NAMD scales fine up to 144 cores under CCM and has a significant scaling drop at 288 cores while Carver and Hopper ESM continue to scale up. Although CCM does not scale as well as Hopper ESM and Carver, it scales sufficiently to allow each replica to use up to 144 cores (for this ~92K atom system). Given the large capacity of Hopper, many more NAMD replicas can run simultaneously, with many more cores per replica, which will bring greater productivity for users.

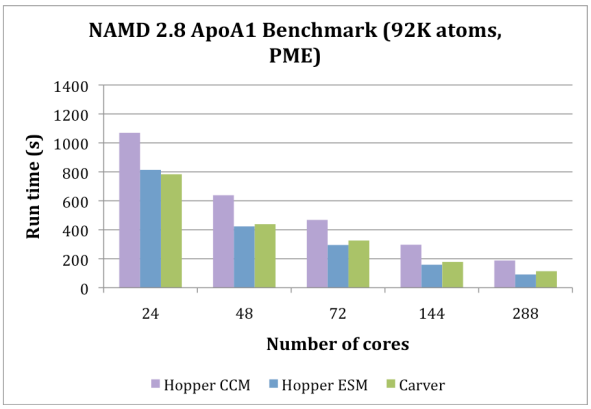

Figure 5. NAMD 2.8 parallel scaling comparison between Hopper CCM, and Carver (Hopper ESM results are also included for reference). The standard ApoA1benchmark (92K atoms, PME) was used.

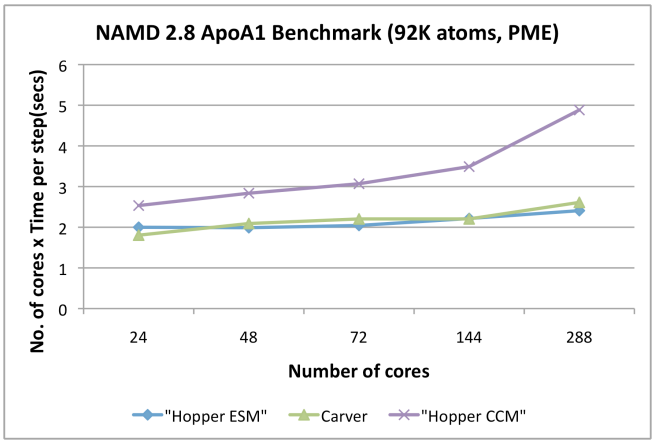

Figure 6. NAMD 2.8 parallel scaling comparison between Hopper CCM, and Carver (Hopper ESM results are also included for reference). The standard ApoA1benchmark (92K atoms, PME) was used.

C. *WIEN2k*

WIEN2k [4] is an ab-initio electronic structure calculation code based on Density Functional Theory. It consists of many component executables connected by shell scripts. It has two layers of the parallel implementation, the k-point and the fine grid parallelization. The former is realized by using ssh and file IO, and the latter is implemented with MPI. WIEN2k is often used for small systems with many k-points, which requires launching multiple job instances (serial or small concurrency parallel executables) on to the same remote node simultaneously using ssh. Since the Cray job placement scheduler, ALPS, does not allow multiple processes to share a single node, WIEN2k could not run on Hopper previously. Even if one can, in principle, replace the ssh command in the WIEN2k scripts with the "aprun –n 1 … &" command to make the k-point parallel execution work on Hopper, it would have to waste most of the 24 available cores. Cray task farmers [12] could be helpful in some cases, but in this specific case the task farmer may not be applicable easily, as the code uses scripts for its complex workflow management including launching parallel binaries from time to time. With the CCM, now WIEN2k runs fine on Hopper as it does on any generic Linux Cluster.

Fig. 7 shows the run time comparison between CCM on Hopper and Carver with a user provided test case. In this case 12 cores were used for each k-point calculation, and 7 k-points were simulated at the same time at the 84 core run, while at 252 core run, all 21 k-points simulated at the same time. The WIEN2k code shows a nice parallel scaling over k-point parallel under both Hopper CCM and Carver (there are very limited communications between different k-points). We can see that at 84 core counts, Hopper CCM is around 90% slower than Carver, however at 252 core counts, Hopper CCM is slower by ~30%. This encourages users to use more cores to shorten the time to solution. Again, we see that if Carver WIEN2k users migrate to Hopper, then they can get benefit from Hopper's larger capacity. On Carver 252 core jobs have to wait much longer time to get started.

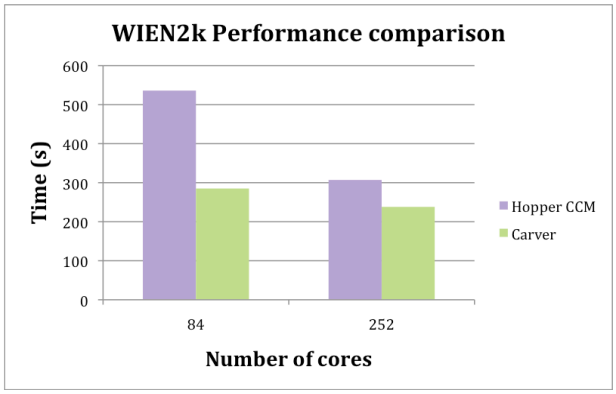

Figure 7. WIEN2k performance comparison between Hopper CCM and Carver. The figure shows that at two different core counts, the time spent in the first cyle of the self consistent electronic step for a mixed k-point (uses ssh) and fine grained parallel (uses MPI) execution. The test system (a user case) has 25 atoms (GaN) and 21 kpoints. Each k-point calculation used 12 cores, therefore 7 kpoints were calucated simultaneously at 84 cores, and at 252 cores, 21 kpoints were calculated at the same time.

IV. PERFORMANCE OF CCM

A. *NERSC Benchmark Applications*

In order to obtain some baseline performance numbers of CCM, we selected four application benchmarks (Table 1) used for the NERSC-6 (Hopper) procurement and run them under both CCM and ESM. The input files were adjusted to fit into the size of the Hopper test machine Grace.

The benchmark code MILC represents part of a set of codes written by the MIMD Lattice Computation (MILC) [12] collaboration used to study Quantum Chromodynamics (QCD), the theory of the strong interactions of subatomic physics. IMPACT-T (Integrated Map and Particle Accelerator Tracking-Time) [13] is a parallel, three-dimensional, quasi-static beam dynamics code used to study dynamics in photoinjectors and RF linear accelerators. The benchmark code PARATEC (PARAllel Total Energy Code) [14] performs ab-initio quantum-mechanical total energy calculations using pseudopotentials and a plane wave basis set. And GTC [15] is a 3-dimensional code used to study microturbulence in magnetically confined toroidal fusion plasmas via the Particle-In-Cell (PIC) method.

TABLE I. NERSC-6 BENCHMARK APPLICATIONS

Benchmark	Science Area	Algorithm	Compiler Used	Concurrency Tested	Libraries
MILC	Lattice Gauge	Conjugate Gradient, Sparse Matrix, FFT	GNU	64, 256, 512, 1024	
ImpactT	Accelerator Physics	PIC, FFT	PGI	64, 256	FFTW
Paratec	Material Science	DFT, FFT, BLAS3	PGI/Intel	64, 256	Scalapack, FFTW
GTC*	Fusion	PIC, Finite Difference	PGI	64, 256, 512, 1024	

* GTC uses weak scaling.

Most applications were run with pure MPI. (GTC also has some hybrid test results). Cray MPICH2 over Gemini network is used for ESM and OpenMPI over TCP/IP is used for CCM (with and without IAA).

B. *Performance Comparison between CCM and ESM*

Each of the four applications was run using pure MPI on Hopper and the Hopper test system Grace where the CCM without IAA was made available for the earlier tests. In this section all Grace runs were done under CCM without IAA.

NERSC-6 benchmarks run time comparison with CCM and ESM using 64 and 256 cores are shown for both Grace

(Fig. 8) and Hopper (Fig. 9). Among these four applications, CCM/ESM run time ratio on Grace ranges between 1.02 times with GTC 64 cores to 2.32 times with MILC 256 cores. And CCM/ESM run time ratio on Hopper ranges between 1.11 times with GTC 64 cores to 1.93 times with MILC 256. The more MPI communications (MILC) an application has, the more slow down of the CCM run time compared to ESM. Also CCM slows down more with 256 cores than with 64 cores.

PGI compiler was used for ESM runs, and Intel compiler was used for CCM runs due to segmentation fault with PGI built executable. ImpactT failed to run at 256 cores with CCM, using both PGI and Intel compilers, with an "Unidentified node: Error detected by IBGNI" error message, so Grace data was used for this data point. A CrayPort bug 783658 has been filed.

NERSC-6 benchmarks scaling comparison between CCM and ESM running with 64 and 256 cores are shown for both Grace (Fig. 10) and Hopper (Fig. 11). N6 benchmarks with 64 to 256 cores ranges from 1.38 to 3.47 with CCM on Grace; and ranges from 1.77 to 3.73 with CCM on Hopper.

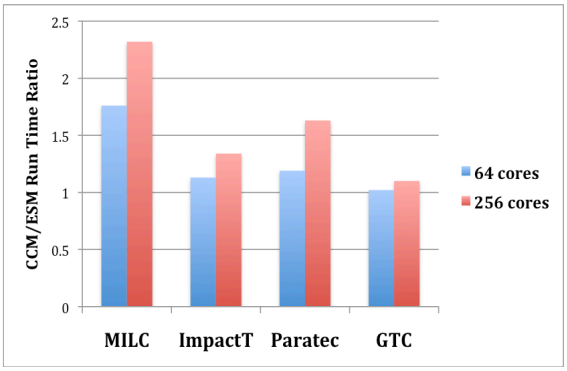

Figure 8. NERSC-6 benchmarks run time comparison between CCM and ESM on Grace.

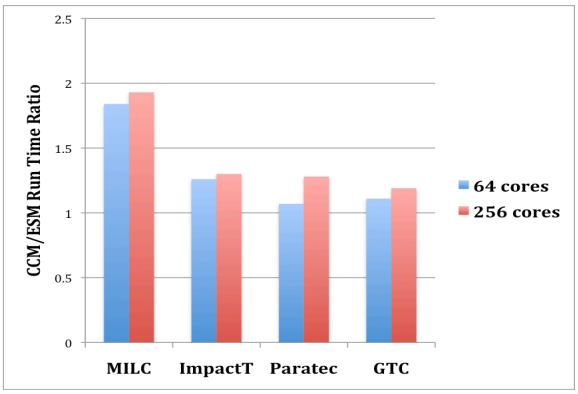

Figure 9. NERSC-6 benchmarks run time comparison between CCM and ESM on Hopper. Paratec: PGI compiler was used for ESM runs, and Intel compiler was used for CCM runs due to segmentation fault with PGI built executable. ImpactT: it failed to run at 256 cores with CCM, so Grace data was used for this data point.

ESM has better speedup than CCM, but CCM is not a lot worse.

Some larger run results were also obtained for MILC and GTC to see the run time performance and scalability of CCM up to 1024 cores (Fig. 12 and 13). CCM runs for these applications using 2048 cores hung (no progress, after printing out "libibgni version RB-4.0UP02-4130-2011-11-16-07:27", then exit until walltime exceeded), although a simple test of "MPI Hello World" completed within 1 minute.

MILC speedup from 64 to 1024 cores is 8.2 under CCM while it is 11.6 under ESM (the ideal speed-up is 16). Here a larger input size for MILC was used compared to the results shown in Fig. 8 to Fig. 11. With the larger input size, a greater speedup from 64 to 256 cores on Hopper CCM, 3.39, is observed compared to the speed-up with the smaller input size (1.77) due to the larger computation vs communication ratio (the ideal speed-up is 4). However, the run time ratio of CCM vs. ESM is still about the same, ranging from 1.83 with 64 cores to 2.6 with 1024 cores.

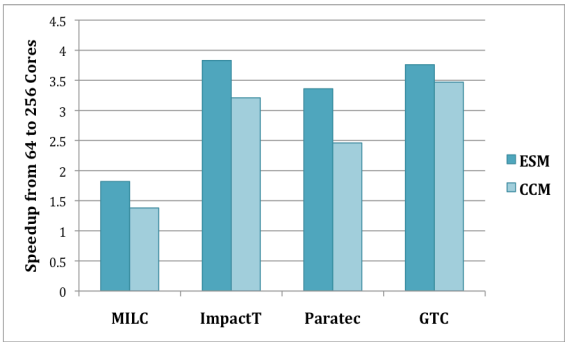

Figure 10. NERSC-6 benchmarks scaling comparison between CCM and ESM on Grace.

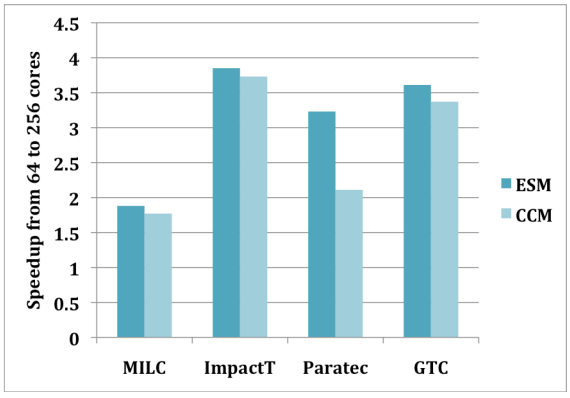

Figure 11. NERSC-6 benchmarks scaling comparison between CCM and ESM on Hopper. Paratec: PGI compiler was used for ESM runs, and Intel compiler was used for CCM runs due to segmentation fault with PGI built executable. ImpactT: it failed to run at 256 cores with CCM, so Grace data was used for this data point.

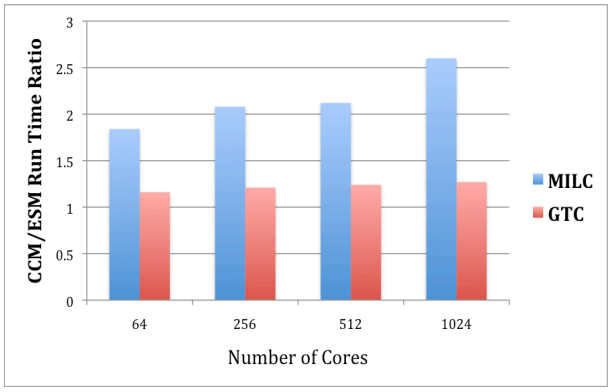

Figure 12. NERSC-6 benchmarks run time comparison between CCM and ESM on Hopper with larger core counts.

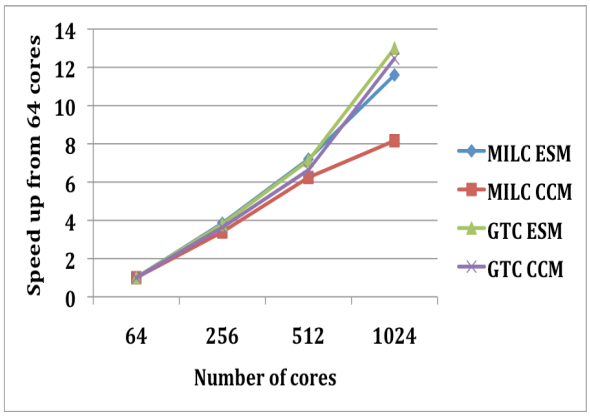

Figure 13. NERSC-6 benchmarks scaling comparison between CCM and ESM on Hopper with larger core counts.

GTC speedup from 64 to 256 cores with ESM is 13, and with CCM is 12.5. And the GTC CCM/ESM run time ratio ranges from 1.16 with 64 cores to 1.27 with 1024 cores. When the application has less communication need, its scaling and run time of CCM better matches to those of ESM.

C. Performance comemnts on MPI+OpenMP hybrid runs

We also examined performance of hybrid MPI and OpenMP runs with CCM using one of the NERSC-6 application benchmarks on Hopper. Fig. 14 shows the GTC hybrid run using 192 total cores with various numbers of OpenMP threads per MPI task.

Both ESM and CCM have a sweet spot at 3 OpenMP threads per MPI task. CCM results are almost identical with ESM results due to minimal MPI communication needs of GTC. Actual CCM/ESM run time ratio ranges from 1.01 times (24 threads) to 1.11 times (1 thread). The CCM performance of hybrid MPI/OpenMP compared to ESM does not have much difference with pure MPI.

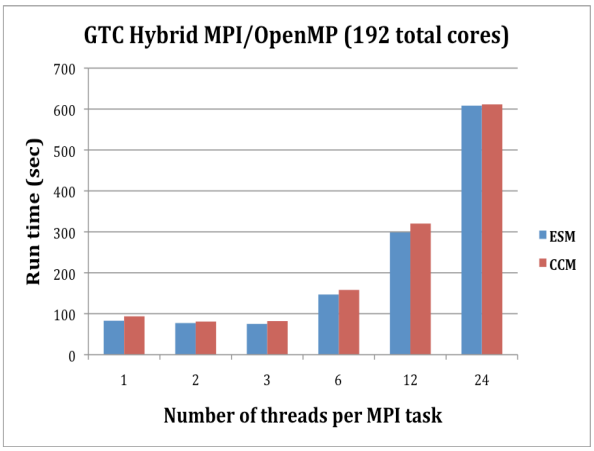

Figure 14. NERSC-6 benchmarks scaling comparison between CCM and ESM on Hopper with larger core counts.

To see more general hybrid code performance under CCM, we did the performance tests under CCM using Quauntum Espresso (QE) code [16], another DFT code, and compared its results to ESM. Compared to GTC, the QE code has a non-trivial MPI + OpenMP implementation. Fig. 15 shows the runtime when the number of threads per task changes for a given number of total cores, 288. The test system contains 112 atoms with two kpoints.

For the hybrid runs that were successful, CCM is around 6% (at threads 24) to 90% (at threads 1) slower than ESM. The code runs fastest at threads=3 as we have seen from the GTC hybrid runs.

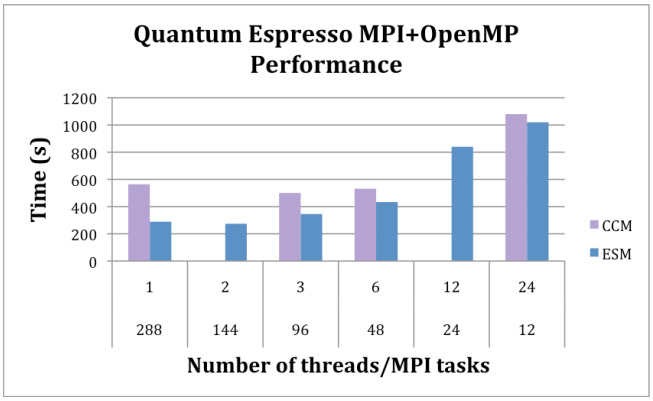

Figure 15. Quantum Espresso MPI+OpenMPI hybrid Performance comparison between CCM and ESM. The figure shows the time spent on the first two self consitent elelctronic iterations when the number of threads changes for the fixed total core counts 288. The test system (standard bechmark AUSRUF112) conaining 112 Au atoms. The code failed under CCM for the number of threads per task 2 and 12. The former failed with segmentation fault, and the latter hung.

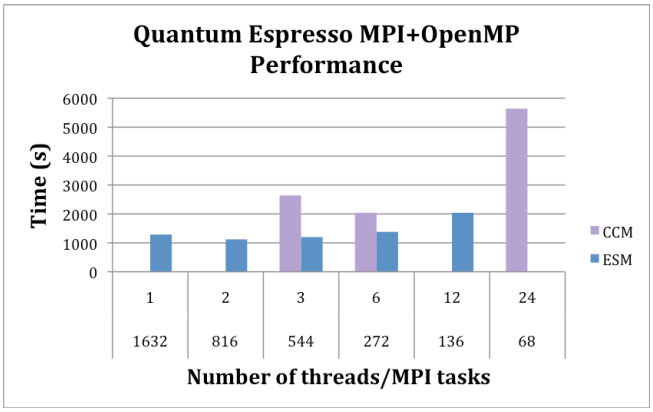

Figure 16. The same as Figure 13, but for a larger test system (standard benchmark, CNT10POR8) containing 1532 atoms using 68 nodes/1632 cores in total. The code failed under CCM for the number of threads per task 1, 2 and 12 with segmentation fault, hang or other errors (eg., libigni error).

On the other hand, QE consistently failed at threads=2 (segmentation fault) and threads=12 (job hang).

We ran QE code with a larger benchmark test case, and with more cores to see if the hybrid code runs at this scale or not under CCM. Fig. 16 shows the results. From the hybrid runs that were successful, the code runs fastest at threads=6 consistenly over the 3 repreated runs. Again, the code failed or hang at threads number 1, 2 and 12.

V. CCM UTILIZATION AT NERSC

As mentioned in the introduction, CCM is of importance to NERSC because it allows Hopper to accommodate our diverse workload and resolve the long queue wait time on Carver.

We started to test CCM on our Hopper test machine since last September, and enabled it on Hopper in production on Jan 18, 2012. We enabled G09, NAMD replica simulation and WIEN2k codes on Hopper under CCM at the same time. On 2/15/2012, we announced the G09 availability on Hopper to our users. And we also contacted a NAMD replica user to try out CCM. Since then there were 60 users (non-NERSC-staff) have tried out the G09 on Hopper so far (4/11/2012). Unfortunately, the users were discouraged by the performance of CCM, most of them didn't stay on running g09 on Hopper after testing its performance. A couple of users requested to increase the max wall limit in order to make CCM to be useful for them. Nonetheless we discovered that more than 13,500 jobs have been run under the ccm_queue and more than 1.3 millions machine hours spent on this queue so far, which is a strong indication of the user need for CCM.

We noticed 3 users have been running NAMD replica jobs regularly with up to 720 cores under CCM, using 12 cores per replica in the most cases. We also identified a few users who need to run serial workloads on Hopper, and recommended CCM to them. One of the use case was that the user needed to run hundreds of serial jobs at the same time, and in each serial job a few short running serial binaries are executed in sequence thousands of times. Unfortunately this job consistently made Hopper nodes OOM, or hang depending on either regular or larger memory nodes the jobs landed on. We opened 2 bugs with Cray. Another use case was that a user code calls multiple instances of G09 (16 instances of G09) at each iteration step, and iterates hundreds of time. This user is still happily using G09 on Hopper.

CCM helped to resolve one of our user reported bug on Hopper. Intel Cilk Plus is a C/C++ extension for improving performance on multi-core processors by spawning multiple workers. XE6 compute nodes do not have the necessary environment for creating Intel Cilk threads. CCM enables it and shows good scaling results using recursive algorithm (Fig. 17).

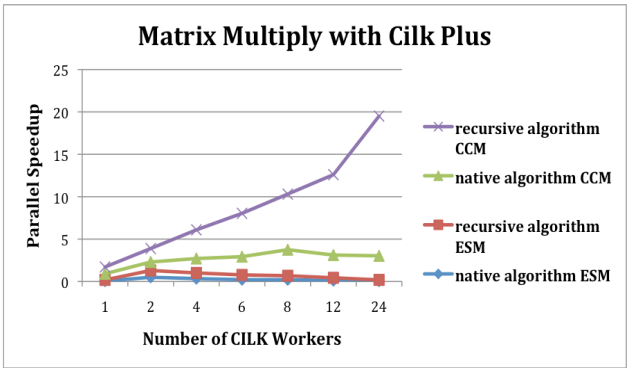

Figure 17. Scaling of matrix multiply with Cilk Plus on Hopper compute nodes.

To promote the CCM usage, we raised the queue priority of the ccm_queue, and also increased the wall time to 96 hours from Hopper native max wall time 48 hours. And to accommodate the interactive workload, we created a new queue ccm_int which has the same priority as the debug/interactive queue. And meanwhile we made the profiling tool IPM and the debugger DDT available under CCM, and also made the Matlab available through CCM. On 4/10/2012, we announced the CCM availability to all NERSC users along with applications and tools available through CCM.

We anticipate more CCM usage from g09 users due to the queue configuration changes. Also we expect more NAMD replica simulation jobs and WIEN2k jobs on Hopper as we haven't announced their availalbility before. We have noticed increasing demands for MATLAB and IDL, and we have recently doubled their license seats at NERSC. Running MATLAB and IDL under CCM gives an advantage of accessing and analyzing data locally without needing to move files from Hopper to Carver/Euclid (NERSC visualization machine), and will allow exclusive access to the compute node memory. We also expect more serial workloads including both cases of a signle user

running on multiple nodes and the multiple users running on a single node, as NERSC has implemented the node sharing between multiple userse under CCM [17]. We expect to have MPMD jobs to run on Hopper without wasting any cores due to aprun doesn't allow node sharing between processes. Probably we will attract some other TCP/IP users who need more flexible workload control over their jobs.

The benefit of having CCM on Hopper is evident, meanwhile we have also observing some issues with CCM on Hopper. As we have discussed in some of the previous secions, the slow performance of CCM is the main concern. We understand that some performance issue is not specific to CCM, eg., the slower process speed, also the lack of process/memory affinity control over ssh and hence G09 jobs might suffer from NUMA penalty greatly, but some performance issues we hope to be still addressable through further imporvement of IAA and DSL. eg., NERSC-6 application benchmark run 10%-90% slower in CCM than in ESM at 256 cores, depending on how heavy the MPI communication is involved. We hope to see a close to ESM performance for CCM jobs running at this scale. In addition, we noticed the CCM jobs often run into various erros, eg., segmentation fault, OOM, libibgni error, and hang, and more. Recently we also discovered that G09 performance degredation when running over multiple nodes (Bug 783803), and also noticed X11 application only run on a signle node (X11 applications do not launch if using the –V options to get on to the head node, Bug 783903) Users need a more reliable CCM to use it in their day to day production computations. We have also seen the extra delays at job start and exit, this is more apprarent when users try to run jobs under CCM interactievely. In addtion, the run time management could be nontrivial and inconvenient some times, especially when one needs to run a few binaries that require conflict runtime environment at the same time.

VI. CONCLUSIONS

CCM enables the applications that couldn't run on Hopper previously, which greatly extends the capability of Hopper being able to accommodate more diverse workloads. The G09, NAMD replica simulation and WIEN2k codes have been enabled to run on Hopper with CCM. In addition, CCM allows serial workloads to run on Hopper as well. This helps NERSC users who have been limited by the lack of TCP/IP services on Hopper compute nodes in the past to get benefit from the larger capacity and the shorter queue wait time of Hopper, which leads to greater scientific productivity. The dynamic queue implementation of CCM not only allows the system to accommodate as large as the whole machine capacity of CCM workloads, but also to assure no computing resource waste when the CCM demand is low at times.

However, the slow performance of CCM has been discouraging users from using it currently. In addition, CCM jobs often hang or run into errors, this is another limiting factor of its utilization in production runs. Since CCM is still in its early developmental stage, we expect that more performing and reliable CCM that will be embraced by NERSC TCP/IP users will be on its way. Our vision is that CCM will be a fantastic feature on a HPC computer with faster processor cores and fewer NUMA domains so that it will not only help accommodate diverse workloads, but also bring higher user satisfaction with improved CCM performance.

ACKNOWLEDGMENT

The authors would like to thank NERSC users who provided the test cases for G09, NAMD, and WIEN2K. The authors also thank Gary Lowell, Randell Palmer and Tara Fly at Cray, Inc for their technical support and help, and also Tina Butler, Nick Wright, Jay Srinivasan, Jack Deslippe and other NERSC staff for their support and useful discussions. This work was supported by the ASCR Office in the DOE, Office of Science, under contract number DE-AC02-05CH11231. It used the resources of National Energy Research Scientific Computing Center (NERSC).

REFERENCES

[1] http://www.nersc.gov/users/computational-systems/hopper

[2] http://www.gaussian.com

[3] http://www.ks.uiuc.edu/Research/namd

[4] http://www.wien2k.at

[5] http://www.nersc.gov/users/computational-systems/carver

[6] https://www.nersc.gov/users/job-information/usage-reports/jobs-summary-statistics/

[7] CLE4.0 release overview: (June 2011)

[8] http://www.slideshare.net/jefflarkin/hpcmpug2011-cray-tutorial, P173-P189.

[9] Tara Fly, David Henseler and John Navitsky, *Case Studies Deploying Cluster Compatibility Mode,* CUG 2012, 4/29-5/3/2012, Stuttgart Germany.

[10] Man pages of ccmrun and ccmlogin commands

[11] Francesca Verdier, 2011 AY code ananlysis, NERSC interanal communication

[12] MILC: http://www.physics.indiana.edu/~sg/milc.html

[13] ImpactT: http://amac.lbl.gov/~jiqiang/IMPACT-T/

[14] Paratec: http://www.nersc.gov/projects/paratec/

[15] GTC: http://w3.pppl.gov/theory/proj_gksim.html/

[16] http://www.quantum-espresso.org

[17] Richard S. Canon, Jay Srinivasan and Lavanya Ramakrishnan, My Cray can do that? Supporting Diverse workloads on the Cray XE-6, CUG 2012, 4/29-5/3/2012, Stuttgart Germany.