# Optimization Strategies for Materials Science Applications on Cori

[Extended Abstract]

Luther D. Martin
Jackson State Univeristy
522 Giles Lane Lane
Madison, MS 39110
luthermartin@lbl.gov

Zhengji Zhao - Advisor
National Energy Research Scientific Computing Center
415 20th Street
Oakland, CA 94612
zzhao@lbl.gov

## ABSTRACT

NERSC is preparing for its next petascale system, Cori [1], a Cray XC system based on an Intel KNL MIC Architecture. Cori will be delivered to NERSC in the fall of 2016. Each compute node will have 72 physical cores and four hardware threads each. This is 12x the number of physical cores than the NERSCâĂŹs current super computer, Edison [2], a Cray XC30. Each Cori node will have 96GB DDR4 memory (500MB per core), and has a 16GB high bandwidth on-package memory (HBM). Cori also comes with a 512-bits vector unit (twice as wide as EdisonâĂŹs). Currently most of the applications that are running on Edison are pure MPI codes, which will not be optimized to take advantage of the higher on-node parallelism, increased processing power and the HBM. In collaboration with Intel, and Cray, NERSC has developed the optimization strategies [3] to help users to get their applications ready for Cori. MPI+OpenMP has been selected as the parallel programming model for Cori to address the increased on-node parallelism. To take advantage of the larger vector unit, vectorization is an essential optimization for Cori. Exploring ways to make efficient use of the HBM is an important optimization as well. To ease the optimization effort, the use of the profiling tools and libraries are strongly recommended by NERSC. In this poster, we recount the effectiveness of three optimization strategies on the number one production code at NERSC, VASP [4-5], a materials science application code. We focus our optimization effort on the single node optimization, which is critical first step to get application codes to perform on Cori.

## General Terms

Performance

## 1. INTRODUCTION

VASP (5.3.5) is a pure MPI code currently. It is a planewave electronic structure and molecular dynamics code based on the density functional theory (DFT). It solves non-linear eigenvalue problems iteratively, and the FFT and BLAS, LAPACK, ScaLAPCK routines are dominant in the run time. Our goal is to develop viable OpenMP/MPI hybrid schemes to match the performance of MPI implementation on a KNL node, which is a critical step before the planewave codes to make use of 1000s nodes on Cori. Using Intel VTune [6], we first identified three candidate loops in the VASP code to add the OpenMP directives in. Among them, two of the loops were dominated by the BLAS routines (zgemm). Since the optimized threaded libraries for BLAS are available already, e.g., MKL, it appears that calling the threaded zgemm routine from a pure MPI code is an option for VASP. However, the large number of repeated calls to the threaded routines generates a huge overhead due to the thread fork and join (Fig. 1 in the poster). To address this problem, we created an OpenMP parallel region outside the iteration loops (a larger parallel region), and called the threaded zgemm routine with the existing threads in the nested OpenMP mode. This effectively saved overhead from the thread fork and join (Fig. 2), and was able to achieve a typical MPI+OpenMP thread scaling (red and green bars in Fig. 3). We are working on further improve the thread scaling.

The KNL node will be arrived in the fall of 2016, so we do not have access to the HBM node. However, using a heap manager developed by Intel, Memkind [7], and the AutoHBW [7] library tool, which can automatically allocate arrays at specified size range to the HBM without code modifications, we were able to simulate the performance impact from the HBM on the two-socket Ivy Bridge Edison compute nodes. The two sockets on the Edison compute nodes are connected by the QPI (Quick Path Interconnect, see Fig. 6). We use the QPI to simulate the slow memory (DDR memory) while use the memory on the near socket as the HBM. To make efficient use of the limited amount of HBM, it is important to identify the arrays that generate the most memory traffic, and allocate only those arrays to HBM. Using Intel VTune memory-access analysis, we were able to identify the arrays that generate most memory traffic (high Loads in the Fig. 7). This approach requires adding an Intel compiler directive, !DIR ATTRIBUTES FASTMEM, to the user codes, and work with the allocatable arrays only. A more convenient tool to experiment with HBM is the Intel AutoHBW tool [7], which is available within the memkind


*SC* '15 Houston, Texas USA

distribution. The AutoHBW [7] library tool can automatically allocate arrays at specified size range to the HBM without code modifications. See Fig. 8 for the performance difference when allocating arrays in the different sizes to the HBM. We observed about 30 percent performance difference. It is expected that on KNL we will see even larger performance difference due to the larger bandwidth difference between HBM and DDR memory (5x).

Due to the increased vector unit on KNL, it is important to get the code vectorized. We used Intel Vectorization advisor to indentify the vectorization hotspots, and vectorized the loops in the two subroutines, forloc, and forhar, by removing the loop dependencies, and were able to speed up the two subroutines by 24

As a conclusion, OpenMP is the direction to go for highly parallelized architecture. There are notable performance gains when implementing OpenMP around loops. Global parallel regions cut down on the overhead cost of forking new parallel regions. There was a 30

## 2. REFERENCES


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

[2] http://www.nersc.gov/users/computational-systems/edison/

[3] http://www.nersc.gov/users/computational-systems/cori/application-porting-and-performance/getting-started-and-optimization-strategy/

[4] VASP: http://www.vasp.at/

[5] G. Kresse and J. FurthmÃijller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. Comput. Mat. Sci., 6:15, 1996

[6] Intel Vtune: http://software.intel.com/en-us/intel-vtune-amplifier-xe

[7] Intel Advisor: http://software.intel.com/en-us/intel-advisor-xe

[8] Memkind and Autohbw: http://github.com/memkind/memkind