Performance of Density Functional Theory Codes on Cray XE6

Zhengji Zhao, *NERSC and* **Nicholas J. Wright**, *NERSC*

ABSTRACT: *Around one third of the computing cycles at NERSC are consumed by materials science and chemistry users in each allocation year, and, approximately 75% of them run various Density Functional Theory (DFT) packages, the majority of which are pure MPI codes. In this paper, we select two representative DFT codes, VASP and Quantum ESPRESSO, and analyze their performance on the Cray XE6. We focus upon the use of OpenMP and/or the multi-threaded BLAS library to help to address the reduced per-core memory on the Cray XE6 and to improve the performance of the codes.*

KEYWORDS: threads, OpenMP, memory, performance, DFT codes, multicore

1. Introduction

The multicore trend in computing has brought new challenges to application codes that currently run on High Performance Computing (HPC) platforms. Specifically, while Moore's Law continues to double the number of transistors on a chip, these extra transistors are being used to add additional cores instead of increase clock speed as in the past. Thus in order to achieve good performance applications must address ever-increased amounts of parallelism. At the same time as the number of cores is increasing memory densities are also increasing, but not at the same rate. Thus the amount of memory-per-core is decreasing, and will continue to do-so. With the arrival of our new Cray XE6, Hopper, at NERSC, our users have been forced to deal with this multi-core challenge. The machine has 24 cores-per-node, with 1.33 GB of memory per node.

Since these multi-core processors are ideal for the threaded applications, implementing OpenMP in application codes is a possible path forward to get most performance out of Hopper. We have seen that some application codes already have OpenMP implemented. Eg., a widely used density functional theory (DFT) code, Quantum ESPRESSO has already implemented OpenMP in the main code. However, adapting to a new programming paradigm is not necessarily so straightforward for many user communities. Looking at the NERSC workload, the majority of the user applications are still implemented using pure MPI. For example, another DFT code, Vienna Ab-inito Simulation Package (VASP), which consumes the most computing cycles of any code at NERSC, is still running with flat MPI.

Materials science and chemistry applications are an important part of the workload at NERSC and correspond to approximately 1/3 of total allocation hours each year at NERSC, of which 75% corresponds to various density functional theory (DFT) codes. Since DFT codes are an important element in the NERSC workload, it is interesting to see how well DFT codes perform currently on Hopper. In this paper, we will present our performance test results with the two representative DFT codes at NERSC, VASP and Quantum ESPRESSO. Our tests focus on the performance and memory requirements due to the use of a hybrid MPI-OpenMP programming model. As the VASP code does not currently contain OpenMP directives we explore the use of a threaded version of the BLAS library to examine the effect of using a threaded programming model.

This paper is organized as follows. Following this introduction, in section 2 we will give a brief introduction on the Density Functional Theory. In section 3 we will describe the configuration of Hopper and in section 4 and 5 we will discuss the performance of VASP and Quantum ESPRESSO codes from using the threads, respectively. And then we will conclude out our paper with a summary.

2. Density Functional Theory

2.1 Khon-Sham equation

Density Functional Theory (DFT) [1] solves the Kohn-Sham equation [2] of the wavefunctions $\Psi_i(r)$, i=1, 2, …, N,

$$\{-\frac{1}{2}\nabla^2 + V(r)[\rho]\}\psi_i(r) = E_i\psi_i(r)$$

The N is the number of electrons in the system. The potential V(r)[ρ] is a functional of the charge density ρ, which is the sum of the wavefunction squared over the occupied wavefunctions :

$$\rho(r) = \sum_{i=1,Nocc} |\psi_i(r)|^2$$

Under a further approximation (local density approximation), the potential V(r)[ρ] can be approximated as follows,

$$V(r)[\rho] = V(r)\sum_R \frac{Z_R}{|r-R|} + \int \frac{\rho(r')}{|r-r'|}d^3r' + \mu(\rho(r))$$

This is a non-linear eigenvalue problem, and has to be solved iteratively.

2.2 A Flowchart of DFT codes

Figure 1 shows a typical computational flowchart of a DFT code. A DFT code starts from a trial charge density and a set of wavefunctions, from which the potential V(r)[ρ] (V_{in}^{tot}(r) in Figure 1) in the Kohn-Sham equation can be obtained.

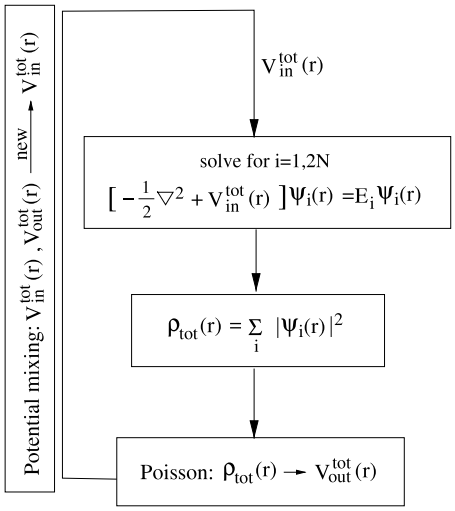

Figure 1 A typical computational flowchart for a DFT code (under the local density approximation).

At this point, the Kohn-Sham equation turns into a linear eigenvalue problem, thus can be solved with one of the available iteration schemes (usually no direct diagonalization) for linear eigenvalue problems. Eg., the residual minimization scheme-direct inversion in the iterative subspace (RMM-DIIS) scheme [3,4], the blocked Davidson iteration scheme [5,6], damped molecular dynamics scheme, etc.. Once a new set of wavefunctions is obtained iteratively, the new charge density (ρ_{tot}(r) in Figure 1) can be calculated from it, so does a new potential (V_{out}^{tot}(r) in Figure 1). If the difference between the input and the output potentials is larger than the required stopping criteria, then start over and repeat the above steps until the stopping criteria meets.

2.3 Planewave DFT codes

DFT codes can be implemented with different basis sets. Eg, the planewave basis, Gaussian basis, etc.. The most commonly used basis in materials science, which deals with the materials with periodicity mostly, is the planewave basis set. The wavefunction $\Psi_i(r)$ for a periodic system can be expanded as sum of planewaves,

$$\psi_i(r) = \sum_G C_{i,k+G} e^{[i(k+G).r]}$$

Where the G is called the reciprocal lattice vector. The electronic state is allowed only at a set of k points determined by the boundary condition that apply to the periodic system [7]. Note the partial differentiation term (kinetic energy term) in the Kohn-Sham equation is diagonal in the **G** space (or the reciprocal space), while the potential term is diagonal in the real space. A frequent Fourier transformation between the real space (**r**) and the reciprocal space (**G**) is involved in the planewave code.

3. Hopper Configuration

Hopper, a Cray XE6, is the NERSC's new petascale machine. It is ranked number five on the November 2010 Top 500 List and has a peak performance of 1.28 Petaflops/sec. It has 212 TB of aggregate memory, and 2 Petabytes of online disk storage. 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 in, as shown in Figure 2. Thus each node essentially is a four-chip node, and there are large NUMA penalties for crossing the chip boundaries. The majority (6008) of the nodes have 32 GB DDR3 1333 MHz memory per node, which is 1.33 GB per core. Hopper compute nodes are connected via the Gemini interconnect, with each pair of nodes is connected to one Gemini chip and they in turn are connected together as a 3D-torus.

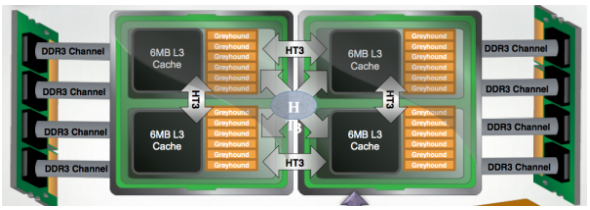

Figure 2 Hopper compute node [8].

4. Performance of VASP on the XE6

4.1 Experimental Setup

The VASP is an application for performing *ab-initio* quantum-mechanical molecular dynamics (MD) calculations using pseudopotentials and a plane wave basis set [9]. Currently it is the most frequently used DFT application at NERSC, and makes up approximately 8% of the NERSC workload [10]. VASP is written in Fortran90 and is parallelized with MPI.

The current version of VASP does not have OpenMP implemented in the code at this time. However, it does spend some of its runtime in Level-3 BLAS routines, thus by linking the code to the Cray multi-threaded scientific library 10.5.01 (-lsci_mc12_mp) we are able to get some measure of the effects of threading upon its performance.

We used VASP v 5.2.11 (released on Jan 18, 2011). The code was compiled with the PGI compiler 10.9.0, with the optimization flags –fast –O3.

We tested multi-threaded VASP on two test cases provided by NERSC users. These are real calculations that they conducted on our systems which we modified for our benchmarking purposes. The first case is for a small system that contains 154 atoms ($Zn_{48}O_{48}C_{22}S_2H_{34}$) and 998 electrons. The calculation was done with a 80x70x140 real-space numerical grid, and the system contains four k-points. This test case will be denoted as A154 hereafter. The second test case is for a medium-sized system containing 660 atoms ($C_{200}H_{230}N_{70}Na_{20}O_{120}P_{20}$) and 2220 electrons. The number of bands is 1456 and the calculation was done with a 240x240x486 real-space numerical grid and one k-point. This test case will be denoted as A660 hereafter.

4.1 Results and Discussion

We tested the performance of VASP 5.2.11 on the two selected cases. Our tests focused on the performance implications from using a hybrid OpenMP + MPI programming model. Therefore, for each test case, we used a fixed number of nodes and ran an MPI-only job and five more jobs where we varied the number of threads whilst keeping the overall concurrency the same. We measured the runtime and the overall memory usage in each case. The memory measurement was done using the NERSC profiling tool IPM [11]. IPM reports the total memory used by the code, therefore we obtained the per-core memory by dividing the total memory reported by the available total number of cores, which is the number of nodes times 24.

Figures 3 and 4 show the time spent on the first 5 electronic steps and the average per core memory for the small test case A154 using 6 nodes, (144 total cores). The time and memory were measured for both the residual minimization scheme-direct inversion in the iterative subspace (RMM-DIIS) (red bars) and the blocked Davidson iteration schemes (blue bars) that are most commonly used algorithms. Note that the flat MPI run on the fully packed nodes (threads =1, MPI tasks = 144) failed due to an out of memory (OOM) error.

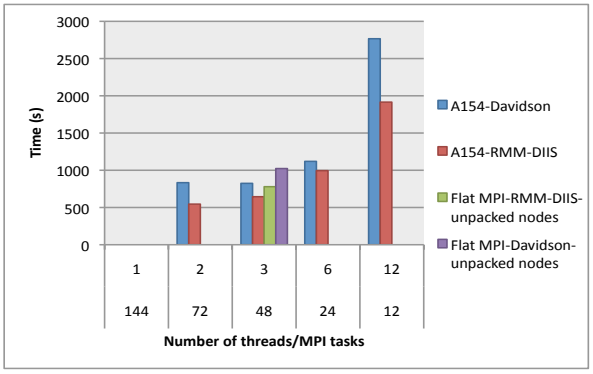

Figure 3 The time spent on the first 5 electronic steps in VASP. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks. Where the red and blue bars are the results corresponding to the two iterative schemes, RMM-DIIS and the blocked Davidson schemes, respectively. All tests were run on 6 nodes. The green and purple bars at threads = 3 are the result of running the flat MPI code on the same number of nodes but using only 8 cores per node (threads = 1, MPI tasks = 48) using the RMM-DIIS and Davidson schemes, respectively.

Figure 3 shows that as the number of threads per MPI task increases and the number of MPI tasks decreases the time to complete the first five electronic steps increases for the RMM-DIIS scheme. The code is simply not spending enough time in the threaded BLAS routines to realize a benefit. The time in the threading library is not zero however; the green bar shows the performance for VASP linked to the non-threaded version of BLAS running on the same number of nodes. Its runtime is 20%

greater than in the threaded case. The results with the Davidson scheme (blue bars) show more benefit from using threading, the runtimes for two and three threads are almost identical, and the increase in runtime from three to six threads is only 35%. Again, we ran the unthreaded version on 48 MPI tasks (purple bar), comparison with the threaded version shows that the threaded version is about 24% faster. In every case we see that for 12 and 24 threads substantial NUMA effects are present, resulting in substantial performance degradation.

Figure 4 shows that the memory usage is reduced in both the RMM-DIIS and Davidson cases as the number of the threads increases. With three threads the memory usage is reduced by ~10% compared to two threads.

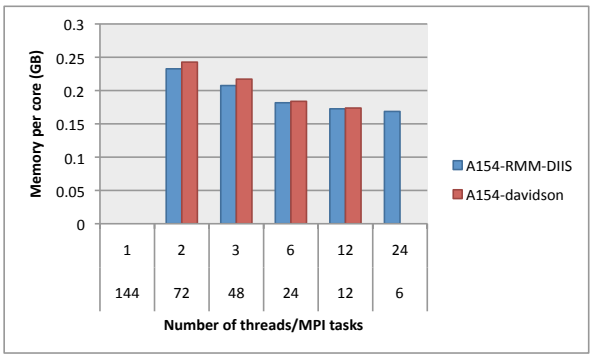

Figure 4 The average per core memory during the first 5 electronic steps in VASP 5.2.11. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks, respectively. All tests were run on 6 nodes. Where the red and blue bars are the results corresponding to the two iterative schemes, the RMM-DIIS and the blocked Davidson schemes, respectively.

Figures 5 and 6 show results for the test case A660 using 32 nodes (768 cores). In this case we see that multi-threading is not providing an overall performance benefit for either the RMM-DIIS or Davidson methods. However, the performance decrease in going from one to two threads, for example, is only 11-13%. Therefore the prospects seem good for implementing OpenMP in the VASP code itself. The memory savings are comparable to the previous case: of the order of 10% for each doubling of the number of threads.

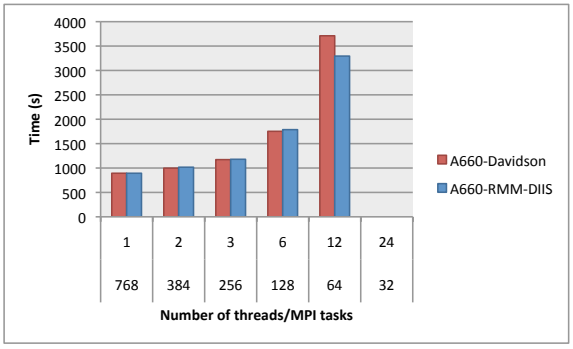

Figure 5 The time spent on the first 5 electronic steps in VASP 5.2.11. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks. Where the red and blue bars are the results corresponding to the two iterative schemes, RMM-DIIS and the blocked Davidson schemes, respectively. All tests were run on 6 nodes. The green bar at threads = 3 shows the result of running the flat MPI code on the same number of nodes with the same number of MPI tasks (unpacked run).

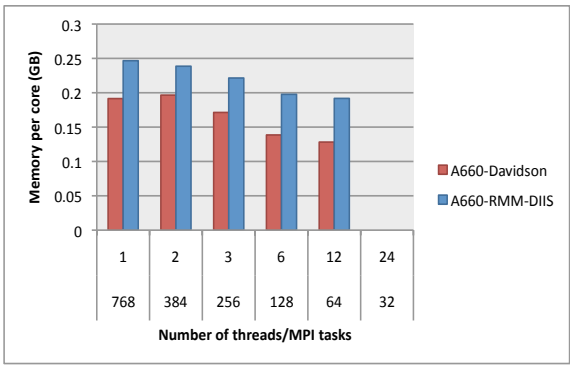

Figure 6 The average per core memory during the first 5 electronic steps in VASP 5.2.11. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks, respectively. All tests were run on 6 nodes. Where the red and blue bars are the results corresponding to the two iterative schemes, the RMM-DIIS and the blocked Davidson schemes, respectively.

5. OpenMP and Performance of Quantum ESPRESSO

5.1 MPI/OpenMP hybrid QE

Quantum ESPRESSO (QE) is an integrated suite of computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and

pseudopotentials (both norm-conserving and ultrasoft) [12].

In this work we use v 4.2.1 which already has OpenMP implemented in the code. It uses multi-threaded BLAS and ScaLPACK routines from the Cray scientific library 10.5.01 (-lsci_mc12_mp) and the FFT routines from ACML 4.4.0. The code was compiled with the PGI compiler 10.9.0, with the optimization flags –fast –O3.

5.2 Test cases

We tested performance on the three standard benchmark cases downloaded from [13], with the slight modifications regarding the file IO and the number of k-points. The first case is a small system that contains 112 gold atoms (Au_{112}). The number of electrons in the system is 1232 (number of bands 800). The calculation was done over 125x64x200 (80x90x288 smooth grids) FFT grids, and the system contains 2 k-points. This test case will be denoted as AUSURF112 hereafter. The second test case is for a medium-sized system containing 686 atoms ($C_{200}Ir_{486}$). The number of electrons in the system is 5174 (number of bands 3104). The calculation was done over 180x180x216 FFT grids, and the system contains 2 k-points. This test case will be denoted as GRIR686 hereafter. The third test case is for a large system containing 1532 atoms ($C_{1164}O_{16}N_{32}H_{320}$). The number of electrons in the system is 5232 (number of bands 2616). The calculation was done over 540x540x540 (375x375x375 smooth grids) FFT grids, and the system contains 1 k-point. This test case will be denoted as CNT10POR8 hereafter. For the small and medium cases the blocked Davidson iteration scheme was used and for the large one the damped MD scheme was used.

5.3 Results and Discussion

In the same way as we did for VASP we report our results for a fixed number of nodes, varying the number of OpenMP threads and MPI tasks simultaneously.

Figures 7 and 8 are the results for the medium QE test case GRIR686 using 60 nodes (1440 cores). The blue bars in Figure 7 show the time to execute the first self-consistent electronic step in QE, and the Figure 8 shows the average per core memory during this run. Note that the flat MPI run on the fully packed nodes (threads =1, MPI tasks = 1440) failed due to the out of memory (OOM) error.

From Figure 7 we can see that when the number of threads per MPI task increases the time to complete the first electronic step increases, especially when threads run across the multiple NUMA nodes (threads > 6). At the threads = 2, the code runs fastest. To see the performance gain from using the OpenMP threads, we ran the flat MPI QE version on the half-packed nodes, ie., running 720 MPI tasks with threads = 1 using the same number of nodes, 60 (purple bar in Figure 5). One can see that the hybrid version runs around 40% faster than the flat MPI version using 2 threads. This indicates the OpenMP implementation in QE is not as efficient as it needs to be to achieve equivalent parallelization between OpenMP and MPI.

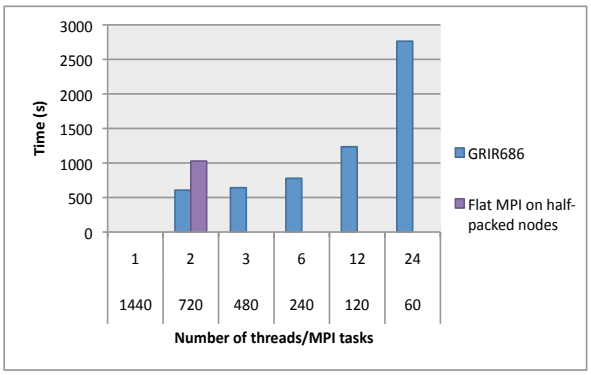

Figure 7 The time spent on the first electronic step for the blocked Davidson iteration Scheme in QE 4.2.1. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks. All tests were run on 60 nodes. The purple bar at threads = 2 shows the result of running the flat MPI code on the half-packed nodes (threads = 1, MPI tasks = 720).

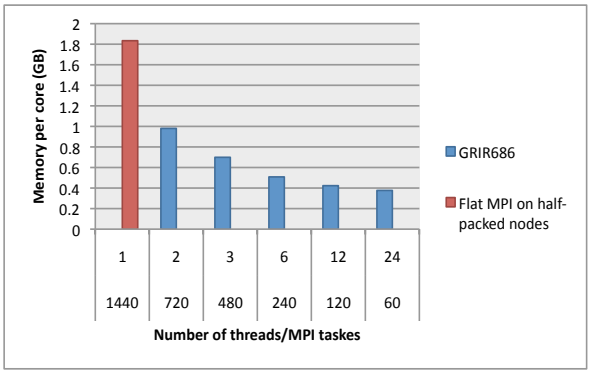

Figure 8 The average per core memory (blue bars) the first electronic step for the blocked Davidson iteration scheme in QE 4.2.1. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks, respectively. Theses tests were run on the 60 nodes. The red bar shows the result of running the flat MPI version on the 120 half-packed nodes (threads = 1, MPI tasks = 1440) for comparison.

From Figure 8 (blue bars) we see that the memory usage reduces when the number of the threads increases. Since the flat MPI run failed on the same number of fully packed 60 nodes (threads = 1, MPI tasks = 1440) due to the OOM error, we conducted a 1440 way flat MPI run on the half-packed 120 nodes (threads = 1, MPI tasks = 1440, the red bar in Figure 6) for reference. One can see that the memory usage is reduced to 64% of the flat MPI run if two threads are used. And the memory is further reduced to 24% of that of the flat MPI run if 6 threads are used.

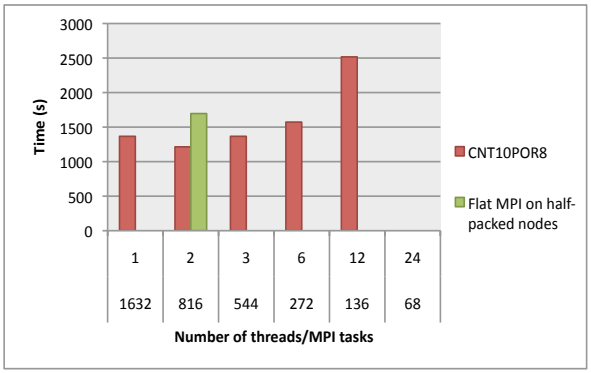

Figure 9 The time spent on the first 2 self-consistent electronic steps in QE 4.2.1. The iteration scheme tested was the damped MD method. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks. All tests were run on 68 nodes. The green bar at threads = 2 shows the result of running the flat MPI code on the half-packed nodes (threads = 1, MPI tasks = 720).

We see similar performance trends and memory reductions for the other two test cases. Figures 9 and 10 show the time spent on the first two self-consistent electronic steps and the average per core memory for the large QE test case CNT10POR8 (1532 atoms) using 68 nodes/1632 cores, respectively; and Figures 11 and 12 show the time and memory results for the small test case AUSURF112 (112 atoms) using 12 nodes, respectively. The green bar in the Figure 9 and the blue bar in Figure 11 are the reference runs on the half-packed nodes. We can see that in both test cases, in addition to the memory requirement decreasing, the hybrid QE version outperforms the flat MPI code at the OpenMP threads = 2. Also, as in the medium case, the two thread case is ~40% faster than the flat MPI one thread result running on the same number of nodes.

From all three cases above, we consistently observed the memory savings and performance gains when threads are used. The memory requirement reduced by up to 10% per core, and the execution time is reduced by ~20%-40% when compared to the flat MPI code.

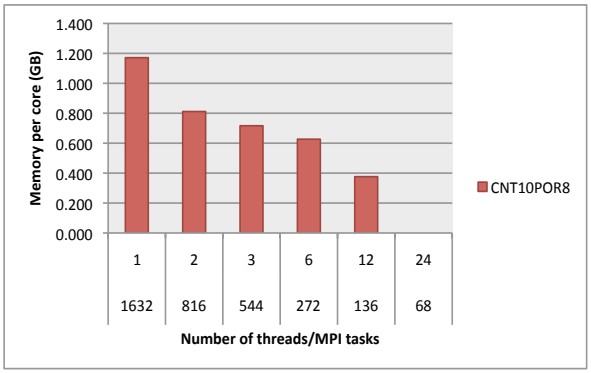

Figure 10 The average per core memory during the first 2 self-consistent electronic QE 4.2.1. The iteration scheme used was the damped MD method. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks, respectively. All tests were run on 60 nodes. The flat MPI code failed to run on the fully packed nodes (threads = 1, MPI tasks = 1440) due to the out of memory error.

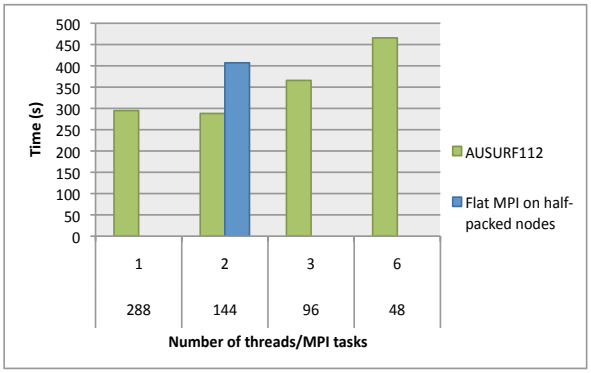

Figure 11 The time spent on the first 2 electronic steps for the blocked Davidson iteration Scheme in QE 4.2.1. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks. All tests were run on 12 nodes. The blue bar at threads = 2 shows the result of running the flat MPI code on the half-packed nodes (threads = 1, MPI tasks = 144).

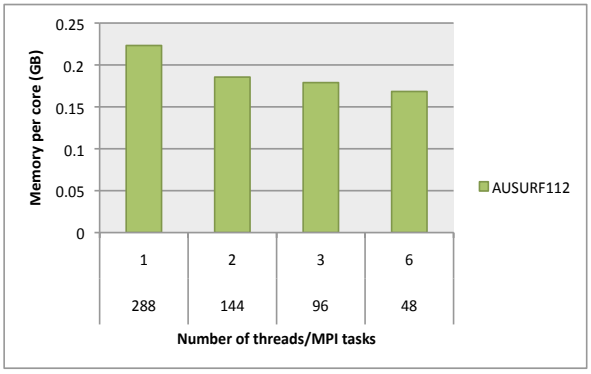

Figure 12 The average per core memory during the first 2 electronic steps for the blocked Davidson iteration scheme in QE 4.2.1. The stacked labels in the horizontal axis show the number of threads per MPI task and the corresponding number of MPI tasks, respectively. All tests were run on 12 nodes.

Conclusions

We examined the performance of two popular DFT codes, VASP and Quantum ESPRESSO on the Cray XE6 machine, Hopper, which is installed at NERSC. Specifically we examined the performance of the codes using the hybrid MPI+OpenMP programming model. Our work was motivated by the decreased memory-per-core available on Hopper as well as the likelihood that on future machines this trend is likely to be exacerbated.

The VASP source code does not contain any OpenMP extensions itself but as it spends a reasonable fraction of its runtime in BLAS routines we linked with threaded versions of those and examined the performance effects. Our results show that VASP performance only benefits slightly from this approach, with performance gains of the order of 20-25% for one of our test case (general kpoint VASP version), but there was no performance gain for the other test case (Gamma point only VASP version). Clearly the addition of OpenMP to the source code could improve this situation considerably.

In contrast the QE code does contain OpenMP extensions. Our assessment shows that this produces performance gains of up to 40% compared to the flat MPI version running on the same number of cores. Also in every case we examined the best performance was achieved using two OpenMP threads, typically with per core memory savings of 20 to 40%.

One advantage of using OpenMP for both these applications is in cases where there is not enough memory per core to run the problem of interest. In this case the cores that would be unused will be utilized, at least somewhat, and the time to solution will be decreased.

In future work we plan to examine the performance of other applications as well as look at the possibility of augmenting a few key routines in VASP with OpenMP directives to improve performance further.

Acknowledgments

The authors would like to thank NERSC users Wai-Yim Ching at University of Missouri - Kansas City and Sefa Dag at Lawrence Berkeley National Laboratory for providing us the VASP test cases. 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).

About the Authors

Zhengji Zhao is a HPC consultant at NERSC at Lawrence Berkeley National Laboratory (LBNL). She supports chemistry and materials science applications at NERSC. She can be reached by email ZZhao@lbl.gov and by phone 510-495-2540. Nicholas J. Wright works in the Advanced Technologies group at NERSC, he can be reached by email NJWright@lbl.gov and by phone 510-486-5730.

References

[1] P. Hohenberg and W. Kohn, Phys. Rev. 136, B864 (1964)

[2] W. Kohn and L.J. Sham, Phys. Rev. 140, A1133 (1965)

[3] D. M. Wood and A. Zunger, J. Phys. A, 1343 (1985)

[4] P. Pulay, Chem. Phys. Lett. **73**, 393 (1980).

[5] E.R. Davidson, *Methods in Computational Molecular Physics* edited by G.H.F. Diercksen and S. Wilson Vol. 113 *NATO Advanced Study Institute, Series C* (Plenum, New York, 1983), p. 95.

[6] B. Liu, in *Report on Workshop "Numerical Algorithms in Chemistry: Algebraic Methods"* edited by C. Moler and I. Shavitt (Lawrence Berkley Lab. Univ. of California, 1978), p.49.

[7] M.C. Payne, M.P. Teter, D. C. Allan, T.A. Arias, and J.D. Joannopoulos, Review of Modern Physics, Vol.64, 1045 (1992)

[8] http://www.nersc.gov/users/computational-systems/hopper/configuration/compute-nodes/

[9] http://cms.mpi.univie.ac.at/vasp/

[10] Francesca Verdier, Code analysis for AY2010, a NERSC internal report

[11] http://ipm-hpc.sourceforge.net/

[12] http://www.quantum-espresso.org/
[13] http://qe-forge.org/frs/?group_id=10&release_id=48