Effects of Hyper-Threading on the NERSC workload on Edison

Zhengji Zhao, Nicholas J. Wright and Katie Antypas
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA

E-mail: {zzhao, njwright, kantypas}@lbl.gov

Abstract - **Edison, a Cray XC30 machine, is the NERSC's newest peta-scale supercomputer. Along with the Aries interconnect, Hyper-Threading (HT) is one of the new features available on the system. HT provides simultaneous multithreading capability on each core with two hardware threads available. In this paper, we analyze the potential benefits of HT for the NERSC workload by investigating the performance implications of HT on a few selected applications among the top 15 codes at NERSC, which represent more than 60% of the workload. By relating the observed HT results with more detailed profiling data we discuss if it is possible to predict how and when the users should utilize HT in their computations on Edison.**

Keywords-Hyper-Threading, HPC workload, application performance

I. INTRODUCTION

Edison, a Cray XC30, is NERSC's next petascale machine [1]. One of the new features available on the machine is Hyper-Threading (HT), Intel's simultaneous multi-threading technology. HT makes a physical core appear as two logical cores. These two logical cores have their own architectural states, but share most of the execution resources on the physical core. Two independent processes/threads can run simultaneously on the two logical cores, and when one of them stalls due to cache misses, branch mis-predictions, data dependencies, and/or waiting for other resources, the other process/thread can run on the execution resources which would otherwise be idle, increasing the resource utilization and improving the performance of the processors. HT has shown big potential in the processor design, because it has introduced a new direction and a complementary approach, Thread Level Parallel (TLP), to the traditional technique, Instruction Level Parallelization (ILP), used to improve the processor speed. The ILP approach improves processor speed by increasing the number of execution resources so that more instructions can be executed per clock cycle. Therefore, ILP increases the number of transistors and power consumption on the processor, and leads to a more complex and expensive processor design. HT, on the other hand, has the same goal of improving the processor speed but by increasing the resource utilization by making use of otherwise wasted cycles with only a small increase on the die size and power cost. HT was first introduced on the Intel® Xeon® processor MP in 2002, and with only 5% more die area, Intel observed a 30% performance gain due to HT with common server application benchmarks [2]. In an Intel follow-on analysis with compute-intensive workloads, significant performance gains, up to 30%, were also observed with threaded applications from a wide range of scientific fields [3].

Intel's measured performance of HT was limited to the threaded applications in the past. Given the big potential of HT, it is of great interest to see if HT improves the performance of MPI/MPI+OpenMP codes, which are a large portion of the HPC workload. As the two processes/threads share the physical resources, the speedup from HT will not necessarily be as great as running on two physical cores. On the one hand, HT could benefit application performance by increasing the processor resource utilization. On the other hand, it may also introduce various overheads. Since the two processes/threads share the caches, the cache sizes available for each process/thread will be only one-half of the cache sizes on the physical core. Therefore HT may cause more cache misses compared to the single stream execution on the same physical core. In addition, HT runs twice as many processes/threads on the node; therefore, the memory available per process/thread will be only one-half of the memory available on the physical core, potentially creating additional memory contention. In addition, as we will discuss later in the paper, a proper way to measure the HT performance gain is to compare the performance with and without HT at the same node counts. Therefore applications with HT run using twice as many MPI tasks/threads compared to single stream runs. This may introduce additional communication overhead. Therefore, whether HT benefits an application performance or not depends on whether the higher resource utilization that HT enables overcomes the overheads introduced from using HT. A good parallel scaling is necessarily for HT to realize any performance benefits. There have been a few previous studies regarding the performance impact of HT on HPC workloads [4,5]. The amount of gain observed from various kernel codes and MPI benchmark codes varied widely. Up to 15% performance gain was observed with some of the compute-intensive codes, while some slow down was observed with other codes. These studies have pointed out that HT benefits some applications while hinders other applications depending on the application characteristics and the processor configurations. Some attempts were made to

predict what characteristics of applications could serve as the indicators for HT performance. Ref. [3] used the cycles per instruction and the cycles per micro operation as the indicators for HT opportunity. However, it has been difficult to come up with a set of common indicators for a wide range of codes to predict if an application could benefit from HT, especially for real applications that are far more complicated than the kernel codes that are specially designed to illustrate certain aspects of computations.

HT is enabled on Edison by default. From the user's perspective, HT presents a "free" extra resource available that may improve scientific productivity by reducing time to solution and/or increasing the throughput. Since the benefit of using HT is highly application dependent, it is interesting to examine what major applications at NERSC could benefit from HT. Since NERSC supports a diverse workload and hundreds of different application codes run on Edison, it is also important to provide general HT performance guidance to users. Therefore we will attempt to find some connection between profiling data and the observed HT performance.

The rest of this paper is organized as follows. We will describe the environment where our HT tests were conducted in Section II. In Section III, we will present our results with five selected applications among top 15 application codes in the NERSC workload, and we will analyze and discuss the measured profiling data. We conclude the paper by summarizing our observations in section IV.

II. EXPERIMENT ENVIRONMENT SETUP

A. *Edison*

Edison, a Cray XC30, is NERSC's next petascale machine. It is scheduled to deliver in two phases. The Phase I system was delivered to NERSC in November 2012 and has been in production for a few months. The Edison Phase I system is composed of 664 dual-socket nodes each with 64GB of DDR3 memory running at 1600 MHz. All sockets are populated with 8-core Sandy Bridge processors running at a frequency of 2.6GHz. Edison compute nodes are interconnected with Cray's Aries high-speed network with Dragon Fly topology. The Phase II system is scheduled to arrive in June 2013. It will be populated with Ivy Bridge processors and will have more than 100K cores. The sustained system performance [6] will be 236 TFlops. The HT performance tests presented in this paper were conducted on Edison Phase I system, where HT is made available through the Intel Sandy Bridge Processors.

B. *NERSC Workloads and Application Code Selections*

NERSC serves a broad range of science disciplines from the DOE office of science, supporting more than 4500 users across about 650 projects. As shown in Fig. 1, the most computing cycles were consumed on Fusion Energy (19%), Materials Science (19%), Lattice QCD (13%), Chemistry (12%), and Climate (11%) research. Fig. 2 shows the top applications codes according to the computing hours used (Jan-Nov of 2012) on Hopper [7], a Cray XE6 machine and the NERSC's main workhorse. Among the top 15 codes, which represent more than 60% of the NERSC workload, we selected five application codes and listed them in Table I. We selected these applications based on the ranking and the scientific fields of the codes. We also tried to cover a variety of programming models. In addition, we use the NERSC-6 benchmark suite [8] to measure Edison's system performance. Among the seven application codes in the NERSC-6 benchmark suite, Cray used HT with four of them to meet the performance requirement in the contract for Edison, which covers the Climate, Fusion Energy, Chemistry, and Plasma sciences (see Table II). We selected one code among these four fields, GTC [9], a fusion plasma code, to further investigate the HT effect. We chose VASP [10], which is the #2 code at NERSC, to

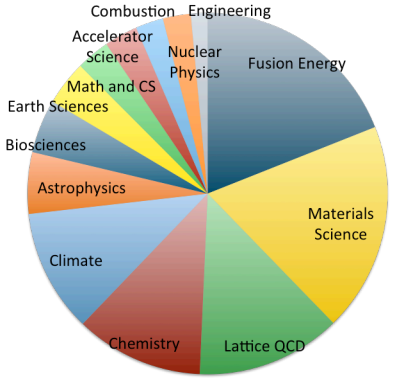

Figure 1. NERSC 2012 allocation breakdown

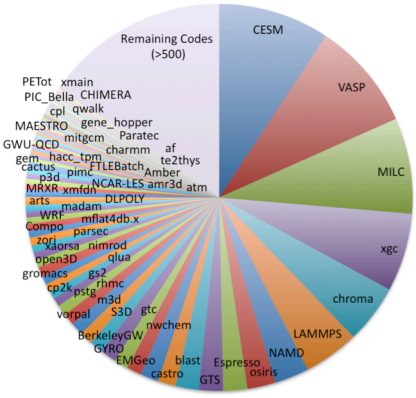

Figure 2. Top application codes on Hopper, Cray XE6 machine by hours used.

TABLE I. SELECTED APPLICATION CODES

Codes	Descriptions	Programming languages and models	Libraries used	Rank
VASP	DFT	Fortran, C MPI	MPI, MKL, FFTW3	2
NAMD	MD	C++ Charm++ (MPI)	Charm++, FFTW2	7
QE	DFT	Fortran, C; MPI, OpenMP	MPI, MKL, FFTW3	9
NWChem	Chemistry	Fortran, C GA, MPI, ARMCI	MKL, GA	13
GTC	Fusion plasma code (PIC)	MPI, OpenMP	MPI	15

TABLE II. NERSC-6 APPLICATION BENCHMARKS

Applications	Descriptions	MPI Concurrencies	HT usage
CAM	Climate	240	Yes
GAMESS	Chemistry	1024	Yes
GTC	Fusion Plasma	2048	Yes
IMPACT-T	Accelerator Science	1024	Yes
MAESTRO	Astrophysics	2048	No
MILC	Quantum Chromodynamics	8192	No
PARATEC	Materials Science	1024	No

represent the materials science workload. We included another materials science code, Quantum Espresso [11] (#9 code) in the tests, because it contains a non-trivial OpenMP implementation in addition to MPI, which is suitable to test the effect of HT on hybrid codes. We chose NAMD [12], a molecular dynamics code, which is widely used by the chemistry and bioscience users (similar to the #6 code, LAMMPS [13], see Fig. 2). NAMD uses Charm++ [14] built on top of MPI as its communication library. We chose NWChem [15], a commonly used chemistry code, which uses Global Arrays (GA) [16], to test the HT effect with the GA programming model.

C. Codes and Test Cases

1) VASP 5.3.3

VASP [10] is a density functional theory (DFT) program that computes approximate solutions to the coupled electron Kohn-Sham equations for many-body systems. The code is written in Fortran 90 and MPI. Plane waves basis sets are used to express electron wavefunctions, charge densities, and local potentials. Pseudopotentials are used to describe the interactions between electrons and ions. The electronic ground state is calculated using the iterative diagonalization algorithms.

We used VASP version 5.3.3 in our tests, and used a test case provided by a NERSC user, which contains 154 atoms ($Zn_{48}O_{48}C_{22}S_2H_{34}$) in the system. The code was built with the Intel compilers and used the MKL for ScaLapack, BLAS and FFTW3 routines. We tested the most commonly used iteration scheme, RMM-DIIS, ran the code over a range of node counts (strong scaling), and reported the total runtime to complete the first four electronic steps.

2) NAMD CVS version 2013-03-28

NAMD [12] is a C++ application that performs molecular dynamic simulations that compute atomic trajectories by solving equations of motion numerically using empirical force fields. The Particle Mesh Ewald algorithm provides a complete treatment of electrostatic and Van der Waals interactions. NAMD was built with the Intel compiler, used the single-precision FFTW2 libraries, and used Charm++ as its communication library. We used the NAMD CVS version 2013-03-28, and tested with the standard STMV (virus) benchmark (containing 1,066,628 atoms, periodic, PME). We ran the tests over a range of node counts (strong scaling), and measured the time to complete the first 500 MD steps.

3) Quantum ESPRESSO 5.2.0

Quantum Espresso [11] (opEn-Source Package for Research in Electronic Structure, Simulation and Optimization) is a materials science program that performs electronic structure calculations and materials modeling at the nanoscale level. Quantum Espresso (QE) is one of the most commonly used DFT codes. It uses a plane wave (PW) basis set and pseudopotentials. The code is written in Fortran 90 and parallelized with MPI and OpenMP.

We used the QE 5.2.0. The code was compiled with the Intel compilers and used the Intel MKL for ScaLapack and BLAS routines, and used the internal FFTW libraries distributed with QE. In the QE benchmark, we tested a self-consistent field (SCF) calculation with a commonly used iteration scheme, Blocked Davidson diagonalization algorithm, with a standard benchmark ausurf112 (containing 112 Au atoms, slightly modified to reduce the amount of IO). We ran the code over a range of node counts (strong scaling), and at each node count ran with different combinations of MPI tasks/threads, and reported the total runtime to complete the first two electronic steps.

4) NWChem 6.1

NWChem [15] is a chemistry application that is designed to be scalable on high performance, parallel computing systems. It is written in Fortran and C, and its parallelization is mainly implemented with Global Arrays. We used the NWChem version 6.1, and tested with the cytosine_ccsd.nw test case from the NWChem distribution, which performs a coupled cluster calculation.

The code was compiled with the Intel compilers and used BLAS routines from the Intel MKL. We ran the code over a range of node counts (strong scaling), and reported the total runtime.

5) 3D Gyrokinetic Toroidal Code

GTC [9] is a 3-dimensional code used to study microturbulence in magnetically confined toroidal fusion plasmas via the Particle-In-Cell (PIC) method. It is written in Fortran 90, and parallelized with MPI and OpenMP. It is one of the NERSC-6 application benchmark codes, which has been used to measure the sustained system performance [14] for Edison. The code was compiled with the Intel compilers and was built without enabling OpenMP directives (in order to be consistent with the standard benchmark runs).

We used the large test case from NERSC-6 benchmark suite, slightly modified to run a fewer iterations in our tests. We ran the code over a range of node counts (strong scaling) and reported the total runtime.

D. Methods

On Edison, HT is enabled in the BIOS by default. Therefore, we were not able to do any tests with HT turned off in the BIOS. In this paper, when we say running jobs with HT, it means running two processes or threads per physical core (dual stream); and by running jobs without HT, it means to run one process or thread per physical core (single stream), which appears as running on the half-packed nodes. It is a runtime option for users to run applications with or without HT. We ran each application with and without HT at the same node counts, and compared the run time. This means jobs using HT use two times as many MPI tasks or threads compared to jobs running without HT. In the previous work mentioned above, Intel VTune [17] was used to profile the applications, which can report accurate and detailed hardware activities on the Intel processors. However, on the Cray XC30, Intel VTune is not supported [18]. To obtain profiling data, we instrumented the application codes with the IPM [19] profiling tool, which can measure the memory usage, the MPI overhead (and detailed MPI profiling), floating point operations and other hardware events available through PAPI [20]. It is worth pointing out that on Sandy Bridge with HT turned on, the floating-point operations could not be measured accurately with PAPI due to the insufficient hardware performance counters available [21]. Therefore we did not use them in our analysis. We measured the total instructions completed (PAPI_TOT_INS), and the total cycles (PAPI_TOT_CYC). Then we derived the cycles per instruction completed for a physical core by (PAPI_TOT_CYC/PAPI_TOT_INS) x (number of logical cores used per physical core). The cycles/instruction metric can be an indicator of whether there are many interruptions (or stalls) during a program execution. Therefore it could serve as an indicator for the HT opportunity as suggested in Ref [3]. Although it is difficult to quantitatively measure all the interruptions occurring during a program execution, especially due to data dependencies, we still tried to measure some of the interruptions that are measurable through the PAPI hardware events available on Sandy Bridge. We also measured L3 cache misses (PAPI_L3_TCM), TLB data misses (PAI_TLB_DM), and conditional branch instructions mis-predicted (PAPI_BR_MSP), which could represent the longer stalls during a program execution. Since only four programmable hardware performance counters are available on Sandy Bridge, we had to run the IPM-instrumented application codes multiple times, each time collecting three different hardware events.

III. RESULTS AND DISCUSSION

Fig. 3 shows the VASP results, where Fig. 3 (a) shows the run time with HT (dual stream) and without HT (single stream) over a range of node counts (strong scaling). Note at each node count, the job with HT ran with two times as many MPI tasks compared to the job without HT. As shown in Fig. 3 (a), HT slows down the VASP code for all node counts instead of improving the performance. The slowdown is about 8% running on a single node, gets larger in percentage when running with a larger number of nodes, and is about 50% when running with eight nodes. Fig. 3 (b) shows the percentage time spent in MPI communication. We can see that due to running with twice as many MPI tasks with HT, the communication overhead for the HT runs is higher compared to the runs without HT at each node count. However, the communication overhead increases by a smaller amount when doubling MPI tasks from the single to the double stream executions at each node count compared to that of doubling MPI tasks by doubling node counts. Fig. 3 (b) shows the code spent about 6-28% of the total runtime on communication, which is an acceptable communication overhead for the VASP code. Fig. 3 (a) and (b) suggest that HT does not benefit VASP and will not likely benefit at any node counts where the code scales.

Figures 4-6 show the analogous results for NAMD, NWChem and GTC in the same format as in Fig. 3. One can see that HT benefits these codes at the smaller node counts, and the performance gain is 6-13%. However, the HT benefit decreases when running with a larger number of nodes, and eventually HT hurts the performance. For example, NAMD runs about 13% faster with HT if running with one or two nodes, but slows down more than 40% if running with 16 nodes. The communication overhead (Fig. 4 (b)) and the parallel scaling (Fig. 4 (a)) suggest that it is preferable to run this job with eight nodes to effectively shorten the time to solution. Unfortunately, HT starts to hurt the performance near this

node count. We see a similar situation with the NWChem code. The only difference is that HT has less of an effect on this code, as the maximum performance gain is around 6% at the single node run. Again we see that HT benefits the runs at small node counts, but slows down the code near the sweet spot of the parallel scaling (near node count 16 or larger). Similarly, the GTC code runs around 12% faster with HT if running with 32 nodes, but the HT performance benefit decreases with the increase of the node counts. At around 256 node counts, HT starts to slow down the codes (Fig. 6 (a)). Fig. 6 (b) shows that this code has a relatively low communication overhead at relatively larger number of nodes. The sweet spot is near the node count 256 or larger, which is outside the HT benefit region (near 32 and 64 nodes). If time to solution is the only concern to users, then HT probably is not very useful to users. However, if users are limited by the allocation hours and have to run with a smaller number of nodes, then HT is helpful to them (NERSC charges the machine hours per physical core). Instead of running at the parallel sweet spot (256 nodes), if running with 64 nodes, HT allows the code to run 10% faster with 10% less charge compared to the run with 64 nodes without HT.

Fig. 7 shows the runtime of QE with different combinations of MPI tasks and OpenMP threads. For an MPI+OpenMP hybrid code, it is desirable to run two threads per physical core with HT, because the two threads on the same physical core may share the cache contents. Fig. 7 (a) is the result of running two threads per physical core with HT, but running one thread per physical core without HT. Fig. 7 (a) shows that there is no observable performance benefit from using HT. Fig. 7 (c) shows a performance gain of up to 15% from HT at two nodes (eight and four threads per MPI task with and without HT, respectively). However, since the runtime at this MPI task and thread combination is much longer than running two threads per physical core with HT (a), the HT performance gain at this MPI task and OpenMP thread combination is probably not relevant to users in practice.

To understand the observed HT effect on these applications, we have attempted to correlate the profiling data with the observed HT performance. Figures 3-6 (d), (e) and (f) show the cycles used per instruction, the L3 cache misses, and the branch mis-predictions per physical core with and without HT for VASP, NAMD, NWChem and GTC, respectively. We were not able to collect the similar data for QE because the IPM available on Edison does not work with the MPI+OpenMP codes. It should be noted that we present these values per physical core instead of per logical core when HT is used to compare with the results without HT. The values per physical core with HT were obtained by adding the values on the two logical cores. As pointed out in Ref. [3] the cycles/instruction metric is an indicator for HT opportunity. If a code spends more cycles retiring a single instruction, it indicates more stalls have occurred during the program execution. From Figures 3-6 (d) we can see that with HT all codes have shown a reduced number of cycles/instruction per physical core, which should be an indication of higher resource utilization from using HT. Meanwhile, we also see that HT introduces extra overheads to program execution. Figures 3-6 (e) show that HT increases the L3 cache misses for a physical core. This is expected because the two processes on the same physical core share the caches (each of them may use only one-half of the caches). In addition, Figures 3-6 (f) show that each physical core has to deal with more branch mis-predictions (NWChem is an exception, probably this is related to the GA programming model). This is also not surprising because now the branch mis-predictions on the two logical cores add up for the physical core. Moreover, the extra communication overhead introduced by running two times as many MPI tasks with HT imposes an additional data dependency across all or many physical cores, which may stop HT from utilizing the idling cycles. Figures 3-6 (d) show that VASP has the highest cycles/instruction among the four codes, which indicates a better chance for HT to improve the code performance; however, we did not observe any performance benefit from HT with VASP. We noticed VASP spends the MPI time almost entirely on MPI_Alltoallv, MPI_Allreduce, MPI_Alltoall, MPI_Bcast, and MPI_Barrier, which may result in some data dependencies across all or many physical cores. This may partially account for the VASP slowdown due to HT. As shown in Figures 3-6 (a) and (d), HT benefit occurred only at the smaller node counts, and a relatively low communication overhead is necessarily for HT to improve the performance. HT should have a better chance to benefit embarrassingly parallel codes for which no or low communication overhead is present. It is worth mentioning again that the communication overhead increase due to doubling MPI tasks from the single to the double stream execution (keeping the number of nodes unchanged) is smaller than that of doubling MPI tasks by doubling node counts with all four codes, which is favorable for HT. As we have listed above, while HT increases the resource utilizations, it introduces extra communication overhead, more cache misses, branch mis-predictions, and other interruptions that are not listed here.

Although we have identified some competing elements that contribute to HT effect, it is difficult to quantitatively predict the HT performance with the profiling data. The contribution from each competing element depends on the application characteristics and the concurrencies at which the applications are run. As an attempt to learn the characteristics of the applications, in Figures 3-6 (c) we show the instructions completed per physical core with and without HT for the codes we examined. This metric represents the workload for a physical core at each node count. Assume for the runs

without HT, the instructions completed per physical core is P + S, where P and S denote the parallel and the sequential portions of the workload, respectively. Since the metric on the physical core with HT is the sum of that on the two logical cores, the instructions completed per physical core with HT can be estimated roughly by (P/2 + S) + (P/2 +S) = P + 2S. This means that if a code contains a large sequential portion, then the instructions completed per physical core with HT would be much larger than that without HT. From Figures 3-6 (c) we can see that when this metric is similar with and without HT (meaning the applications are highly parallelized), the applications have a better chance to get performance benefit from HT.

To summarize, for HT to realize any performance benefit for applications, it seems the following conditions have to be met.

- The cycles/instruction value should be sufficiently large so that HT has enough work to do to help, although HT may not address all the interruptions.
- The communication overhead needs to be sufficiently small. In particular, the extra communication overhead from doubling MPI tasks should be small so that the amount of the interruptions that HT cannot address do not dominate the HT effect. This indicates that HT benefits are likely to happen at relatively smaller node counts in the parallel scaling region of applications except for embarrassingly parallel codes.
- The number of instructions completed per physical core with and without HT need to be similar, which requires highly efficient parallel codes.

On Edison, these numbers can be measured using the easy-to-use IPM tool (with the PAPI_TOT_INS hardware event), so it will be helpful if users look into these numbers to see if their application is a candidate for HT performance benefits. The overall HT performance should be the competing result between the higher resource utilization that HT enables and the overheads that HT introduces to the program execution.

IV. CONCLUSIONS

We investigated the HT performance impact on five applications selected to represent a portion of the NERSC workload. We consider a proper measure of the HT performance effect to be comparing performance with and without HT on the same number of nodes, meaning applications run with two times as many MPI tasks when HT is used. We compared the runtime with and without HT for the five selected applications over a range of node counts (strong scaling). We observed that HT slows down VASP and QE, but improves the performance of NAMD, NWChem and GTC by 6-13% when running with a smaller number of nodes, where the communication overhead is relatively small. However, the HT performance gain for these codes decreases and a big performance penalty occurs when running with a larger number of nodes, where the sweet spot of the parallel scaling of the codes usually resides. For NAMD, NWChem, GTC codes, the parallel scaling sweet spots do not overlap with the HT benefit region; therefore, the HT performance gain occurring with the smaller node counts probably has a limited use to users in practice when the time to solution is the main concern to users. However, if users are limited by the allocation hours, then HT may be helpful to them. It should be noted that the HT benefit is not only application dependent, but also concurrency dependent, i.e., at which node counts an application is run. Therefore blindly using HT may result in a large performance penalty, and users should use HT with caution.

We also attempted to relate the observed HT performance to profiling data such as cycles/instructions, L3 cache misses, branch mis-predictions, and the communication overhead. We were able to confirm that with HT the cycles/instruction per physical core are reduced for all codes, indicating higher resource utilizations. Meanwhile, HT increases L3 cache misses, branch mis-predictions, and other stalls during the program execution, which contribute negatively to the code performance. In addition, the extra communication overhead due to running two times as many MPI tasks with HT contributes negatively to the code performance as well. The overall HT performance should be the competing result between the higher resource utilizations that HT enables and the various overheads that HT introduces to the program execution. Our analysis shows that the applications with higher cycles/instruction could be candidates for HT benefits, although this metric alone is not sufficient to predict the HT effect because HT is not able to address all the interruptions occurring in a program execution. In addition, for HT to realize any performance benefit, low communication overhead and high parallel efficiency are necessary; therefore, the HT benefits are likely to occur at relatively lower nodes counts in the parallel scaling region of applications unless the applications are embarrassingly parallel codes.

HT, as a complementary approach (thread level parallel) to the existing modern technique (instruction level parallel) to improve processor speed, has a big potential in processor design. We observed that the HT benefit is not only application dependent, but also concurrency dependent, occurring at the smaller node counts. Since the gap between the HT benefit region and the parallel sweet spot is relatively large for the major codes we examined in the NERSC workload, and also some of the codes do not get any performance benefit from HT, the HT performance benefit to the NERSC workload is probably limited on Edison at this point. However, with the continuous improvement on the HT implementation, we may expect to see more role of HT in HPC workloads in the future.

VASP

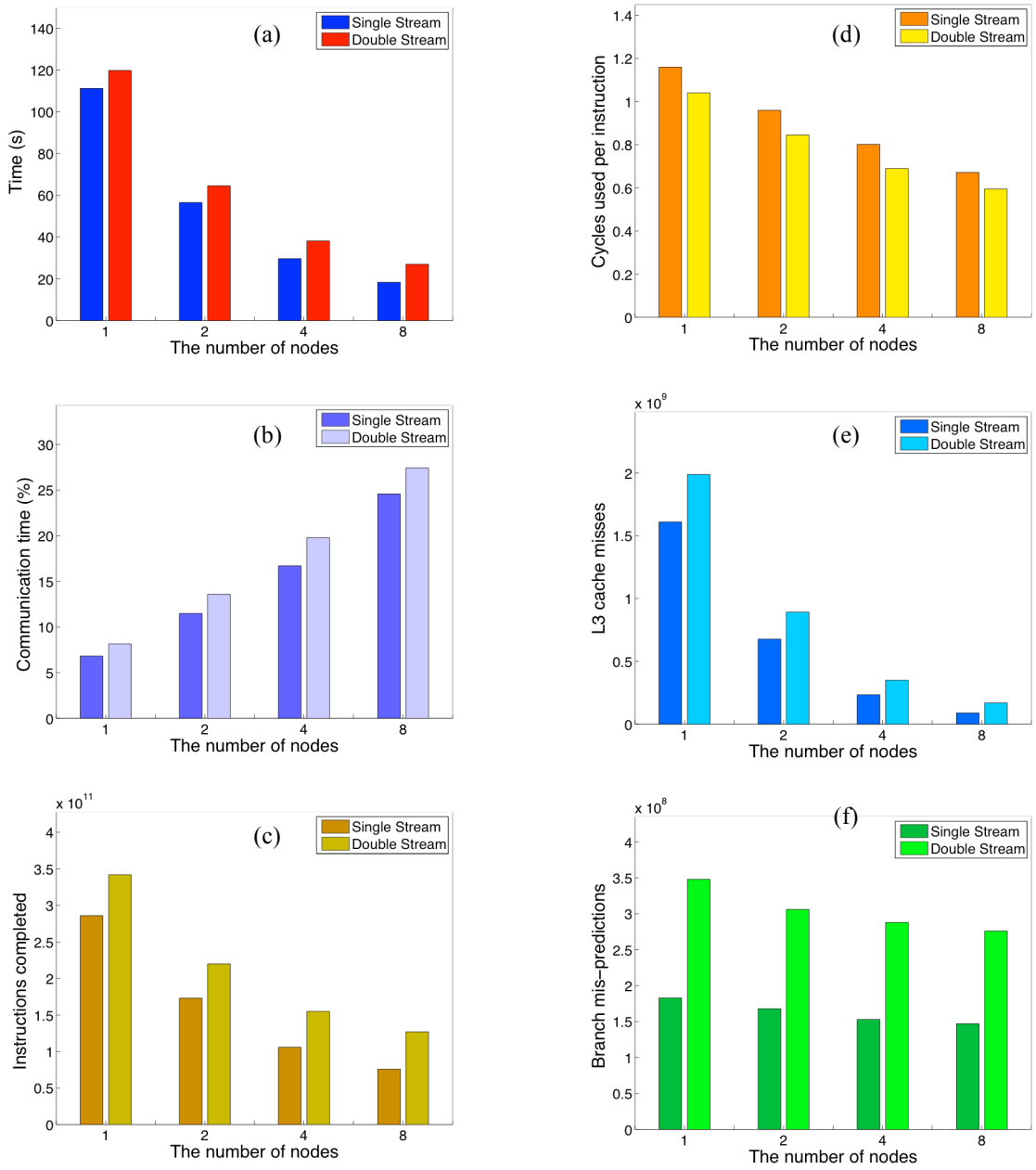

Figure 3. The VASP results, where (a) shows the run time using HT (dual stream) and without using HT (single stream) at a range of node counts with a test case containing 105 atoms. Figure (b) shows the percentage time spent on communication, and Figure (c) shows the instructions completed per physical core with and without using HT, and Figure (d) shows the cycles used per instructions completed per physical core with and without using HT, and the panel (e) shows the L3 cache misses per physical core with and without using HT, and the panel (f) shows the branch mis-predictions per physical core with and without using HT. At each node count, the run with HT used twice the number of MPI tasks.

NAMD

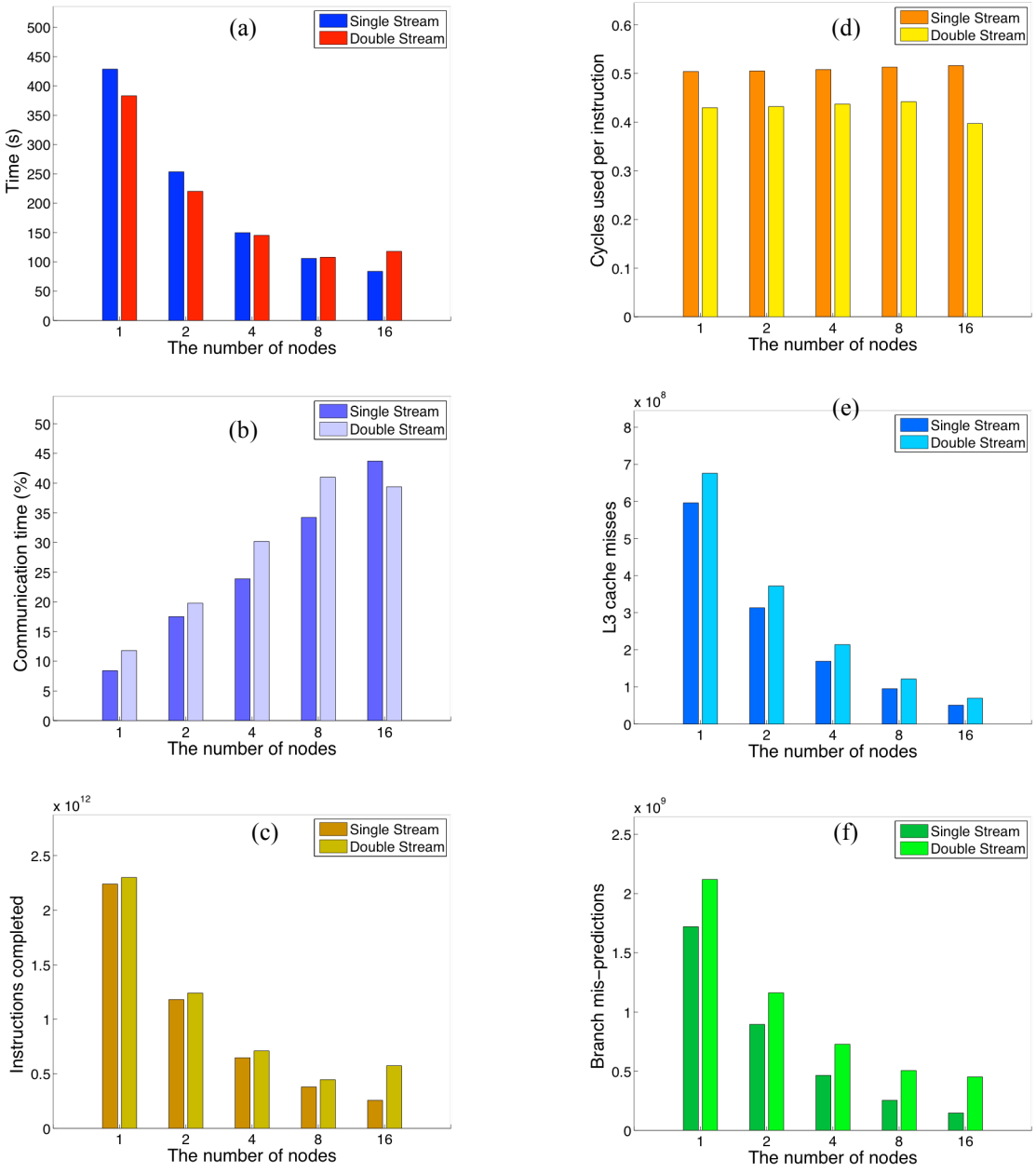

Figure 4. The NAMD results in the same format as in Fig. 3. The STMV standard benchmark case was used.

NWChem

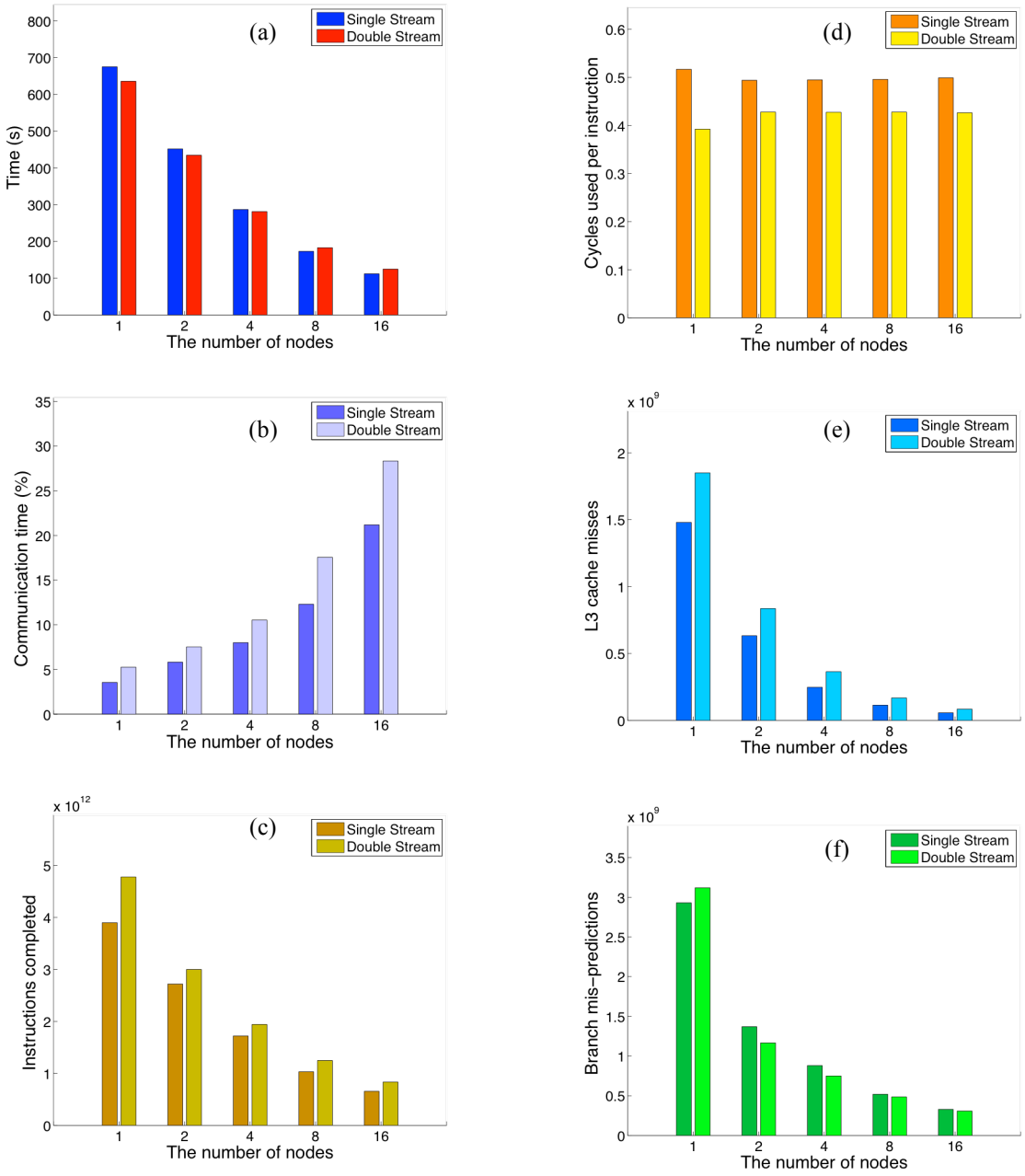

Figure 5. The NWChem results in the same format as in Fig. 3 with a standard benchmark case, cytosine_ccsd.nw. NWchem allows to indicate the memory request in input files. The runs with HT requested one-half of the memory (stack, heap and global memory) of that the runs without HT.

GTC

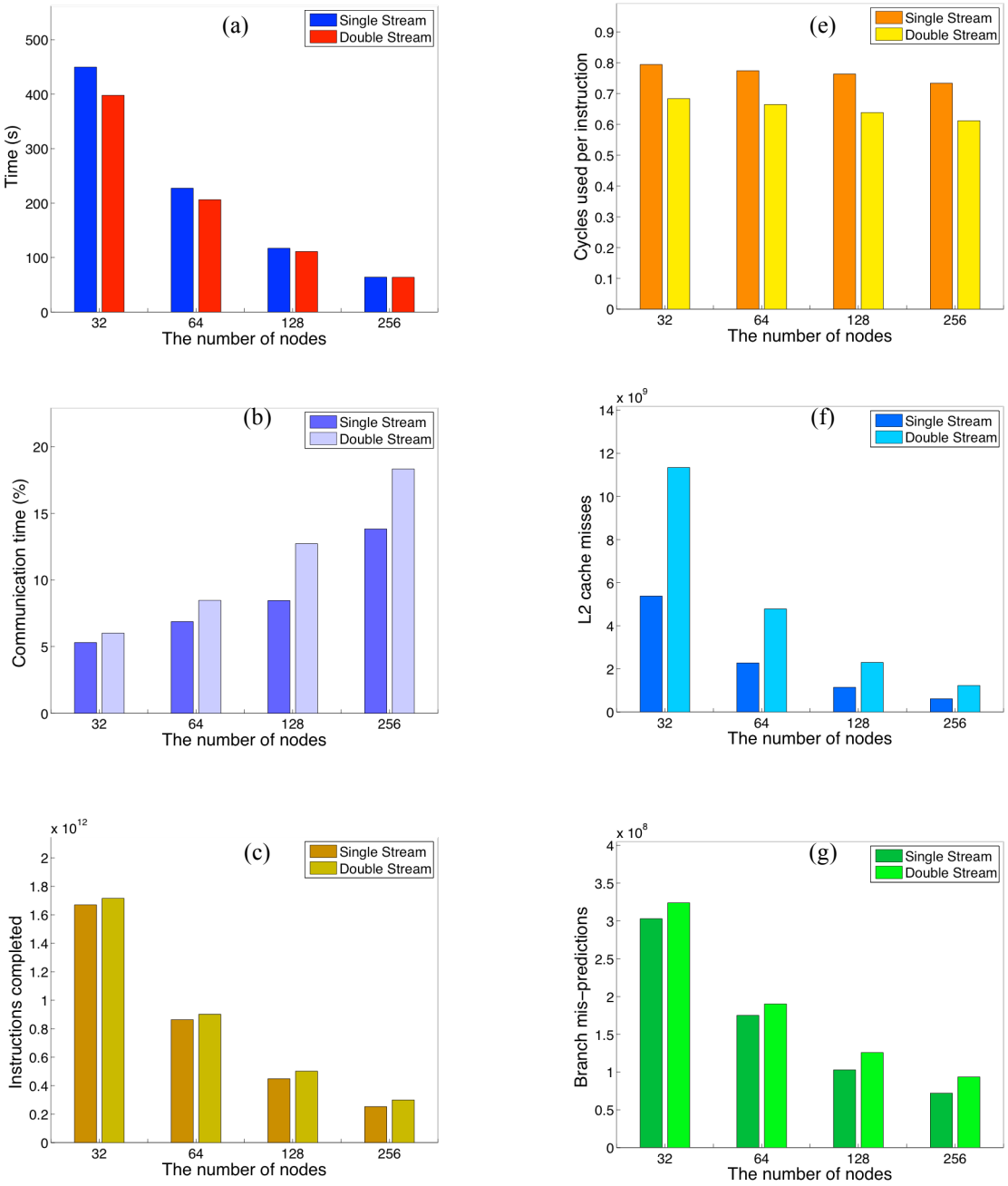

Figure 6. The GTC results in the same format as in Fig. 3 with a slightly modified NERSC-6 benchmark input.

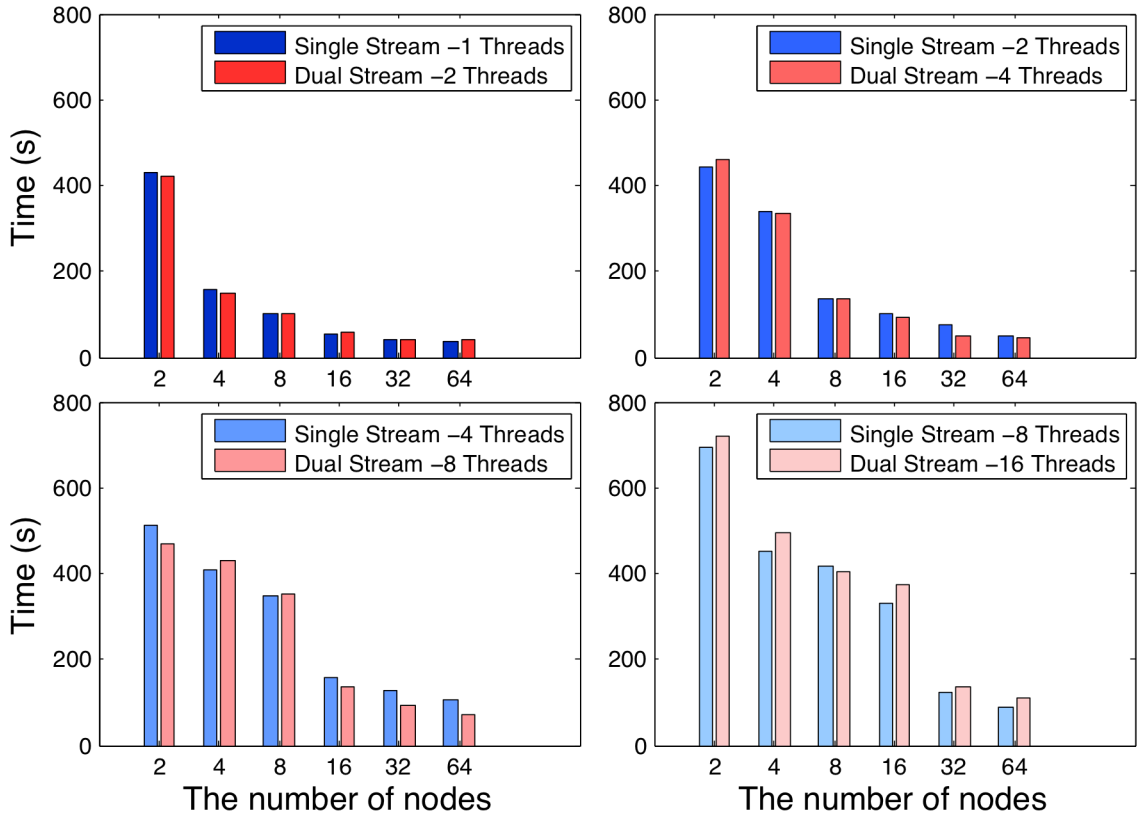

Figure 7. The run time comparison of Quantum Espresso running with HT (dual stream) and without HT (single stream) over a range of node counts with a standard benchmark, ausurf112 (containing 112 atoms). At each node count, the two runs with and without HT used the same number of MPI tasks, but HT runs used twice as many threads per MPI task.

ACKNOWLEDGMENT

Authors would like to thank Larry Pezzaglia at NERSC for his support and help with early HT study on the Mendel clusters at NERSC. Authors would like to thank Hongzhang Shan at Lawrence Berkeley National Laboratory, Haihang You at the National Institute for Computational Sciences, and Harvey Wasserman at NERSC for the insightful discussions and help. Authors would also like to thank Richard Gerber and other members in the User Services Group at NERSC for the 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/edison/

[2] Deborah T. Marr etal., Hyper-Threading Technology Architecture and Microarchitecture, *Intel Technology Journal*, Volume 6, Issue 1 p.11. 2002.

[3] William Magro, Paul Petersen, and Sanjiv Shah, Hyper-Threading Technology: Impact on Compute-Intensive Workloads, *Intel Technology Journal*, Volume 6, Issue 1 p.1, 2002.

[4] http://www.democritos.it/activities/IT-MC/cluster_revolution_2002/PDF/11-Leng_T.pdf

[5] http://www.cenits.es/sites/cenits.es/files/paper_performance_study_of_hyper_threading_technology_on_the_lusitania_supercomputer.pdf

[6] http://www.nersc.gov/research-and-development/performance-and-monitoring-tools/sustained-system-performance-ssp-benchmark

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

[8] http://www.nersc.gov/research-and-development/benchmarking-and-workload-characterization/nersc-6-benchmarks/

[9] http://phoenix.ps.uci.edu/GTC/

[10] M. Marsman. (Oct. 14, 2011). *The Vasp Guide* [online]. Available: http://cms.mpi.univie.ac.at/vasp/vasp/Introduction.html

[11] Quantum Espresso. (2012). *General Documentation* [online]. Available: http://www.quantum-espresso.org/?page_id=40

[12] Theoretical Biophysics Group University of Illinois, (April 5, 2012). *NAMD User's Guide* [online]. Available: http://www.ks.uiuc.edu/Research/namd/2.9b3/ug/

[13] http://lammps.sandia.gov/

[14] http://charm.cs.uiuc.edu/research/charm/

[15] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, W.A. de Jong, "NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations" Comput. Phys. Commun. 181, 1477 (2010); NWChem. (Oct. 5, 2011). *NWChem 6.1 User Documentation* [online]. Available: http://www.nwchemsw.org/index.php/Release61:NWChem_Documentation

[16] http://www.emsl.pnl.gov/docs/global/

[17] http://software.intel.com/en-us/intel-vtune-amplifier-xe

[18] Internal communication between NERSC and Cray

[19] http://www.nersc.gov/users/software/debugging-and-profiling/ipm/

[20] http://icl.cs.utk.edu/papi/

[21] /http://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:SandyFlops