Performance of Hybrid MPI/OpenMP VASP on Cray XC40 Based on Intel Knights Landing Many Integrated Core Architecture

Zhengji Zhao[1], Martijn Marsman[2], Florian Wende[3], and Jeongnim Kim[4]
1) National Energy Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory, Berkeley, USA. E-mail: zzhao@lbl.gov
2) Computational Materials Physics, University of Vienna, Vienna, Austria, E-mail: martijn.marsman@univie.ac.at
3) Zuse Institute Berlin (ZIB), Germany. E-mail: wende@zib.de
4) Intel, USA. E-mail: jeongnim.kim@intel.com

***Abstract* - With the recent installation of Cori, a Cray XC40 system with Intel Xeon Phi Knights Landing (KNL) many integrated core (MIC) architecture, NERSC is transitioning from the multi-core to the more energy-efficient many-core era. The developers of VASP, a widely used materials science code, have adopted MPI/OpenMP parallelism to better exploit the increased on-node parallelism, wider vector units, and the high bandwidth on-package memory (MCDRAM) of KNL. To achieve optimal performance, KNL specifics relevant for the build, boot and run time setup must be explored. In this paper, we will present the performance analysis of representative VASP workloads on Cori, focusing on the effects of the compilers, libraries, and boot/run time options such as the NUMA/MCDRAM modes, Hyper-Threading, huge pages, core specialization, and thread scaling. The paper is intended to serve as a KNL performance guide for VASP users, but it will also benefit other KNL users.**

Keywords – Hybrid MPI/OpenMP VASP, KNL, MCDRAM, Cache mode, Flat mode, Hyper-Threading, Huge pages, Performance

I. Introduction

With the recent installation of Cori [1], a Cray XC40 system based on the Intel Xeon Phi Knights Landing (KNL) many integrated core (MIC) architecture [2], NERSC is transitioning from the previous MPI-dominant multi-core era to a more energy-efficient many-core era, which favors the MPI/OpenMP programming model. Most application codes currently running at NERSC need to be ported, optimized, or re-implemented in order to run efficiently on this new architecture. The Vienna Ab initio Simulation Package (VASP) [3-4], a widely used materials science code that is highly ranked at NERSC (Fig. 1) and other supercomputing centers worldwide, has recently completed the transition from the former MPI-only to a hybrid MPI/OpenMP code base [6], and is currently under beta testing. The hybrid VASP code has been highly optimized for CPUs including KNL, achieving a 2-3 times speedup on KNL in comparison to the MPI-only VASP v5.4.1, the current production release (Fig. 2). Although the main KNL features, such as the increased on-node parallelism (68 cores and 272 hardware threads per node), wider vector units (512 bits and AVX-512 instruction sets), and the high bandwidth on-package memory (MCDRAM) have been addressed throughout the code optimization process, other KNL specifics relevant for the build, boot time options and runtime setup have yet to be explored to achieve optimal performance.

In this paper, we will present our performance evaluation of the hybrid MPI/OpenMP VASP code under the various execution configurations on Cori KNL system. We studied the effects of NUMA/MCDRAM modes, Hyper-Threading, huge pages, and core specialization, as well as compilers and libraries on the code performance. The tested compilers include Intel, GNU, and Cray compilers, and the tested libraries include the FFT and BLAS/LAPACK/ScaLAPACK libraries from Intel MKL [7], ELPA [8-9], Cray LibSci [10], and FFTW [11]. We have also studied the parallel/thread scaling of the hybrid MPI/OpenMP VASP, which is essential to run the hybrid code at optimal MPI task and OpenMP thread counts. To cover the representative workloads and to exercise different code paths in VASP, we have used multiple test cases with different problem sizes, constituent ionic types, and electronic structure calculation methods. Based on our performance

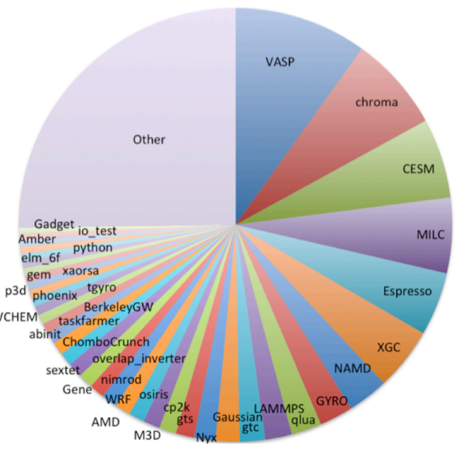


Figure 1. The computing cycles breakdown by application codes at NERSC in 2015. VASP used about 12% of the total computing cycles [5].

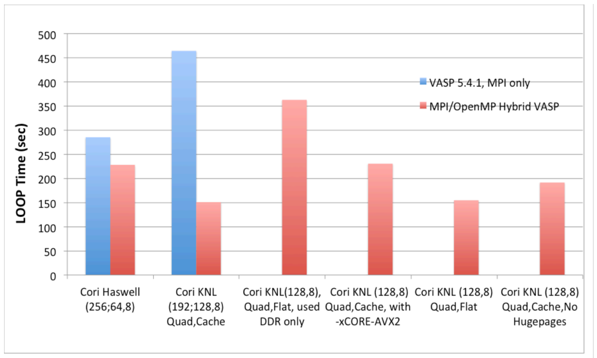


Figure 2. Performance comparison between the hybrid MPI/OpenMP (red bars) and the MPI-only VASP v5.4.1 code (blue bars) on Cori KNL and Haswell nodes. Where the numbers inside the parenthesis, ([num;] num, num), are the number of MPI tasks used for the MPI-only VASP (v5.4.1), if present; the MPI tasks and OpenMP threads per task used to run the hybrid VASP. The hybrid VASP outperforms the MPI-only code by 2-3 times (the second bar groups). The figure also shows the performance comparison between other build/boot/run time configurations on KNL. The hybrid VASP ran with 8 nodes and 8 threads per task.

evaluation and analysis, we will provide the best practice tips for VASP users. Additionally, the performance data presented in this paper provides feedback for computer vendors about the architecture's performance in a real world scientific application with a large user base.

The rest of the paper is organized as follows: we will describe the Cori configuration and environment in Section II, and will describe the VASP code and the test cases used in Section III. In Section IV, we will present the VASP performance results under the various combinations of build/boot/run time options, and discuss the observed performance results. We will conclude the paper by summarizing our observations in Section V.

II. System Configuration and Envrionments Setup

A. *Cori*

Cori is a Cray XC40 system that has been recently installed at NERSC. Cori has 9688 single-socket **Intel® Xeon Phi™ Processor 7250 ("Knights Landing")** nodes @1.4 GHz with 68 cores (272 hardware threads or CPUs) per node, 16 GB high bandwidth on-package memory (MCDRAM) (>460 GB/sec) and 96 GB DDR4 2400 MHz memory (102 GB/s) per node. Each core has two 512-bit wide vector units, and a 64 KB L1 cache. Two cores form a tile that share 1 MB L2 cache. In addition to the KNL nodes, Cori has 2388 dual-socket 16-core **Intel® Xeon™ Processor E5-2698 v3 ("Haswell")** nodes @2.3GHz with 32 cores (64 hardware threads or CPUs) per node, and 128 GB 2133 MHz DDR4 memory. Each core has a 256-bit vector unit, and has 64 KB and 256 KB L1 and L2 caches. There is also a 40 MB shared L3 cache per socket. Cori nodes are interconnected with Cray's Aries network with Dragonfly topology.

Compared to the previous generations of Intel Xeon processors (e.g, Haswell), KNL features a more flexible boot time and more runtime options. For example, the MCDRAM can be configured at the boot time as a third level cache (the cache mode), as a distinct NUMA node (the flat mode), or somewhere in between (the hybrid mode) (Fig. 3, upper panel). The advantage of configuring MCDRAM as a third level cache is that it does not require any changes in application source codes. However, the misses are expensive because applications need to access both the MCDRAM and the DDR memory. If the MCDRAM is configured in the flat mode, it is mapped to physical address space and exposed as a NUMA node (allocatable memory). Therefore it allows application developers and users to have full control over how to use the MCDRAM. The downside is that code modifications may be required in order to utilize the MCDRAM efficiently. In addition to MCDRAM, the KNL mesh can be configured to three different clustering modes: all-to-all, quadrant, and sub-NUMA clustering, which results in at least nine different combinations of NUMA/MCDRAM modes (Fig. 3 lower panel illustrates the quadrant cluster mode). Each mode has its pros and cons, and it is very challenging from a software perspective to understand the most suitable mode for an application. The NERSC staff has done extensive tests with the available

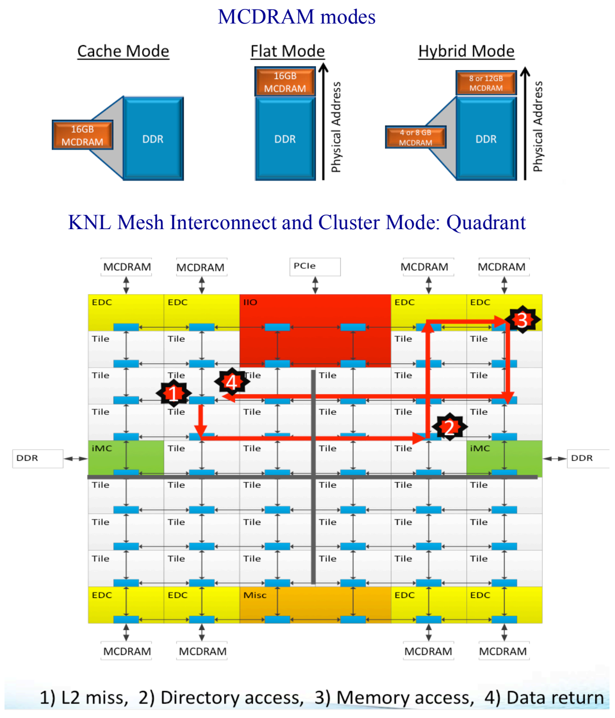


Figure 3. MCDRAM modes (upper panel), and the KNL mesh interconnect and the quadrant mode illustration.

NUMA/MCDRAM modes at the early testing stage of the Cori KNL system, and has recommended using quad,cache and quad,flat modes based on their ease of use and performance. Because of this, our test cases have run on the quad,cache and quad,flat modes only.

Cori runs Cray Linux Environment (CLE) 6.0 Update 3 [12] as its compute node OS, and SLURM 17.02 [13] as its batch system. All our tests were performed on Cori and its development and test system, Gerty, which is a smaller version of Cori.

III. CODE DESCRIPTION AND TEST CASES

A. VASP

The Vienna Ab-initio Simulation Package (VASP) [3-4] is a widely used materials science application for performing Ab-initio electronic structure calculations and quantum-mechanical molecular dynamics (MD) simulations using pseudopotentials or the projector-augmented wave method and a plane wave basis set. VASP computes an approximate solution to the many-body Schrödinger equation, either within the Density Functional Theory (DFT) to solve the Kohn-Sham equation or the Hartree-Fock (HF) approximation to solve the Roothaan equation. Hybrid functionals that mix the HF approach with DFT have been implemented, and Green's functions methods (GW quasi-particles and ACFDT-RPA) and many-body perturbation theory (2nd-order Møller-Plesset) have also been implemented in VASP.

The fundamental mathematical problem that VASP solves is a non-linear eigenvalue problem (Eq. (1)) that is solved iteratively via self-consistent iteration cycles until a desired accuracy is achieved.
While ε_n and Ψ_n are an eigenvalue (energy) and eigenfunction (wavefunction or band) pair, N is the number of wavefunctions to solve (N is roughly the same as the number of electrons in the system). These N wavefunctions must be orthonormal to each other (Eq. (2)). VASP stores the wavefunctions as coefficients of the Fourier components (or plane-wave basis sets) as shown in Eq. (3). VASP makes use of efficient iterative matrix diagonalization techniques such as the residual minimization method with direct inversion of the iterative subspace (RMM-DIIS) and the blocked Davidson (BD) algorithms. It heavily depends on FFT and Linear Algebra libraries (BLAS/LAPACK/ScaLAPACK).

The majority of the VASP code is written in Fortran90. So far, all released codes including the most recent release, VASP v5.4.4, are MPI-only codes. The MPI-only VASP is parallelized in two levels (the top level k-point parallel is not counted here). At the high level, the bands are distributed over MPI tasks in a round-robin fashion; at the lower level, the Fourier components of each band are distributed across multiple MPI tasks. In general, VASP execution is dominated by the back and forth FFTs (from/to the real space to/from the Fourier space or **G** space), and zgemm/dgemm calls. The MPI time is dominated by the MPI_Allreduce calls. More details about parallel implementation of VASP can be found in [6].

B. Hybrid MPI/OpenMP VASP and optimizations for KNL

In the hybrid MPI/OpenMP VASP the bands are distributed over MPI tasks (high level) as they are in the MPI-only code, and the Fourier components of the bands are distributed over OpenMP threads, either explicitly via adding OpenMP directives in the LOOP level, or implicitly via the use of the threaded libraries, such as FFTW and LAPACK/BLAS3 libraries. So far (up to 4/13/2017), there has been no nested OpenMP usage in the hybrid VASP code. More details about the OpenMP implementation in the hybrid VASP can be found in [6].

Another major optimization in the hybrid VASP code is SIMD (Single Instruction, Multiple Data) vectorization to promote the execution of parts of the program on KNL's 512 bit wide vector units. To stay within the high-level Fortran language and to keep the code portable, we have fully drawn upon OpenMP 4.x SIMD constructs in the user code using both explicit loop-level vectorization via *!$omp simd*, and sub-routine/function vectorization through *!$omp declare simd*. Optimization targets comprise, beside simple loops, the vectorization of nested function calls ("deep" calling hierarchies) and complex loop structures mixing scalar and vector codes. For that purpose, we partly use a combination of user-defined high-level vectors together with OpenMP SIMD loop vectorization to process these vectors. The latter must be packed/unpacked manually, but can be easily moved in

$$H[\{\psi\}]\psi_n(\boldsymbol{r}) = \varepsilon_n\Psi_n(\boldsymbol{r}), \quad n = 1, \dots, N \tag{1}$$

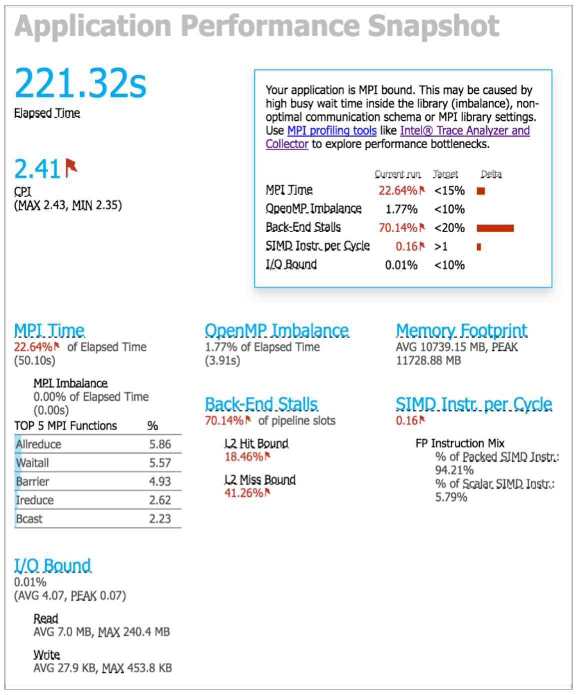
$$\int \Psi_n^*(\boldsymbol{r})\Psi_m(\boldsymbol{r})d\boldsymbol{r} = \delta_{nm} \tag{2}$$

$$\psi_n(\boldsymbol{r}) = FFT\{\Psi_n(\boldsymbol{G})\}(\boldsymbol{r}) \tag{3}$$

context of the SIMD function context and in combination with scalar code via loop splitting.

The Intel Application Performance Snapshot [14] reports more than 94% of packed SIMD instructions for the code execution (Fig. 4). More details about SIMD vectorization and other optimization efforts in the hybrid VASP, such as the SIMD optimization portability among compilers, can be found in [6].

The explicit use of MCDRAM via Intel compiler's FASTMEM directive and the Memkind libraries [15] was also explored in the hybrid VASP [16]. Although the expected performance boost from the targeted MCDRAM use was observed with the emulated MCDRAM using dual-socket Intel Ivy Bridge nodes, the code unfortunately slowed down significantly on KNL due to the conversion



Application Performance Snapshot

221.32s
Elapsed Time

2.41
CPI
(MAX 2.43, MIN 2.35)

Your application is MPI bound. This may be caused by high busy wait time inside the library (imbalance), non-optimal communication schema or MPI library settings. Use MPI profiling tools like Intel® Trace Analyzer and Collector to explore performance bottlenecks.

	Current run	Target	Delta
MPI Time	22.64%	<15%	▮
OpenMP Imbalance	1.77%	<10%	
Back-End Stalls	70.14%	<20%	▬
SIMD Instr. per Cycle	0.16	>1	▮
I/O Bound	0.01%	<10%	

MPI Time
22.64% of Elapsed Time
(50.10s)

MPI Imbalance
0.00% of Elapsed Time
(0.00s)

TOP 5 MPI Functions	%
Allreduce	5.86
Waitall	5.57
Barrier	4.93
Ireduce	2.62
Bcast	2.23

OpenMP Imbalance
1.77% of Elapsed Time
(3.91s)

Memory Footprint
AVG 10739.15 MB, PEAK 11728.88 MB

Back-End Stalls
70.14% of pipeline slots

L2 Hit Bound
18.46%

L2 Miss Bound
41.26%

SIMD Instr. per Cycle
0.16

FP Instruction Mix
% of Packed SIMD Instr.: 94.21%
% of Scalar SIMD Instr.: 5.79%

I/O Bound
0.01%
(AVG 4.07, PEAK 0.07)

Read
AVG 7.0 MB, MAX 240.4 MB

Write
AVG 27.9 KB, MAX 453.8 KB

Figure 4. Intel APS report for the HSE functional calculation of VASP on Cori KNL. This is a run on Si256_hse test case with 8 nodes, 128 processors with 8 threads/task.

of some stack array variables to heap variables in order to use MCDRAM. Because of this, the FASTMEM directives were not adopted in the hybrid VASP. VASP uses MCDRAM as a cache or a flat memory via the numactl (or using srun –mem_bind) in the preferred mode. Intel's AutoHBW tool [17] can be used with VASP if the explicit management of array allocations (by array sizes only) is preferred.

C. *Test cases*

In this study, we used six different test cases, as shown in Table 1. The first test case, denoted as PdO4 in Table 1, is a PdO slab containing 348 atoms in total (300 Pd atoms and 48 O atoms). This system was chosen to test the most commonly used (and basic) code path, the DFT functional calculation with PBE potentials using the RMM-DIIS iteration scheme. Since problem size matters in benchmarking, we added a smaller PdO slab (denoted as PdO2 in Table 1), which has 174 atoms (150 Pd and 24 O atoms) in the tests. A ternary alloy structure, GaAsBi-64, was included to cover the metalic systems with the default iteration scheme, Block Davidson + RMM-DIIS algorithms. Two HSE hybrid functional calculations with different atomic configurations and problem sizes were included in the tests, because hybrid functional calculations are increasingly important workloads in VASP. Si256_hse is a 256-atom silicon supershell with a vacancy, and B.hR105_hse is a small hexa-boron structure (105 atoms). The Van Der Waals functional calculation was also included (denoted as CuO_vdw in Table 1).

We disabled heavy I/O (LWAVE = .FALSE.) in the tests. All runs started from a given WAVECAR file so that different runs for the same benchmark always start with an exact same initial wavefunction. Each test was run multiple times (>3 times) and reported the best LOOP+ time, which will be the dominant portion in the execution time for the production runs.

TABLE 1. Test cases used in the performance tests. These cases were chosen to cover the representative VASP workloads and to exercise different code paths.

	PdO4	GaAsBi -64	CuC_vdw	Si256_hse	B.hR105_hse	PdO2
Electrons (Ions)	3288 (348)	266 (64)	1064 (98)	1020 (255)	315 (105)	1644(174)
Functional	DFT	DFT	VDW	HSE	HSE	DFT
Algo	RMM (VeryFast)	BD+RMM (Fast)	RMM (VeryFast)	CG (Damped)	CG (Damped)	RMM (VeryFast)
NEML(NELMDL)	5 (3)	8 (0)	10 (5)	3(0)	10 (5)	10 (4)
NBANDS	2048	192	640	640	256	1024
FFT grids	80x120x54 160x240x108	70x70x70 140x140x140	70x70x210 120x120x350	80x80x80 160x160x160	48x48x48 96x96x96	80x60x54 160x120x108
NPLWV	518400	343000	1029000	512000	110592	259200
IRMAX	1445	4177	3797	1579	1847	1445
IRDMAX	3515	17249	50841	4998	2358	3515
LMDIM	18	18	18	18	8	18
KPOINTS	1 1 1	4 4 4	3 3 1	1 1 1	1 1 1	1 1 1

D. Cori software and runtime specifications

In this study, we have used the hybrid MPI/OpenMP code (last commit date: 4/13/2017). Wannier90 v1.2 was enabled in the VASP builds. The majority of the tests used the binaries built with the Intel compiler (2017 Update1) and linked to the MKL (2017 Update1) and ELPA (2016.05.004) libraries, except where noted. The FFT routines were also from MKL via the FFTW3 wrapper interfaces. The binaries for the compiler/library performance tests used GCC 6.3.0 and CCE 8.5.8 and were linked to the LibSci 16.11.1, FFTW 3.3.4.11, and the MKL 2017 Update1. All binaries are linked to Cray MPICH 7.5.3 as well.

A proper process/thread/memory affinity is the basis for optimal performance. On Cori, we used the –cpu_bind and –c options of the srun (SLURM's parallel job launcher) to pin MPI tasks to a subset of CPUs and used OpenMP environment variables like OMP_PROC_BIND and OMP_PLACES to control the thread affinity. When running in the flat mode, the "numactl –m 1" was used to enforce the use of the MCDRAM. A representative job script was as follows:

```
#!/bin/bash -l

#SBATCH -p debug
#SBATCH -C knl,quad,cache   # to request cache mode
#SBATCH -N 8
#SBATCH -t 30:00

module load craype-hugepages2M
export MKL_FAST_MEMORY_LIMIT=0

export OMP_PROC_BIND=true
export OMP_PLACES=threads
export OMP_STACKSIZE=512m
export OMP_NUM_THREADS=8

# to run with 1 hardware thread per core
srun -n 64 -c 32 --cpu_bind= cores ./vasp_std
```

The resulting thread affinity of this script is the same as that of KMP_AFFINITY="granularity=thread,scatter" of Intel compilers, and generates the same or compatible thread affinity for applications built with Cray and GNU compilers on KNL.

IV. Performance Results and Discussion

A. Thread scaling of hybrid VASP

Understanding the parallel/thread scaling of applications is critical to run the codes efficiently on HPC systems. Running a code outside the parallel scaling region either slows down scientific productivity or wastes valuable computing resources. It may even cause various unexpected runtime errors and issues. While the MPI-only code is known to scale up to one core per atom (the non-collinear VASP scales further out), the hybrid MPI/OpenMP code is still new to the VASP community. We have studied the parallel/thread scaling of the hybrid VASP on the Cori KNL and Haswell nodes (Fig. 5) using the six selected benchmarks that were carefully chosen to cover representative workloads and to exercise commonly used code paths. Note that these selected benchmarks are for the day-to-day scientific runs (not designed for heroic runs). Among the six selected benchmarks, the first three cases, PdO4, Si256_hse, and CuC_vdw, were relatively larger systems, so we ran tests with up to 16 nodes, at which an MPI-only code could run with up to 1024 MPI ranks (or 4096 with four hardware threads per core) on KNL. This would be already beyond the parallel scaling regions of the MPI-only VASP for these test systems (see Table 1 for the atomic configurations, the number of bands and plane-waves, and the real/Fourier space grids for each case). The rest of the test cases ran with up to eight nodes. At all node counts, 1, 2, 4, 8, and 16 OpenMP threads per MPI task were tested. The number of MPI tasks used for the node and thread counts are shown in Table 2 for both the KNL and Haswell runs.

Fig. 5 shows that the hybrid VASP performs better with four or eight threads per MPI task at all node counts for all benchmarks. For the runs with smaller node counts (one or two nodes), using four threads per task performs best. However, when the node count increases, using eight threads per task outperforms four threads per task for all benchmarks except PdO2, the smaller DFT functional calculation (for which VASP performs best with four threads). VASP scales up to 8 nodes for all the benchmarks used in the tests. For the Si256_hse and CuC_vdw test cases (two larger cases), the hybrid VASP scales further out to 16 nodes and 16 threads per task. We ran the Si256_hse case using 32 nodes with 8 16, and 32 threads per task and compared it to the 16 node results (Fig. 6). One can see that the hybrid VASP further scales to 32 nodes with all 8, 16 and 32 threads per task. Craypat [18], a Cray profiling tool, reports 28% of MPI time for the runs using 32 nodes and 32 threads per task, which is still within the acceptable MPI time for a production run. These results show that VASP scales to more nodes and threads for large problems. Being able to use more threads per task and to scale up to larger number of nodes makes it possible for the hybrid VASP to solve larger and complex problems faster. In practice, we would recommend using eight threads per task (except small cases) and 16 threads or more for larger systems.

As described in Section III, the MPI parallelism over the Fourier components (lower level MPI parallelism) has been replaced by the OpenMP parallelism (threads), meaning the hybrid VASP no longer uses the NCORE (or NPAR), which was used to specify how many bands to solve simultaneously in the MPI-only VASP. Since the MPI parallelism is over the bands only in the hybrid code, it imposes an upper limit on how many MPI tasks the code can make use of efficiently. In practice, when deciding how many MPI tasks to use for a given problem, a reference number would be 1/8 - 1/4 of the number of atoms in the system (for a single k-point calculation), although the sweet spot of the parallel scaling is problem-dependent. To get optimal performance the calculations need to be addressed case by case.

Fig. 5 also shows the comparison between KNL (blue bars) and Haswell (yellow bars). One can see that at smaller node counts, the hybrid VASP performs better on KNL, but eventually, at larger node counts, its

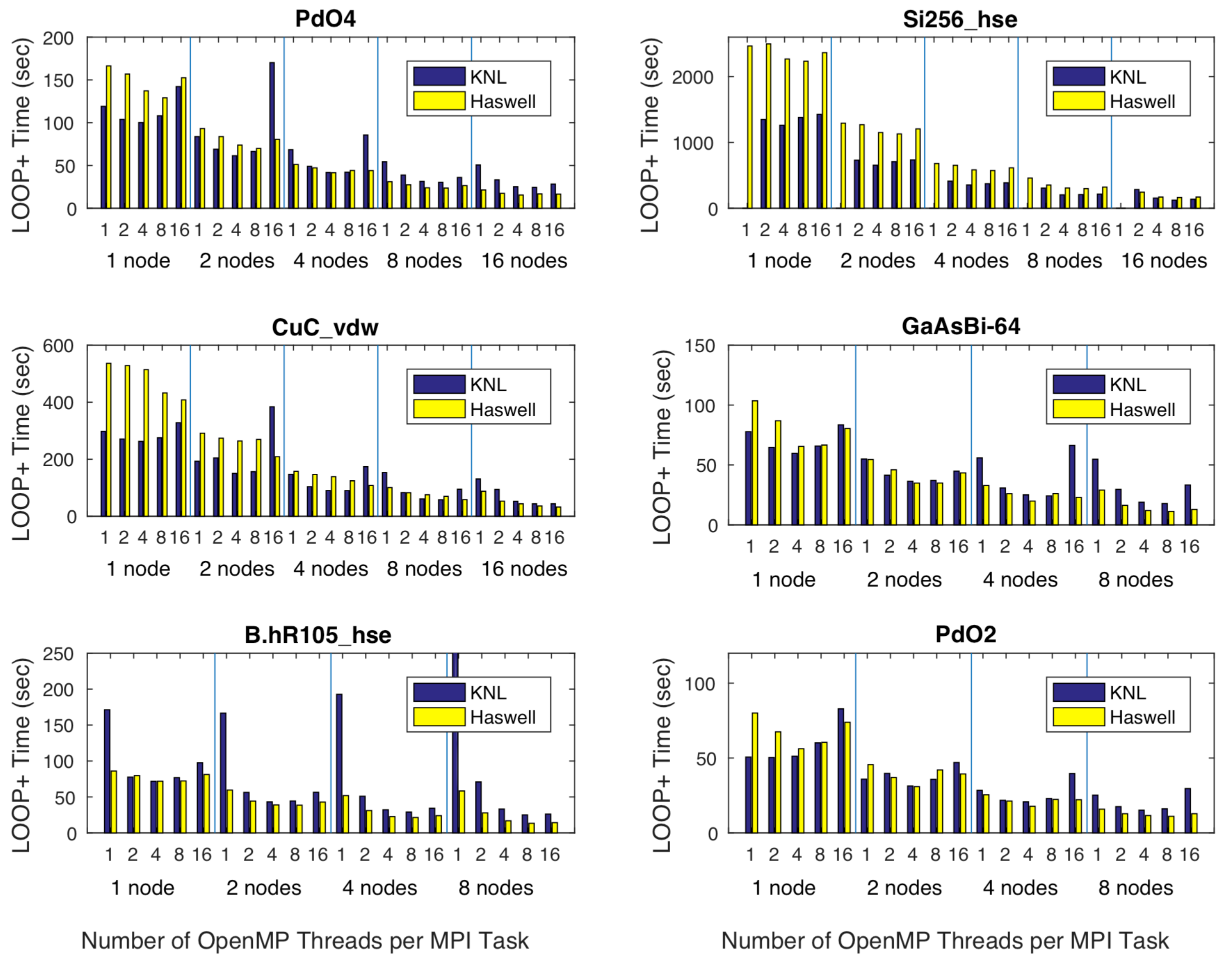
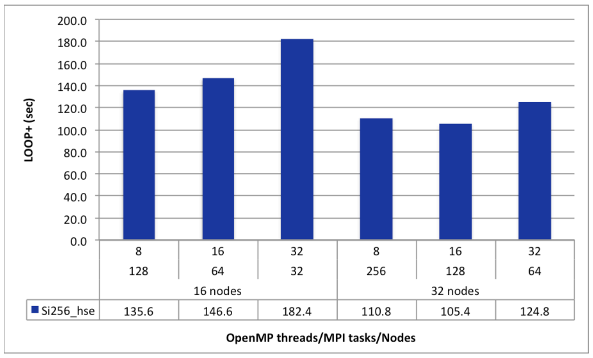


Figure 5. The parallel/thread scaling of the hybrid MPI/OpenMP VASP (version 4/13/2017) on the Cori KNL and Haswell nodes. The horizontal axis shows the number of OpenMP threads per task and the number of nodes used, and the vertical axis shows the LOOP+ time (the dominant portion in the execution time). All runs used one hardware thread per core, and 64 cores per node. The 2M huge page memory was used. All the KNL runs were under the quad,cache mode. Note, the spikes at the thread count 1 and 16 are reproducible results. It is not clear yet what was the root cause. Need further investigation to understand the cause. The Si256_hse tests ran out of the 2M huge page memory when using one OpenMP thread per MPI task. Note that the hybrid VASP does not use NCORE any more (or NCORE=1), however for the runs with one OpenMP thread per task, i.e., running the hybrid VASP as an MPI-only code, $NCORE \neq 1$ is still supported. We used NCORE=4 for all the runs with one OpenMP thread per task for fair comparison.

TABLE 2. The MPI tasks used for each node and thread combination shown in Fig. 1. All runs used one hardware thread per core. For the KNL runs, 64 cores out of 68 available were used.

No. of Nodes	1					2					4					8					16				
No. of Threads	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
MPI Tasks (KNL)	64	32	16	8	4	128	64	32	16	8	256	128	64	32	16	512	256	128	64	32	1024	512	256	128	64
MPI Tasks (Haswell)	32	16	8	4	2	64	32	16	8	4	128	64	32	16	8	256	128	64	32	16	512	256	128	64	32



	16 nodes			32 nodes		
	8 / 128	16 / 64	32 / 32	8 / 256	16 / 128	32 / 64
■ Si256_hse	135.6	146.6	182.4	110.8	105.4	124.8

Figure 6. The hybrid VASP performance with 16 and 32 Cori KNL nodes. The 8, 16, and 32 threads per task were tested. The runs used the Craypat instrumented code.

performance on Haswell catches up and shows a better parallel scaling trend then KNL.

Note that those spikes at thread counts 1 and 16 are reproducible results, and are not yet well understood. Further investigation is needed to understand its root cause.

B. *Hyper-Threading (HT)*

Each Cori KNL core has four hardware threads (CPUs). HT could improve the application performance through increasing the resource utilization by simultaneously running multiple process/threads on the hardware threads on the core, making use of the cycles that would otherwise be wasted due to cache misses, branch mis-predictions, data dependencies, and/or waiting for other resources in a single process/thread execution on the core. A previous study on the effects of HT on the performance of MPI or MPI/OpenMP applications [19] on a Cray XC30 system based on Intel Sandy Bridge

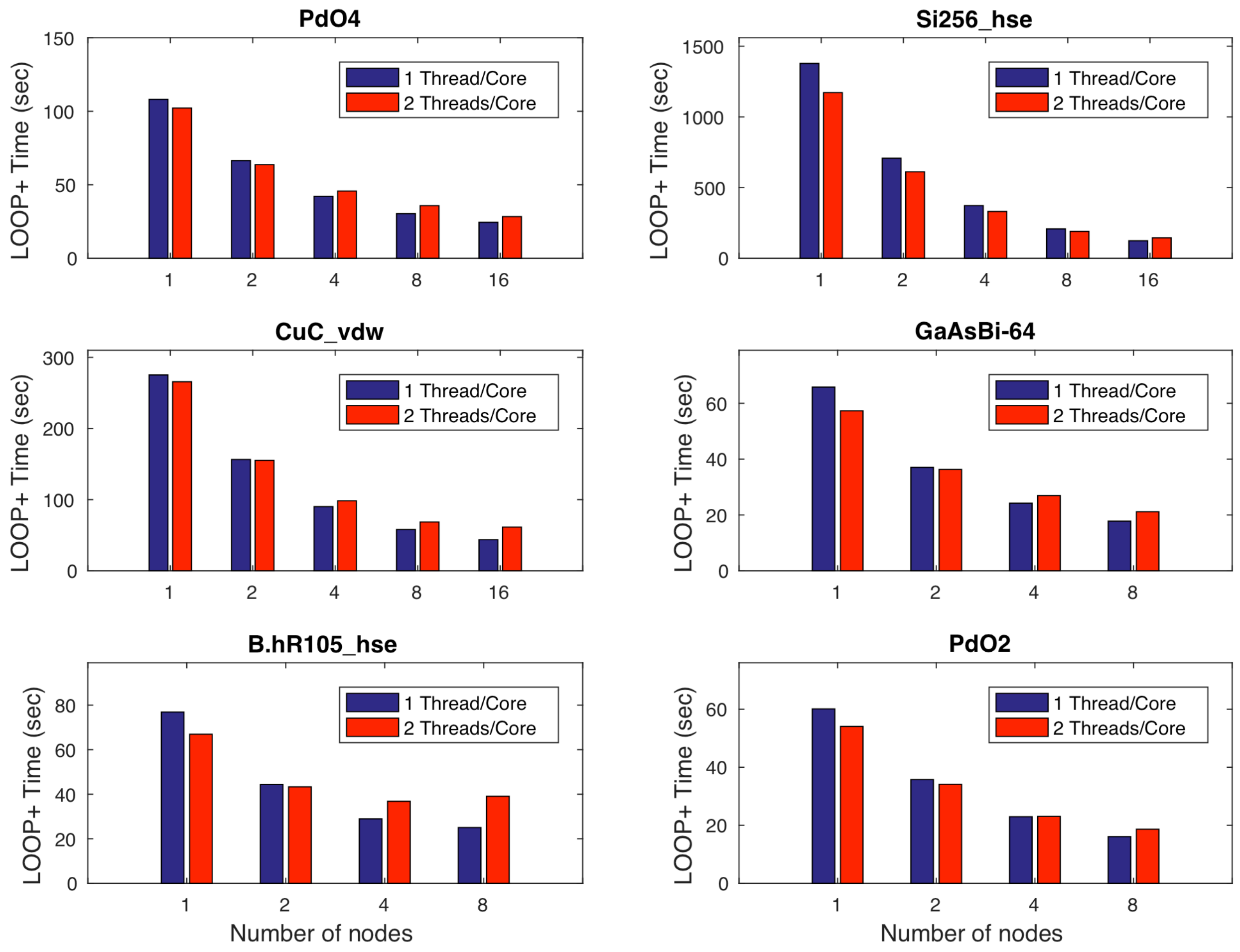


Figure 7. The performance effects of Hyper-Threading to the hybrid MPI/OpenMP VASP on the Cori KNL nodes. The horizontal axis shows the number of nodes, and the vertical axis shows the LOOP+ time (the dominant portion in the execution time). All runs were under the quad,cache mode with one hardware thread per core. All runs used 64 cores out of 68 available on the KNL node. The 2M huge page memory was used.

processor nodes showed that its effect is highly application-dependent. In general, HT may increase the performance if an application is efficiently parallelized and runs at smaller node counts. However, the MPI-only VASP did not get any performance benefit from using HT, and the code was instead slowed down by more than 50% on Sandy Bridge nodes (two hardware threads per core, 16 cores per node). Now with the KNL system, which provides more hardware threads per core (4), it is worth exploring whether or not HT helps VASP performance. Fig. 7 shows the comparison between using one (64 CPUs per node were used) and two (128 CPUs per node were used) hardware threads per core. Fig. 7 shows that HT does not help hybrid VASP for most of the workloads, except the HSE workloads. We have also run tests with four hardware threads per core, but it greatly worsened the code performance (data not shown).

C. NUMA/MCDRAM modes

Fig. 8 shows the comparison between using the quad,cache and quad,flat modes. We choose to run the six benchmarks at the selected MPI/Thread counts that would be used by production runs. There is no obvious performance difference between using the flat and the cash modes in all six benchmarks. Note that all these runs were using a memory much lower than 16 GB and can therefore fit within the MCDRAM when running in the quad,flat mode. In practice, it is a good strategy to use more nodes and threads in order to reduce the memory footprint on the node so that the run can fit into the MCDRAM memory (for the flat mode and also for the cache mode). As one can see from Fig. 2 in the introduction, the biggest performance boost was from using MCDRAM (for HSE workloads), so it is important to make use of MCDRAM whenever possible.

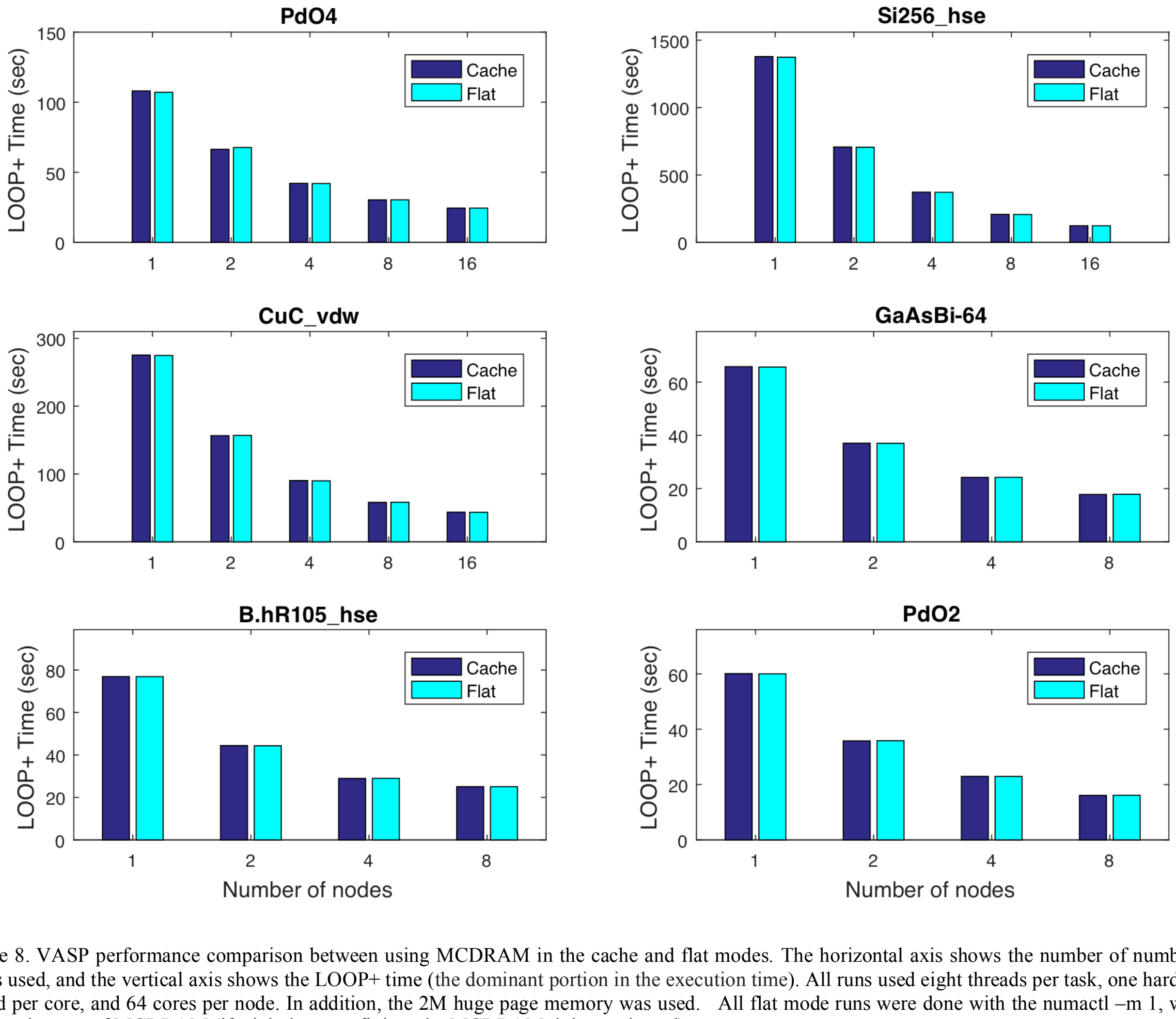


Figure 8. VASP performance comparison between using MCDRAM in the cache and flat modes. The horizontal axis shows the number of number of nodes used, and the vertical axis shows the LOOP+ time (the dominant portion in the execution time). All runs used eight threads per task, one hardware thread per core, and 64 cores per node. In addition, the 2M huge page memory was used.   All flat mode runs were done with the numactl –m 1, which enforce the user of MCDRAM (if a job does not fit into the MCDRAM, it is terminated).

D. Huge pages

Huge pages are virtual memory pages that are bigger than the 4Kbyte default base page size. Huge pages can improve memory performance for common access patterns on large data sets. Huge pages also increase the maximum size of data and text in a program accessible by the high speed network [20]. However, using huge pages has not been widely adopted among NERSC users because using the huge pages increases job failure rates presumably due to the decreased amount of available huge page memory on the node from memory fragmentation when the system has been running for some time. In addition, huge page effects are application-dependent, and not all applications can get a benefit from using huge pages. Given the fact that KNL nodes are frequently rebooted by user jobs, huge page memory may benefit user applications more in comparison to the previous generation of Intel Xeon process nodes. As shown in Fig. 9 the huge page memory helped VASP performance in all our test cases.

E. Compilers and libraries

Fig. 10 shows the comparison between different compilers and libraries. As one can see using Intel compiler + MKL outperforms all other alternatives. The huge performance gap between using MKL and LibSci + FFTW may indicate some issues in the LibSci and/or FFTW libraries. Among the tested combinations within Intel compilers, the Intel + MKL + ELPA provides the best performance. We recommend the use of Intel compilers + MPL + ELPA libraries for the hybrid VASP until the performance gap between MKL and LibSci+FFTW is removed. There was a problem with the GNU compiler (GCC v6.3.3) build when linked to the GNU OpenMP runtime library, libgomp.a (too many thread error). Because of this, we linked the hybrid VASP to the Intel OpenMP runtime to be able to run the VASP binary built with GNU compilers.

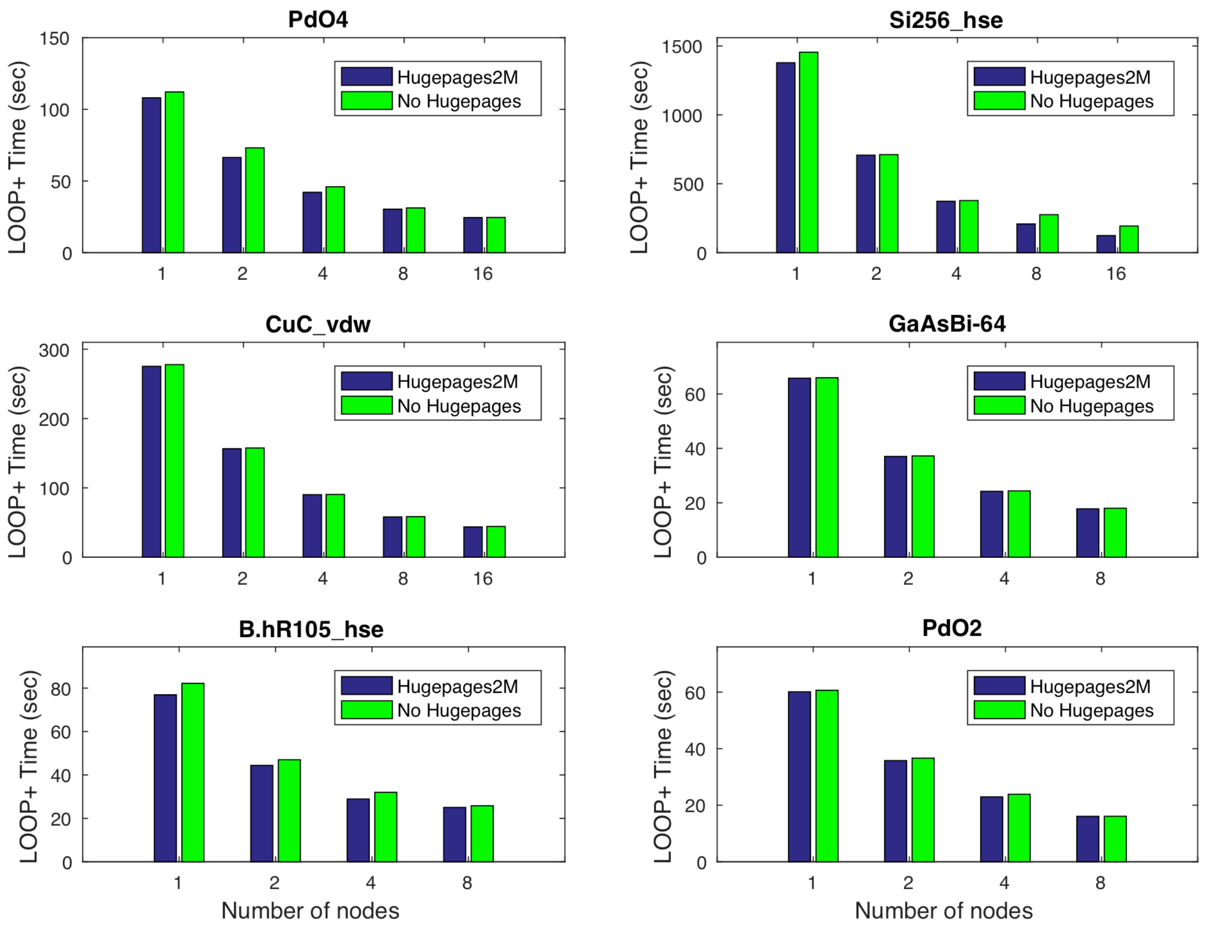


Figure 9. Effects of the huge page memory to the hybrid VASP performance (version 4/13/2017) on the Cori KNL nodes. The horizontal axis shows the number of nodes used, and the vertical axis shows the LOOP+ time (the dominant portion in the execution time). All runs were under the quad,cache mode, and used one hardware thread per core, 64 cores per node.

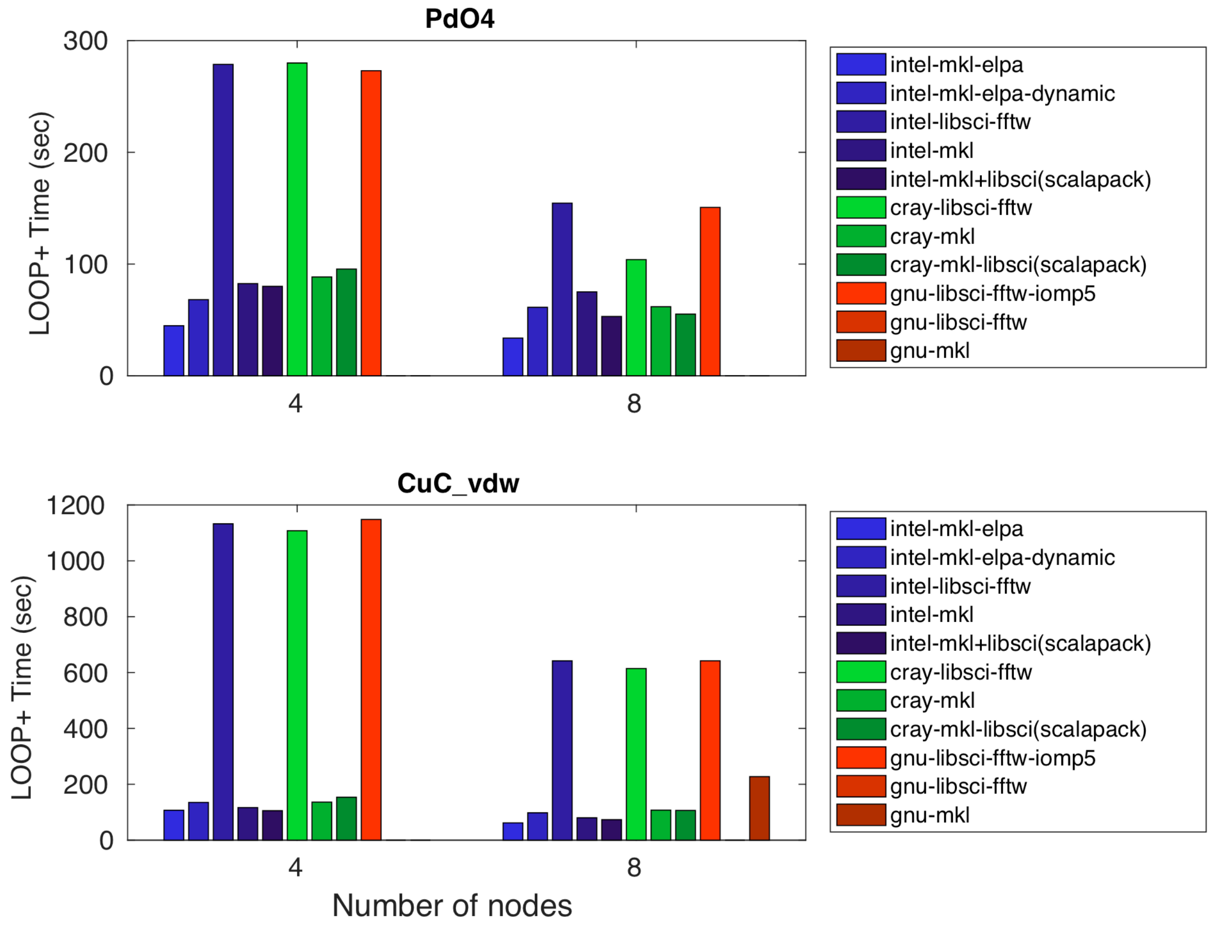


Figure 10. Performance comparison between different compilers and libraries. The horizontal axis shows the number of nodes used, and the vertical axis shows the LOOP+ time (the dominant portion in the execution time). The "intel", "cray", and "gnu" in the legend strings denote Intel, Cray and GNU compilers, and the "mkl", "elpa", "libsci", and "fftw" denote the MKL, ELPA, Cray LibSci, and the FFTW libraries. The "intel-mkl-elpa" denotes the VASP binary built with an Intel compiler and linked to the MKL and ELPA libraries. The "dynamic" means a dynamically linked binary. The "intel-mkl-libci (scalapack)" means the VASP binary built with an Intel compiler and linked to the MKL for its LAPACK/BLAS/FFT libraries, and the Cray LibSci for its ScaLAPACK libraries. The "gnu-libsci-fftw-iomp5" denotes a binary built with a GNU compiler and linked to the LibSci, FFTW, and the Intel OpenMP runtime library (libiomp5.a) instead of the GNU OpenMP runtime (libgomp.a). Note that some of the binaries failed to run (e.g., two of the GNU builds). In addition, Cray builds failed to run with the HSE test cases.

F. Core specializations

We have evaluated the effect of using core specializations on Cori KNL. We have tested with 1,2,4, and 0 cores specialized for the system use (via SLURM's sbatch option #SBATCH –S 1), but did not see any obvious performance gain from using the core specializations. Occasionally, we saw a slightly better performance with one core specialization, but using more cores slightly slows down the code. It should be noted that in our tests there was no heavy I/O involved, which may have some effect on the core specialization tests.

V. CONCLUSIONS

We have studied the parallel/thread scaling of the hybrid MPI/OpenMP VASP code with representative workloads on the Cori KNL system, and have tested its performance impact from a few build/boot/run time options. Our study shows that the hybrid code performs best at four or eight threads per MPI task. Using eight threads per task in production runs is generally recommended while for small cases four threads per task may perform better. It should be noted that in our tests the number of bands/plane-waves were selected such that the load on each MPI task/thread are well balanced in most of the tests. A reference number when choosing the number of MPI tasks to use for a given system is 1/8 - 1/4 of the atoms in the system (assuming using eight threads/tasks) for a single k-point calculation. Intel compilers + MKL (and FFTW wrappers to MKL) deliver the best performance among other compiler and library combinations - e.g., an Intel, Cray or GNU compiler + LibSci and FFTW. Huge pages either help or do not hinder VASP performance in almost all cases, so their use is recommended. For the workloads that fit into the MCDRAM, the cache and flat modes perform similarly, although we would recommend the cache mode for the hybrid VASP for ease of use. The hybrid VASP (HSE workloads) gets the most performance benefit from using MCDRAM, so it could be beneficial to use more nodes and threads (eight threads) to reduce the memory requirement per node, thus fitting the workloads to MCDRAM wherever possible. In general, using one hardware thread per core is recommended. However, hyper-threading (using two hardware threads per core) could help VASP performance with the HSE workloads, especially when running at smaller node counts. Using 64 cores out of 68 available is recommended.

For future work, we are planning to further analyze the profiling data that we have recently collected, as well as the performance bottlenecks reported by APS and Craypat. We will also address the compiler portability issue - e.g., the hybrid VASP did not run with the HSE test cases if built with a Cray compiler. We will also explore the MPI turning options, such as the asynchronous progress engine that is available in Cray MPICH. Lastly, we will investigate the reproducible runtime spikes observed at thread counts 16 and 1 with some of the tests.

ACKNOWLEDGEMENT

The authors would like to thank Heidi Paxon at Cray for resolving multiple Craypat issues allowing us to collect profiling data for the hybrid VASP with Craypat. The authors would also like to thank Dmitry Prohorov at Intel for his timely help and support with APS. This work is partially supported by Intel Corp. within the “Research Center for Many-Core High-Performance Computing” (IPCC) at ZIB, by NERSC’s Exascale Science Applications Program (NESAP) [21], and by the ASCAR Office in the DOE Office of Science, under the contract number DE-AC02-05CH11231. The computational resources are from the National Energy Scientific Computing Center (NERSC).

REFERENCES

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

[2] Sodani, A., Gramunt, R., Corbal, J., Kim, H.S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.C.: Knights landing: Second-generation intel xeon phi product. IEEE Micro 36(2) (2016) 34–46

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

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

[5] http://portal.nersc.gov/project/mpccc/baustin/NERSC_2014_Workload_Analysis_30Oct2015.pdf

[6] Florian Wende, Martijn Marsman, Zhengji Zhao and Jeongnim Kim, *“Porting VASP from MPI to MPI + OpenMP [SIMD] - Optimization Strategies, Insights and Feature Proposals”*, submitted to IWOMP 2017.

[7] Intel® Math Kernel Library, http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation

[8] Andreas Marek, Volker Blum, Rainer Johanni, Ville Havu, Bruno Lang, Thomas Auckenthaler, Alexander Heinecke, Hans-Joachim Bungartz, and Hermann Lederer, *“The ELPA Library - Scalable Parallel Eigenvalue Solutions for Electronic Structure Theory and Computational Science”,* The Journal of Physics: Condensed Matter 26, 213201 (2014).

[9] Eigenvalue Solvers for Petaflop-Applications (ELPA), https://elpa.mpcdf.mpg.de/about

[10] Cray Scientific and Math Libraries (CSML, also known as LibSci), http://pubs.cray.com/content/S-2529/16.10/xctm-series-programming-environment-user-guide-1705-s-2529/cray-scientific-and-math-libraries-csml

[11] FFTW, http://www.fftw.org/

[12] Cray Linux Environment (CLE) 6.0 update 3, http://docs.cray.com/PDF/XC_Series_System_Administration_Guide_CLE60UP03_S-2393.pdf

[13] Morris A. Jette , Andy B. Yoo , and Mark Grondona, *SLURM: Simple Linux Utility for Resource Management (2002),* In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003; https://slurm.schedmd.com/

[14] Intel Application Performance Snapshots (APS), https://software.intel.com/sites/products/snapshots/application-snapshot

[15] Christopher Cantalupo, Vishwanath Venkatesan, Jeff R. Hammond, Krzysztof Czurylo and Simon Hammond, *User Extensible Heap Manager for Heterogeneous Memory Platforms and Mixed Memory Policies,* http://memkind.github.io/memkind/memkind_arch_20150318.pdf

[16] Zhengji Zhoa and Martijn Marsman, *“Estimating the Performance Impact of the MCDRAM on KNL Using Dual-Socket Ivy Bridge Nodes on Cray XC30”,* Cray User Group meeting, May 8-12, 2016, England, UK.

[17] http://memkind.github.io/memkind/examples/autohbw_README

[18] Craypat, http://docs.cray.com/books/S-2376-63/S-2376-63.pdf

[19] Zhengji Zhao, Nicholas J. Wright and Katie Antypas, *“Effects of Hyper-Threading on the NERSC workload on Edison”*, Cray User Group meeting, May 6-9, 2013, Napa Valley, CA.

[20] Huge pages man page, “man intro_hugepages” on Cray XC40.

[21] http://www.nersc.gov/nesap