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

Zhengji Zhao[1] (zzhao@lbl.gov), **Martijn Marsman[2]** (martijn.marsman@univie.ac.at), **Florian Wende[3]** (wende@zib.de), **and Jeongnim Kim[4]** (jeongnim.kim@intel.com)

[1)] National Energy Research Scientific Computing Center (ERSC), Lawrence Berkeley National Laboratory, Berkeley, USA; [2)] University of Vienna, Vienna, Austria;[3)] Zuse Institute Berlin (ZIB), Germany; [4)] Intel, USA

Cray User Group Meeting, May 11, 2017, Redmond, WA

Acknowledgement

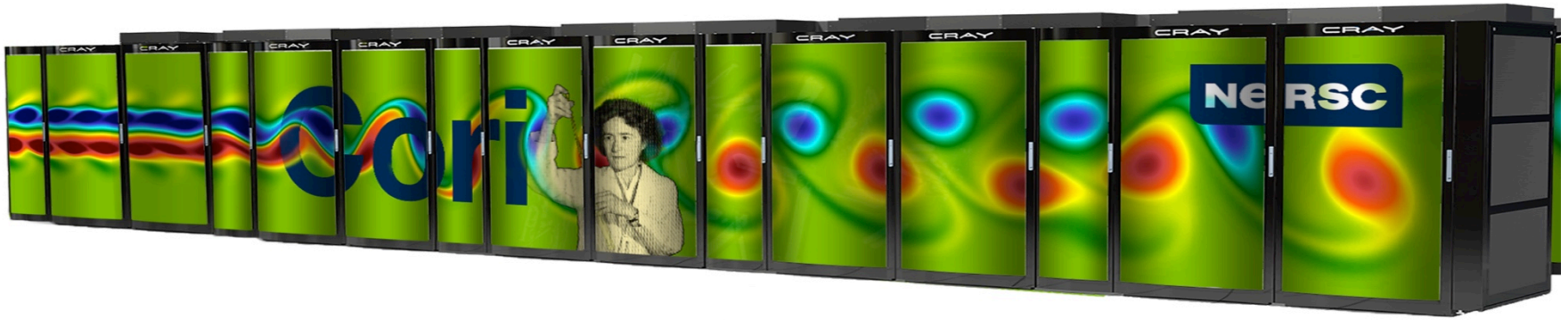
- Intel Corp. within the “Research Center for Many-core High-Performance Computing” (IPCC) at ZIB.
- NERSC Exascale Science Applications Program (NESAP).

Outline

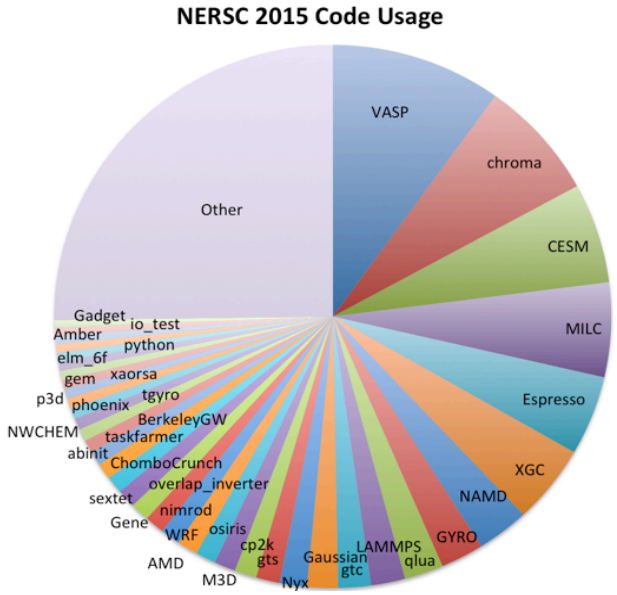
- Motivation
- MPI+OpenMP Hybrid VASP – Optimizations for KNL
- Benchmarks and Experiment Setups
- Performance and Analysis
  - Thread scaling, NUMA,MCDRAM modes, hugepages, Hyper-Threading, compilers and libraries
- Summary and Future work

Background

- With the recent installation of Cori KNL system, NERSC is transitioning from the multi-core to the more energy-efficient many-core era.
- Most of the applications at NERSC must be ported, optimized, or re-implemented to run efficiently on this new architecture.
- Code optimizations need to address increased parallelisms on the node, larger vector units, high bandwidth on chip memory.



VASP has recently completed the transition from an MPI-only to an MPI/OpenMP hybrid code base

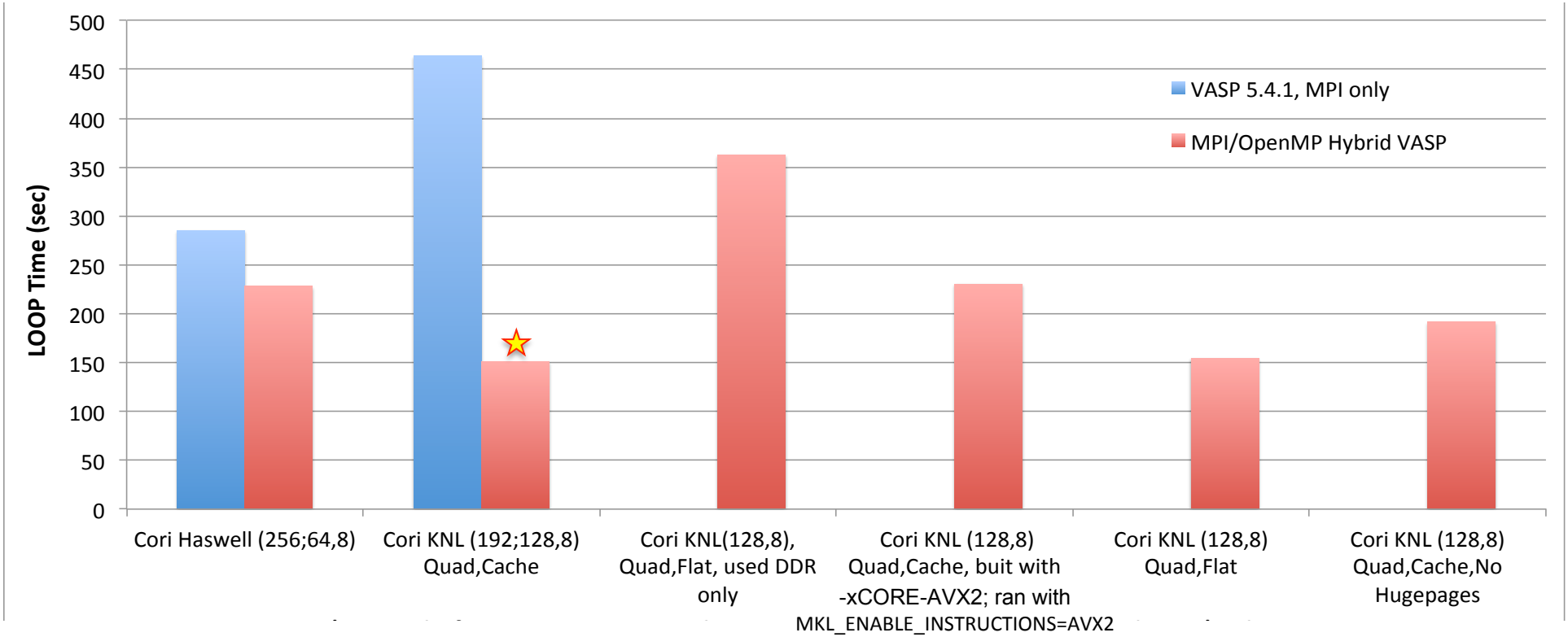


Vienna Ab initio Simulation Package (VASP), is a state-of-art electronic structure (ES) code.

- Supporting a wide range of electronic structure methods, from Density-Functional-Theory (DFT), Hartree-Fock (HF) and hybrid (HF/DFT) functionals, to the many-body-perturbative approaches based on the random-phase-approximation (GW and ACFDT).
- Solving non-linear eigenvalue problem iteratively. FFTs and Linear Algebra libraries (BLAS/LAPACK/ ScaLAPACK) are heavily depended on.
- Written in Fortran 90 and parallelized with MPI prior to the MPI/OpenMP hybrid VASP.

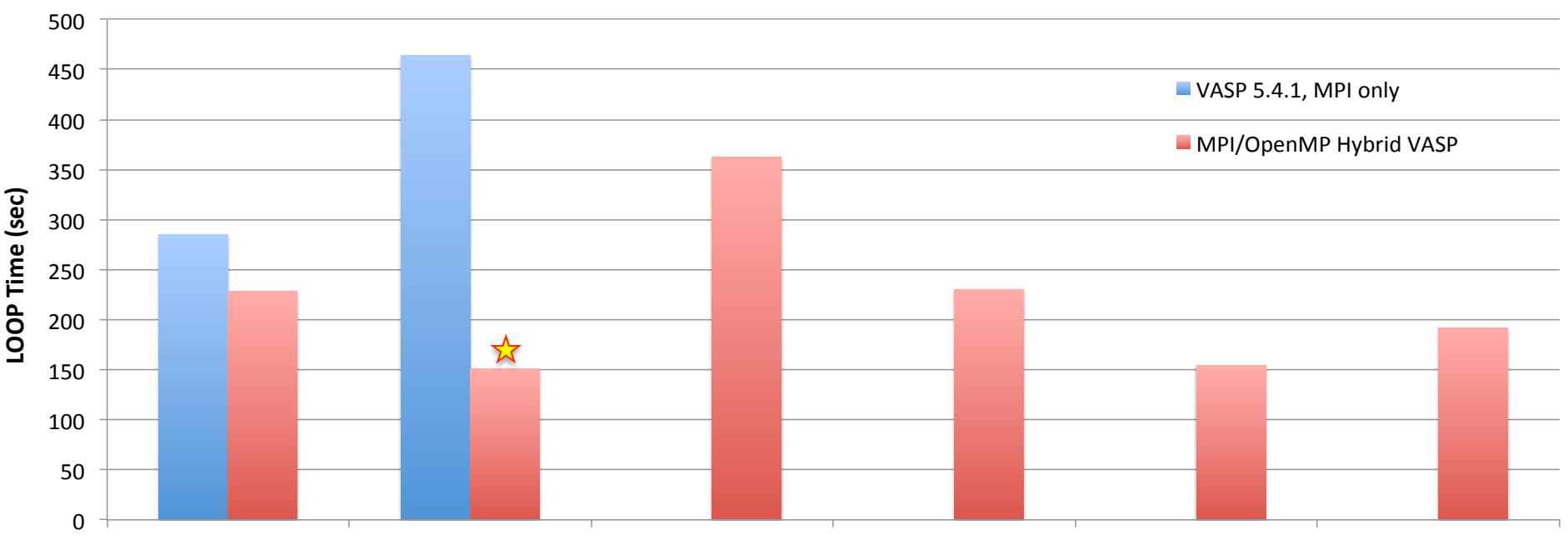
VASP, ranked #1 among ~700 application codes at NERSC, consumes more than 10-12% of the computing cycles at NERSC.

MPI/OpenMP hybrid VASP outperforms the pure MPI code by 2-3 times on Cori KNL



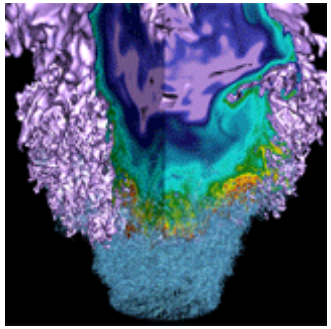
All runs used 8 Haswell or KNL nodes on Cori. The numbers inside the "()", [num;] num,num, are the number of MPI tasks used for the MPI only VASP 5.4.1, if present; the MPI tasks, OpenMP threads per task used to run the Hybrid VASP.

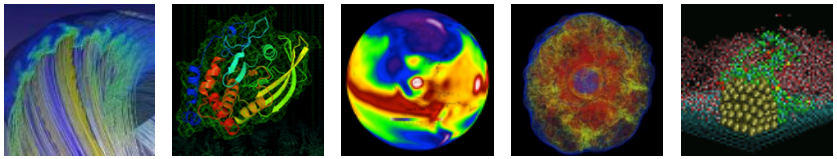
MPI/OpenMP hybrid VASP outperforms the pure MPI code by 2-3 times on Cori KNL



The optimal performance was *not possible* without using optimal build/run/boot time options and optimal number of MPI tasks and OpenMP threads.

MPI/OpenMP Hybrid VASP and Optimizations for KNL





More details on optimizations in the hybrid VASP can be found in a IWOMP17 submission:

Porting VASP from MPI to MPI + OpenMP [SIMD]
Optimization Strategies, Insights and Feature Proposals

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NERSC

universität wien

intel Look Inside.

ZIB

OpenMP threading are added into existing MPI code base

- VASP solves a set of Schrodinger-like eigenvalue/function problems
  - using iterative matrix diagonalization schemes, e.g, Blocked Davidson or RMM-DIIS.
- MPI parallelization (distributing data)
  - over the bands (high level)
  - over Fourier coefficient of the bands (low level)
- MPI + OpenMP parallelization
  - MPI over bands (high level)
  - OpenMP threading over the coefficients of bands, either by explicitly adding OpenMP directives or via using threaded FFTW and LAPACK/BLAS3 libraries
  - No nested OpenMP

$$H[\{\psi\}]\psi_n = \varepsilon_n \psi_n, \qquad n = 1,..,N$$

$$\int \psi_n^*(\mathbf{r})\psi_m(\mathbf{r})d\mathbf{r} = \delta_{nm}\,.$$

$$\Psi_n(\mathbf{r}) = FFT\{\Psi_n(\mathbf{G})\}(\mathbf{r})$$

SIMD vectorization is deployed extensively in the hybrid VASP

Either implicitly within library calls or explicitly at the loop level

- loop vectorization via !$omp simd
- function vectorization via !$omp declare simd

1. SIMD Vectorization for nested subroutines calls
   - by splitting the loops into chunks of SIMD length, pack and unpack data into vectors .
2. LOOP splitting

```
module simd
  type, public :: simd_real8
    real*8 :: x(0 : SIMD_WIDTH-1)
  end type simd_real8
  type, public :: simd_mask8
    logical :: x(0 : SIMD_WIDTH-1)
  end type simd_mask8
  ..
  interface
    function simd_exp(x) bind(c,name="__exp_finite")
!$omp declare simd (simd_exp)
      real*8 :: simd_exp
      real*8, value, intent(in) :: x
    end function simd_exp
    ..
  end interface
end module simd
```

For GNU compiler to use libmvec

```
subroutine foo(..)
  ..
  do i = 1, n

    call bar(x(i),y(i))

  enddo
  ..
end subroutine foo

subroutine bar(x, y)
  real*8 :: x, y

  y = log(x)

end subroutine bar
```

```
subroutine foo(..)
  ..
  do i = 1, n, SIMD_WIDTH
!$omp simd
    do ii = 0, SIMD_WIDTH-1
      vmask%x(ii) = .false.
      if ((i + ii) .le. n) then
        vmask%x(ii) = .true.
        vx%x(ii) = x(i + ii)
      endif
    enddo
    call vbar(vx,vy,vmask)
!$omp simd
    do ii = 0, SIMD_WIDTH-1
      if (vmask%x(ii)) x(i + ii) = vx%x(ii)
    enddo
  enddo
  ..
end subroutine foo

subroutine vbar(x, y, mask)
  type(simd_real8) :: x, y
  type(simd_mask8) :: mask
  integer :: ii
!$omp simd
  do ii = 0, SIMD_WIDTH-1
    if (mask%x(ii)) y%x(ii) = log(x%x(ii))
  enddo
end subroutine vbar
```

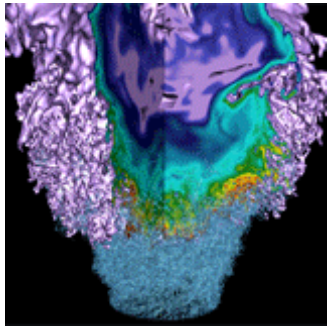
Explicit use of MCDRAM was explored in hybrid VASP via Intel compiler directive, *however,* it was not adopted in the hybrid VASP

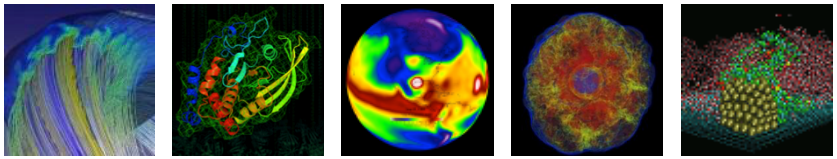
- To use MCDRAM, some of the stack variables had to be converted to allocatable heap variables, unfortunately this change itself slowed down the code significantly.
- VASP uses MCDRAM as cache or flat memory via numactl.
- Other external tools such as Intel AutoHBW can be exploited as well

```
 SUBROUTINE RACC0MU(NONLR_S, WDES1, CPROJ_LOC, CRACC, LD, NSIM, LDO)
...
   REAL(qn),ALLOCATABLE:: WORK(:),TMP(:,:)
   GDEF,ALLOCATABLE    :: CPROJ(:,:)

   !DIR$ ATTRIBUTES FASTMEM :: WORK,TMP,CPROJ
...
   ALLOCATE(WORK(ndata*NSIM*NONLR_S%IRMAX),TMP(NLM,ndata*2*NSIM),CPROJ(WDES1%NPRO_TOT,NSIM))
...
 END SUBROUTINE RACC0MU
```

Benchmarks and Performance Test setups





U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NERSC

universität wien

intel Look Inside.

ZIB

Cori hardware and software overview

- Cori, a Cray XC40 system at NERSC based on Intel KNL and Haswell architectures, interconnected with Cray Aries network
  - Cori has over 9300 single-socket **Intel® Xeon Phi™ Processor 7250 ("Knights Landing")** nodes @1.4 GHz with 68 cores (272 threads) per node, two 512 bit vector units per core, and 16 GB high bandwidth on-package memory (MCDRAM) with 5X the bandwidth of DDR4 DRAM memory (>400 GB/sec) and 96 GB DDR4 2400 MHz memory per node.
  - Cori has over 2000 dual-socket 16-cor **Intel® Xeon™ Processor E5-2698 v3 ("Haswell")** nodes @2.3GHz with 32 cores (64 threads) per node, two 256 bit vector units per core, 128 GB 2133 MHz DDR4 memory. Cori nodes are interconnected with Cray's Aries network with Dragonfly topology.
- Cori runs CLE 6.3 Update 4, and SLURM 2017.02.

Selected 6 benchmarks cover representative VASP workloads, exercising different code paths, ionic constituent and problem sizes

	PdO4	GaAsBi -64	CuC	Si256	B.hR105	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

Benchmarking approach

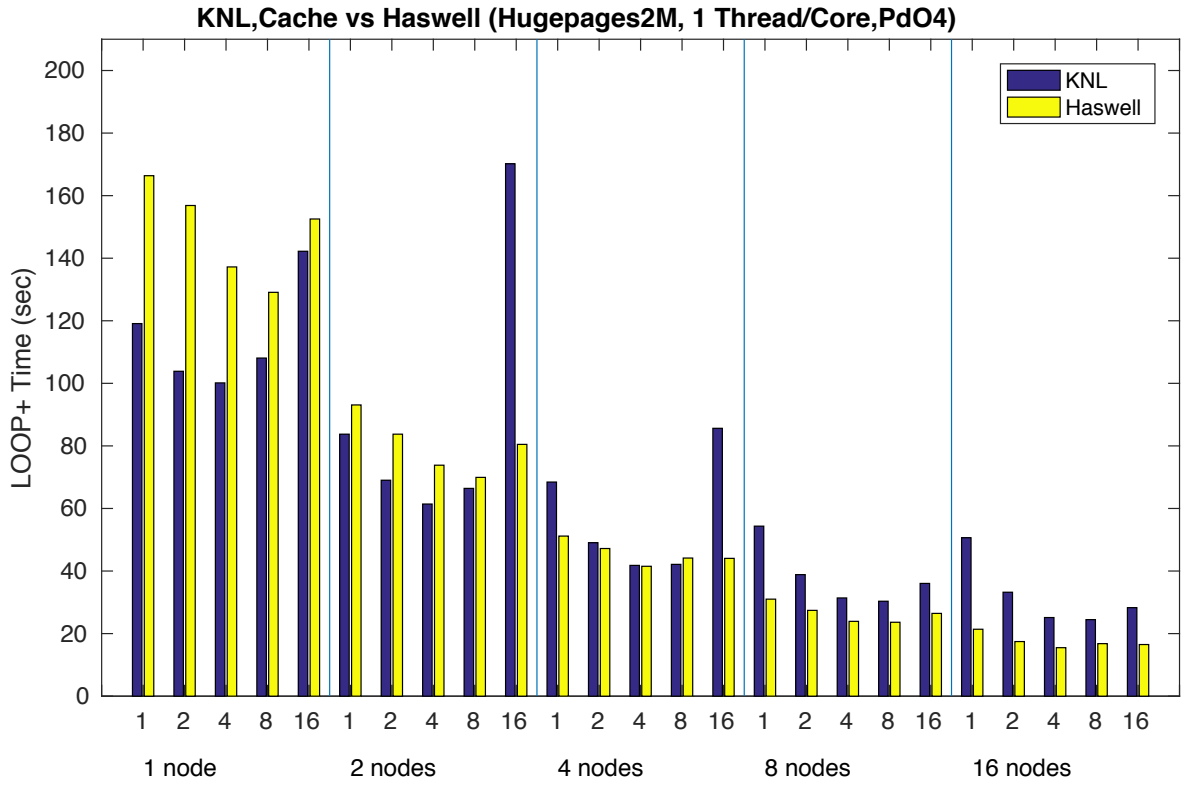
- Benchmark measures the LOOP+ time, which is the major portion of the execution time in the production execution of the VASP (disabled I/O).
- Run each benchmark multiple times (>3 times) and took the best run time.
- Process/thread affinity controlled by the OpenMP runtime (memory affinity by numactl) across all compiler builds of of VASP.
  - Export OMP_PROC_BIND=true
  - Export OMP_PLACES=Threads

VASP versions, compilers and libraries used

- MPI+OpenMP hybrid version (last commit date 4/13/2017) was used in the most of the tests, some earlier versions, e.g., 3/23/2017 was used in some of the tests as well.
- CDT 17.03 (cray-mpich/7.5.3, cray-libsci/16.11.1, fftw/ 3.4.6.6)
- Intel compiler and MKL from 2017 Update 1 + ELPA (version 2016.005)
- GNU compiler 6.3
- Cray compiler 8.5.4

MPI/OpenMP Parallel Scaling

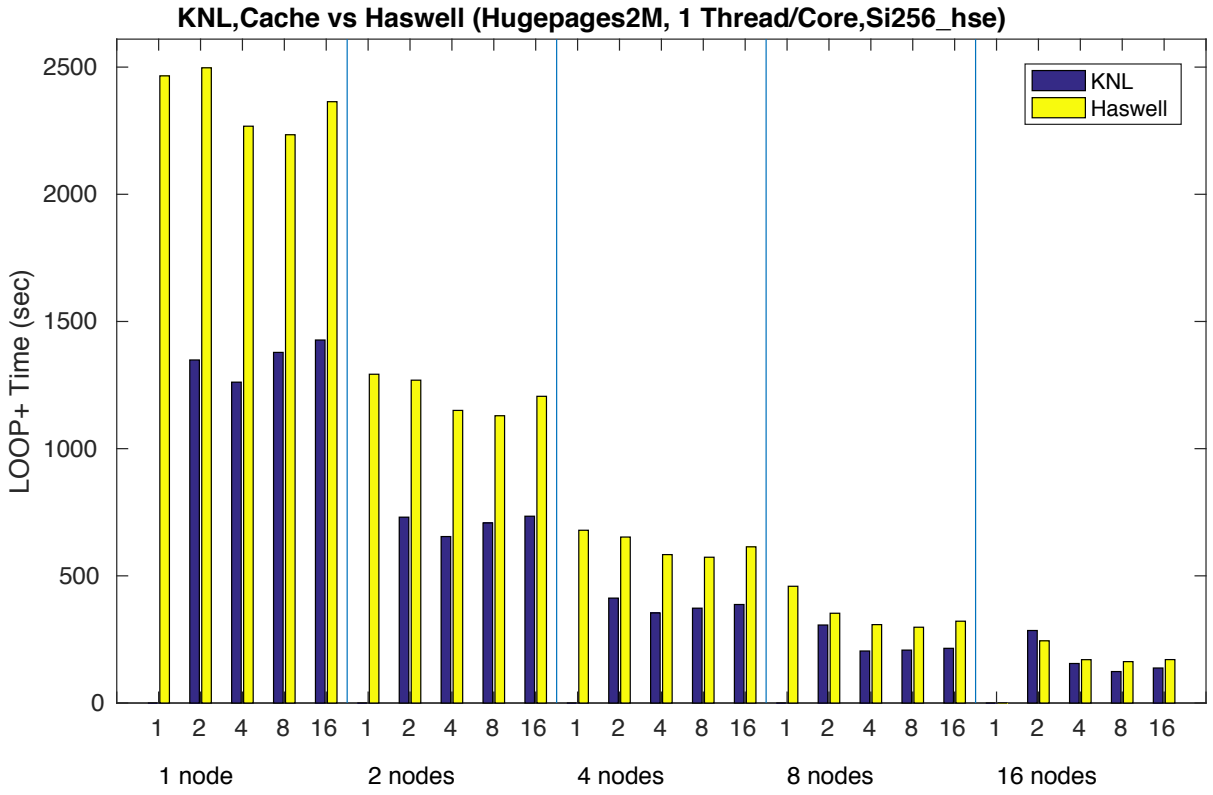
Hybrid VASP performs best with 4 or 8 OpenMP threads/ MPI task



Thread Scaling

The spikes at thread counts at 16 needs to be investigated, which could be some indication of the system issue, as Cori KNL system is undergoing continuous configuration change and system upgrades before entering productions.

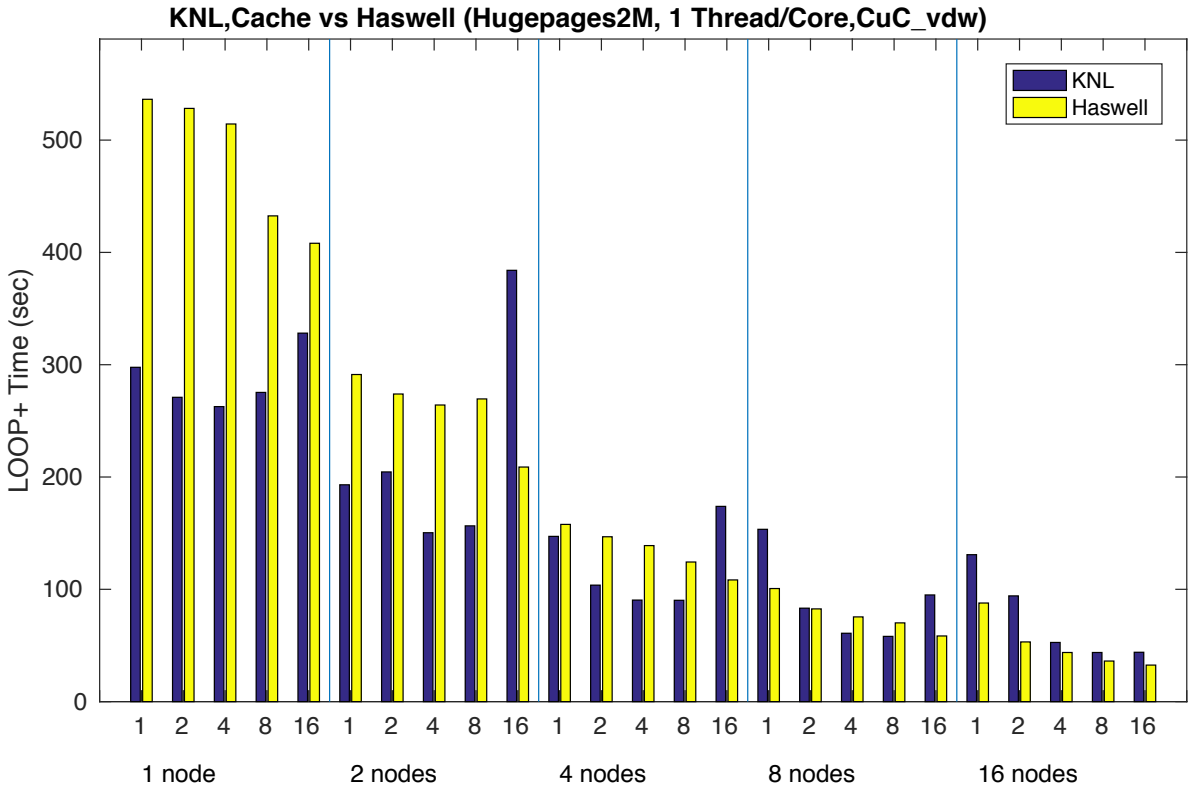
Hybrid VASP performs best with 4 or 8 OpenMP threads/ task



Thread Scaling

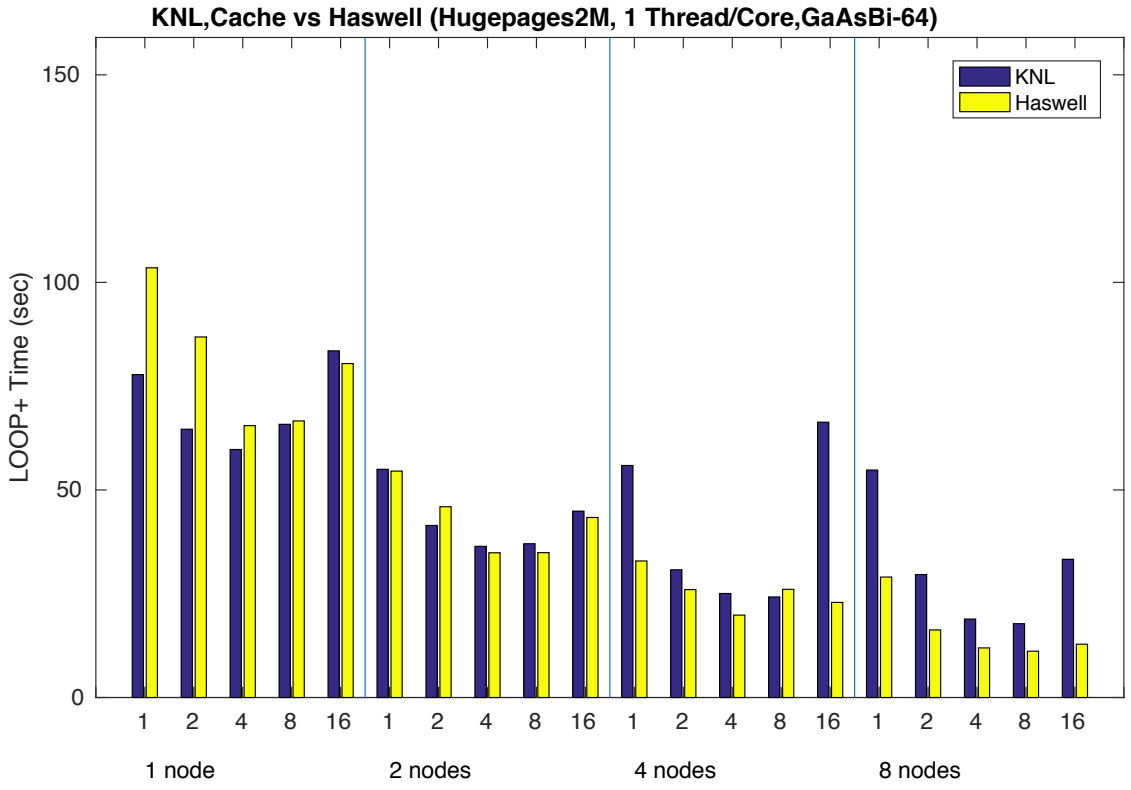
At larger node counts, the code can make use of the more threads per task (e.g, 16). This is a promising feature of the code, which opens door to scale to more nodes to solve bigger and more complex problems faster.

Using 4 and 8 threads helps the performance.



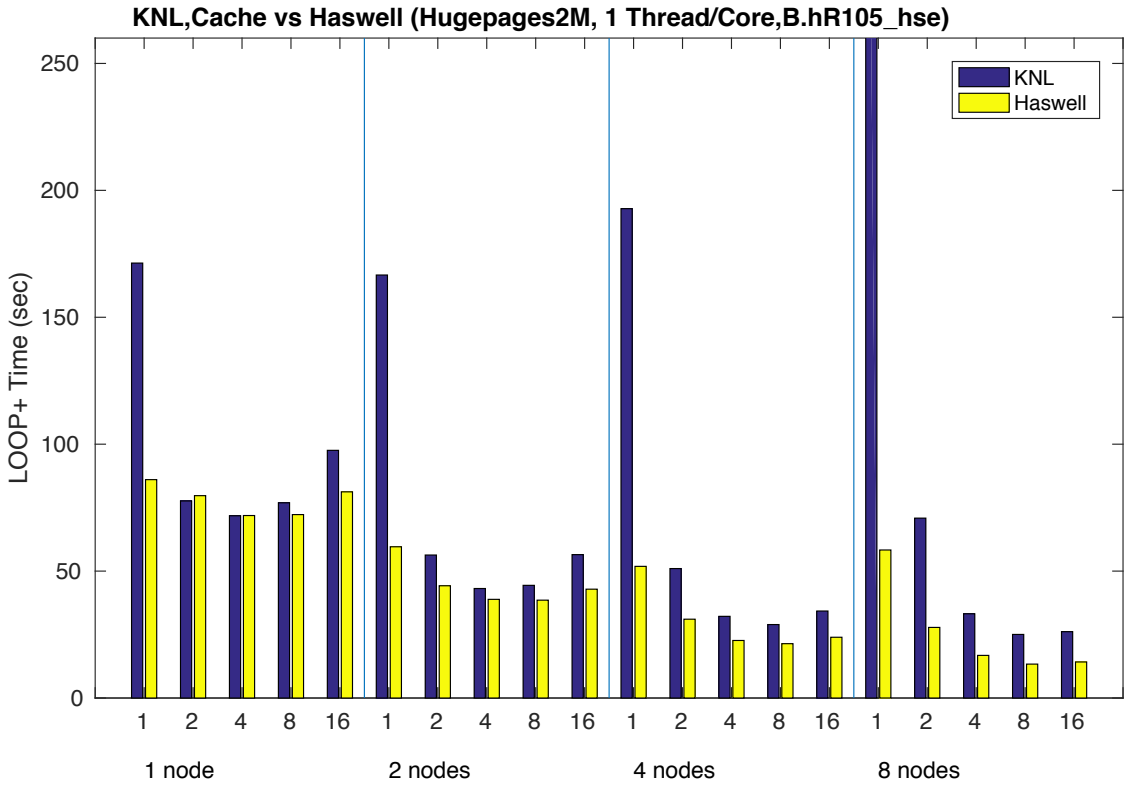
Thread Scaling

Hybrid VASP performs best with 4 or 8 OpenMP threads/task



Thread Scaling

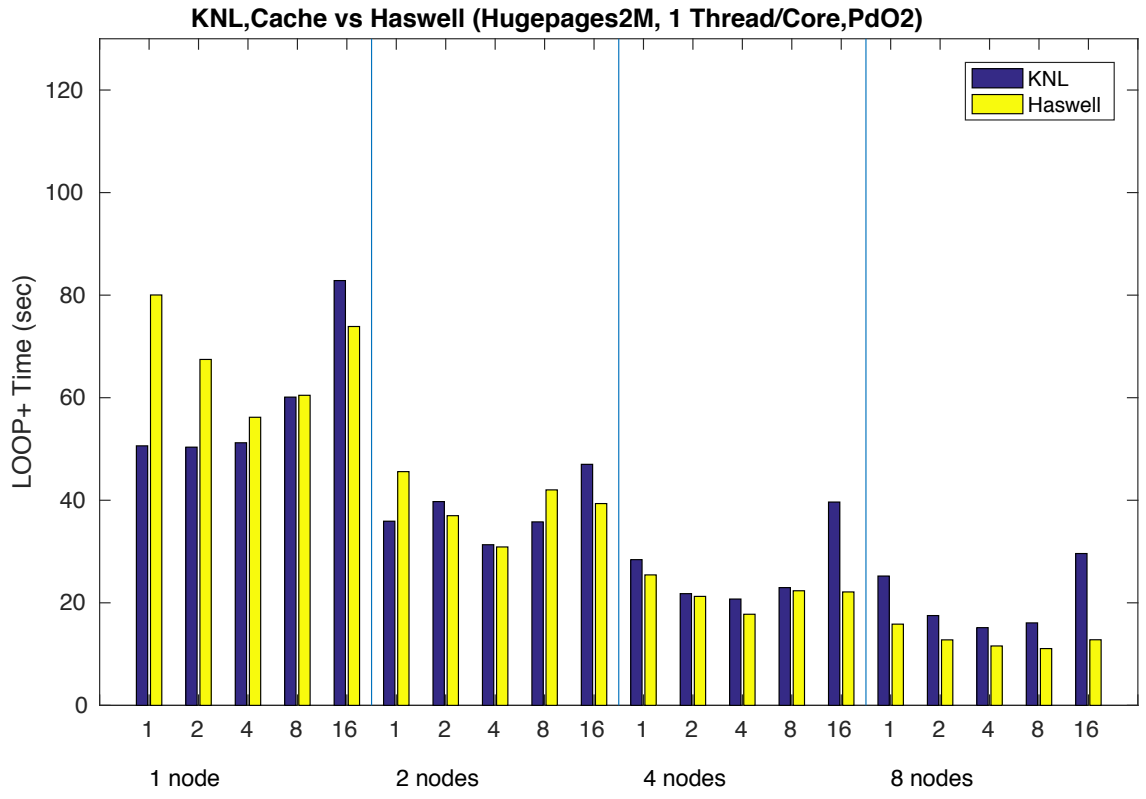
Hybrid VASP performs best with 4 or 8 OpenMP threads/task



The spikes were reproducible, need further investigation.

Thread Scaling

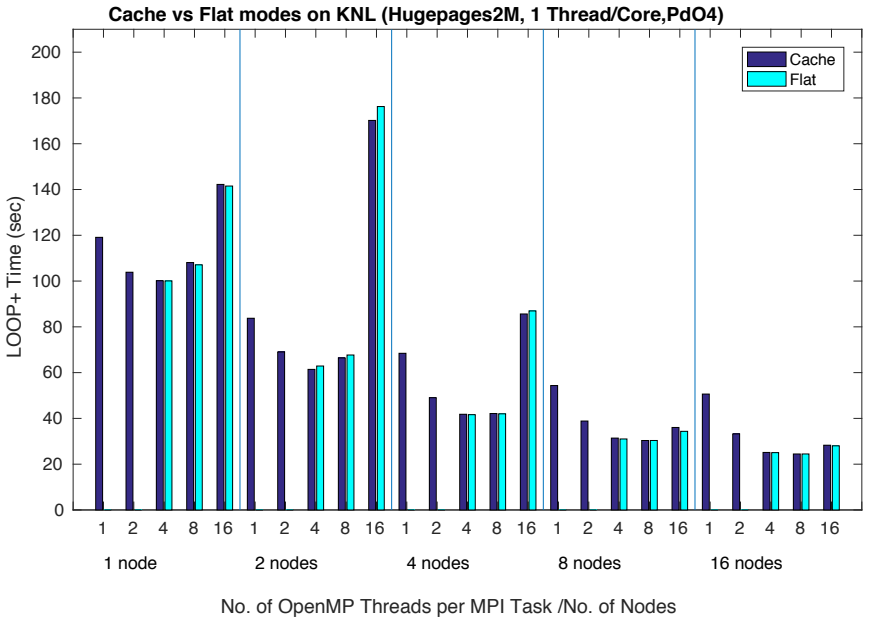
Hybrid VASP performs best with 4 or 8 OpenMP threads/ task

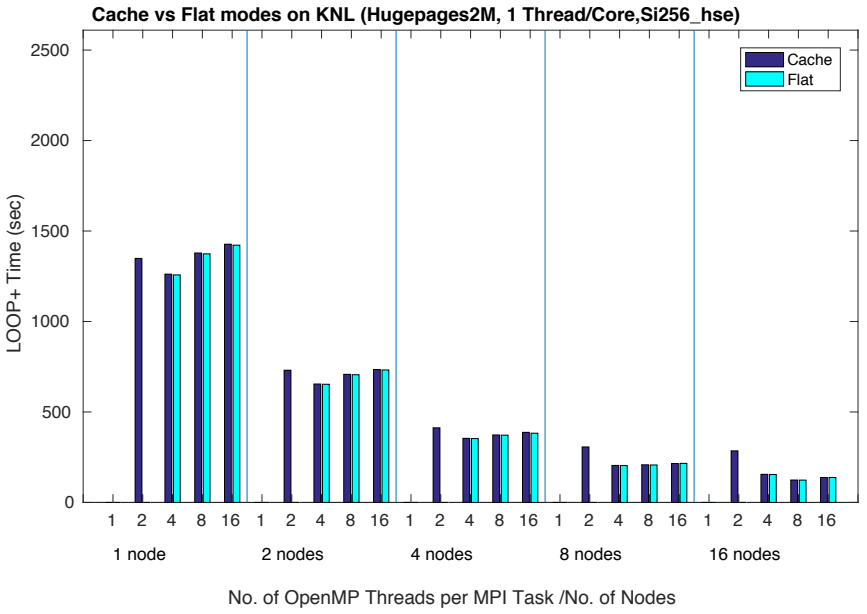


Thread Scaling

NUMA,MCDRAM Modes

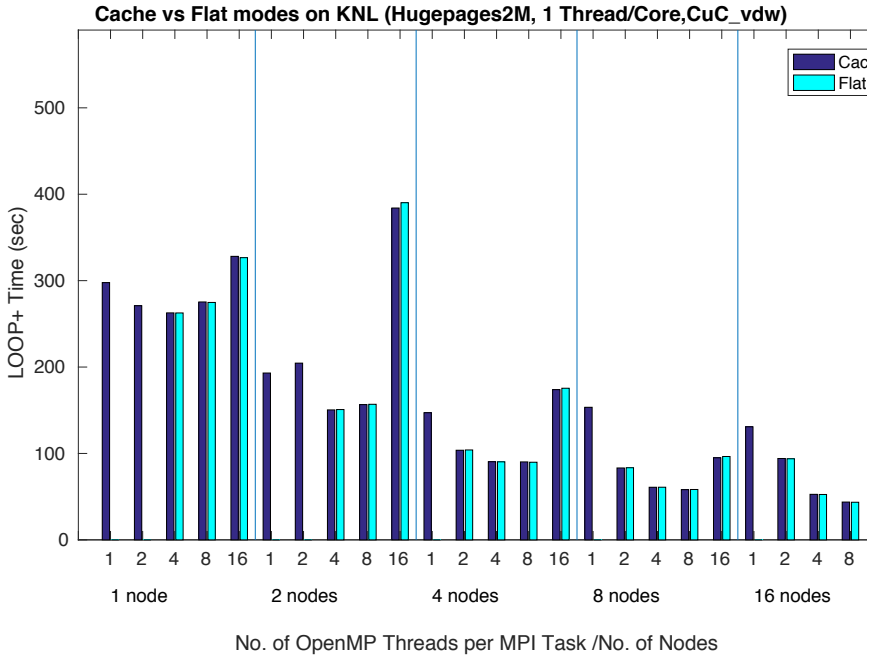
Hybrid VASP performs similarly under the cache/flat modes for the workloads that fit into MCDRAM

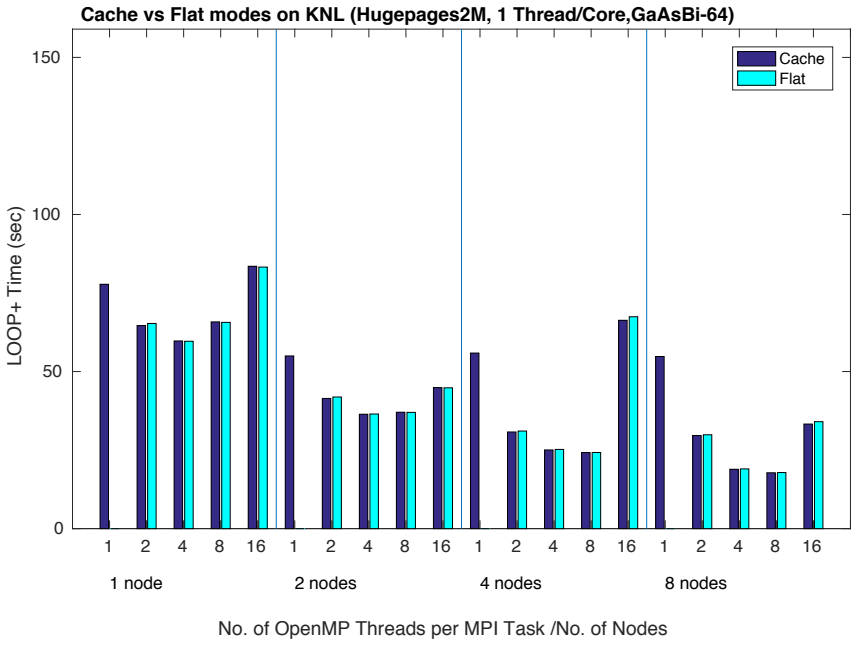




NUMA/MCDRAM

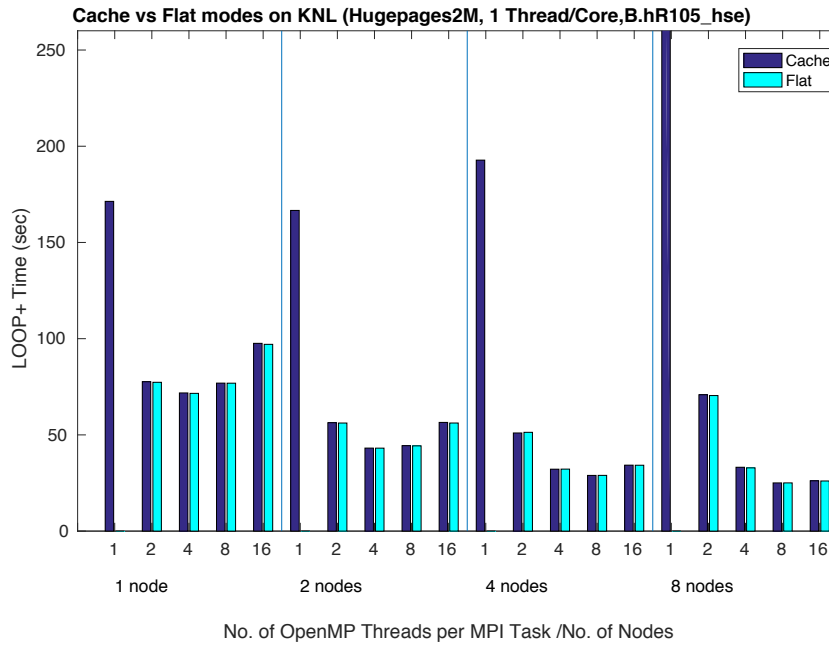
Hybrid VASP performs similarly under the cache/flat modes for the workloads that fit into MCDRAM

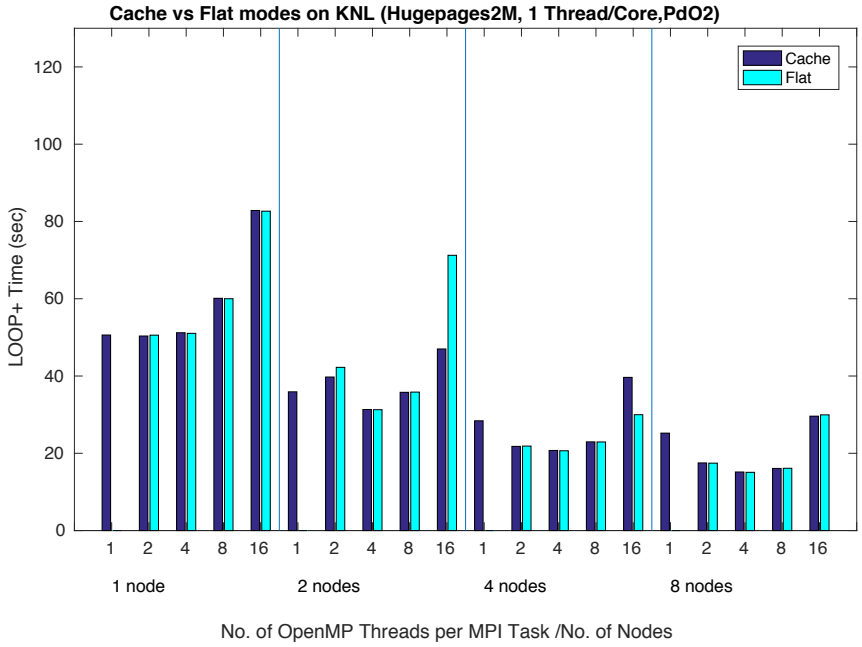




NUMA/MCDRM

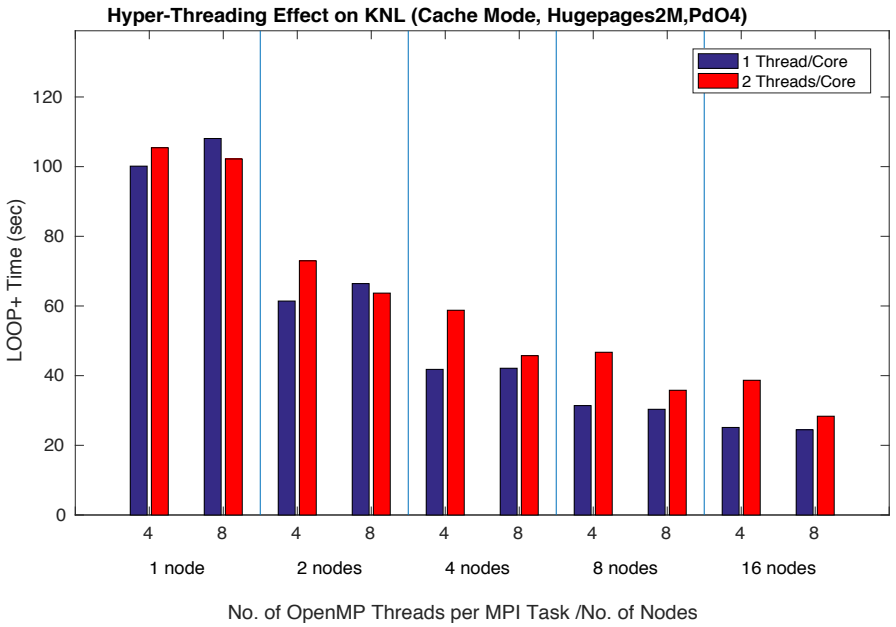
Hybrid VASP performs similarly under the cache/flat modes for the workloads that fit into MCDRAM

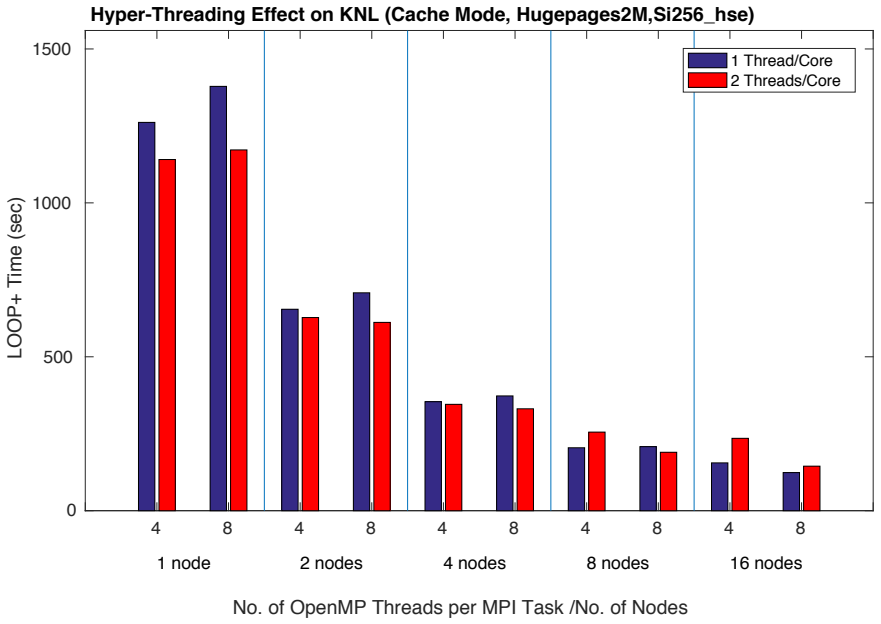




Hyper-Threading

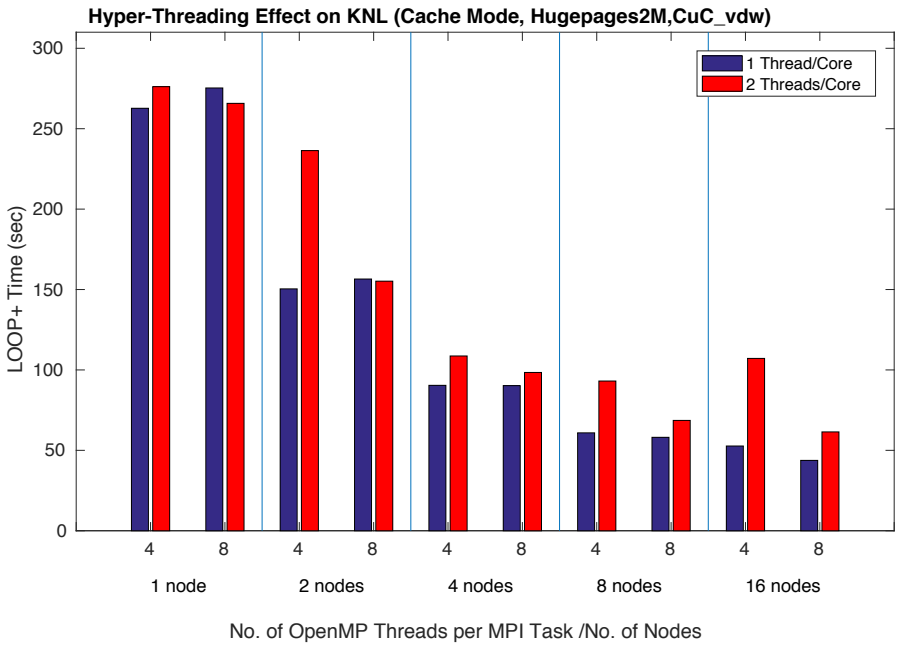
Hyper-Threading helps HSE workloads (arguably), but not other workloads in the parallel scaling regions on KNL

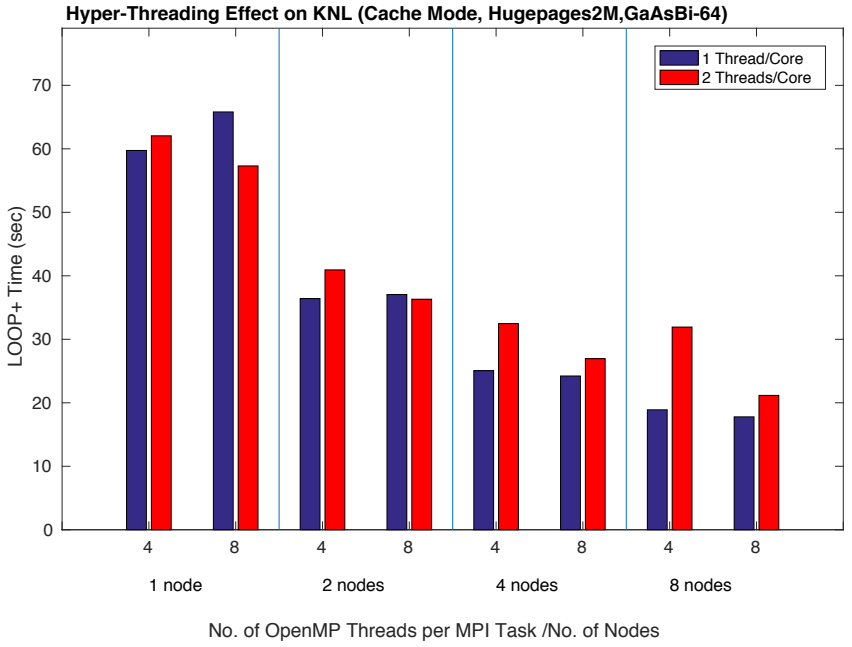




Hyper-Threading

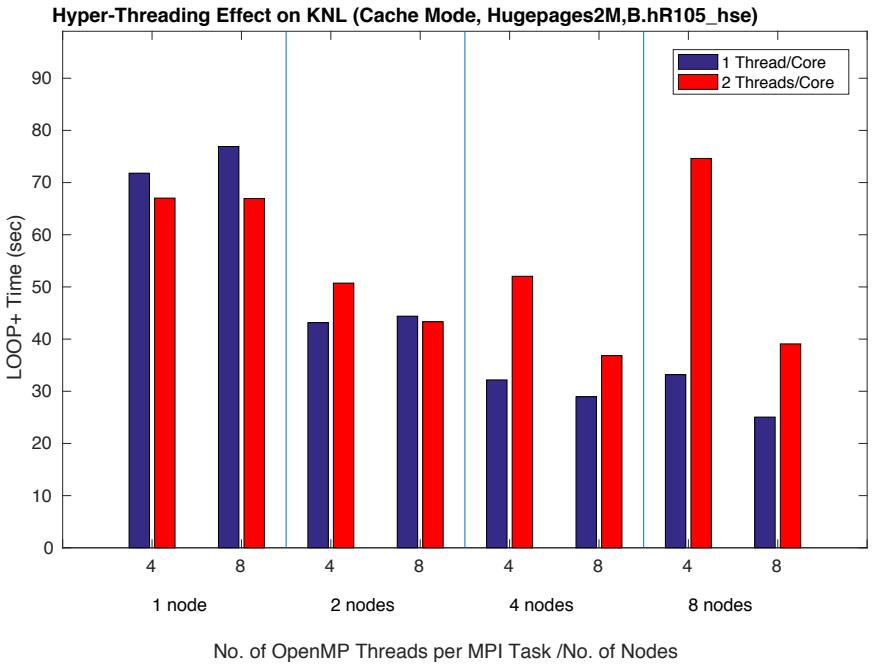
Hyper-Threading rarely helps the hybrid VASP performance on KNL

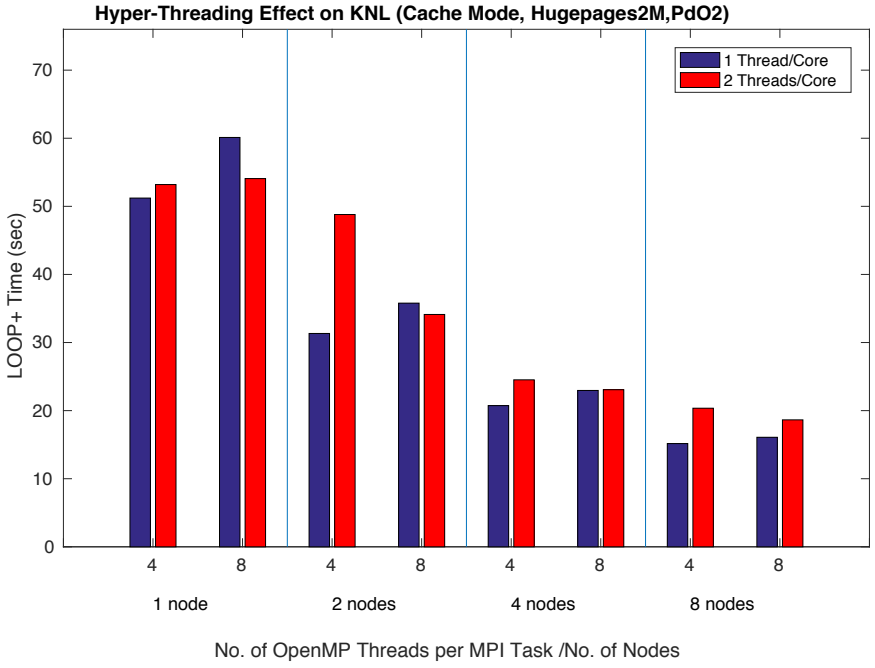




Hyper-Threading

Hyper-Threading rarely helps the hybrid VASP performance on KNL

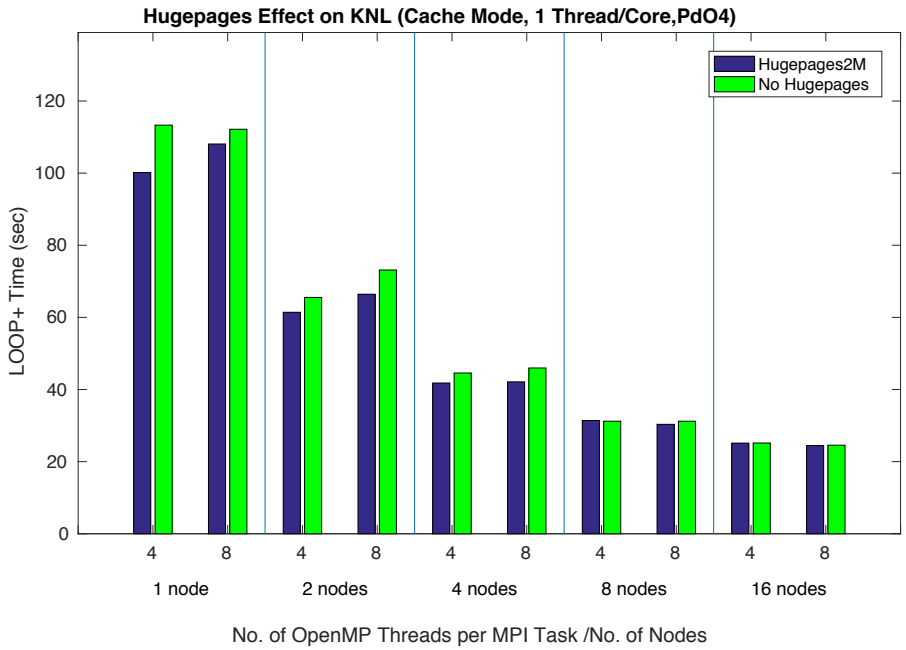


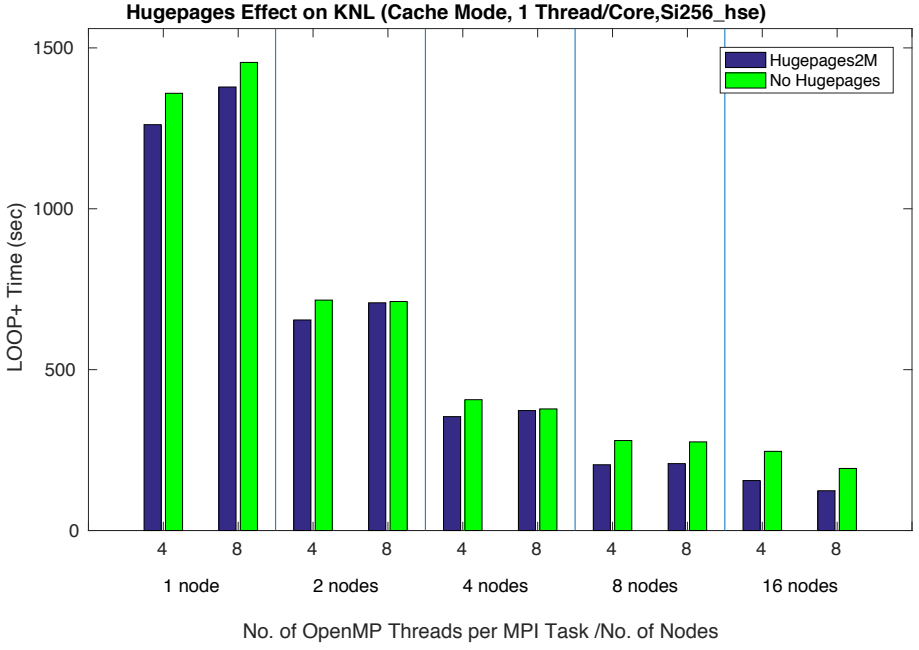


Hyper-Threading

Hugepage Memory

Hugepage memory helps hybrid VASP performance on KNL

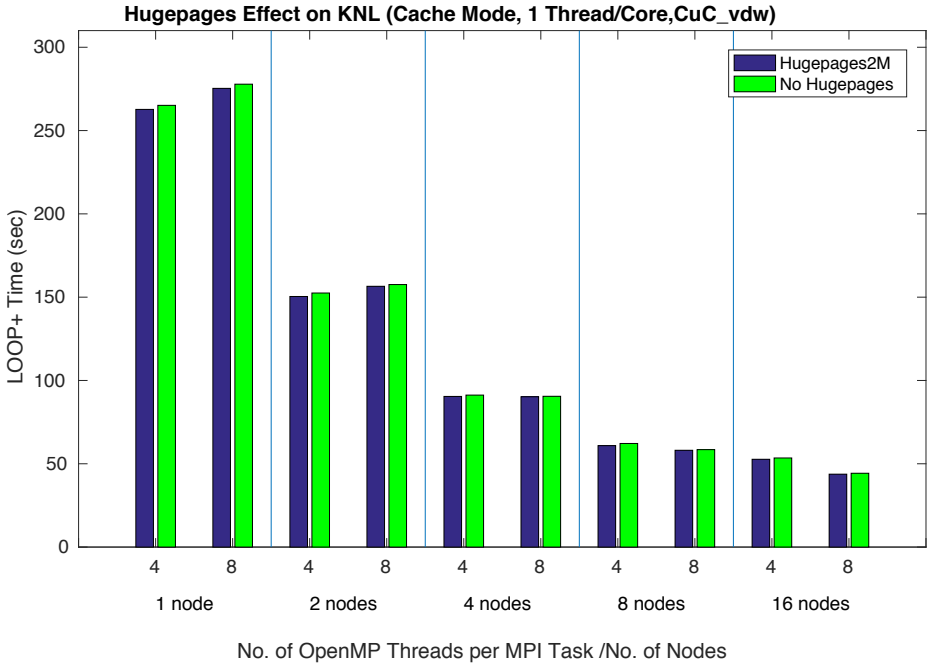


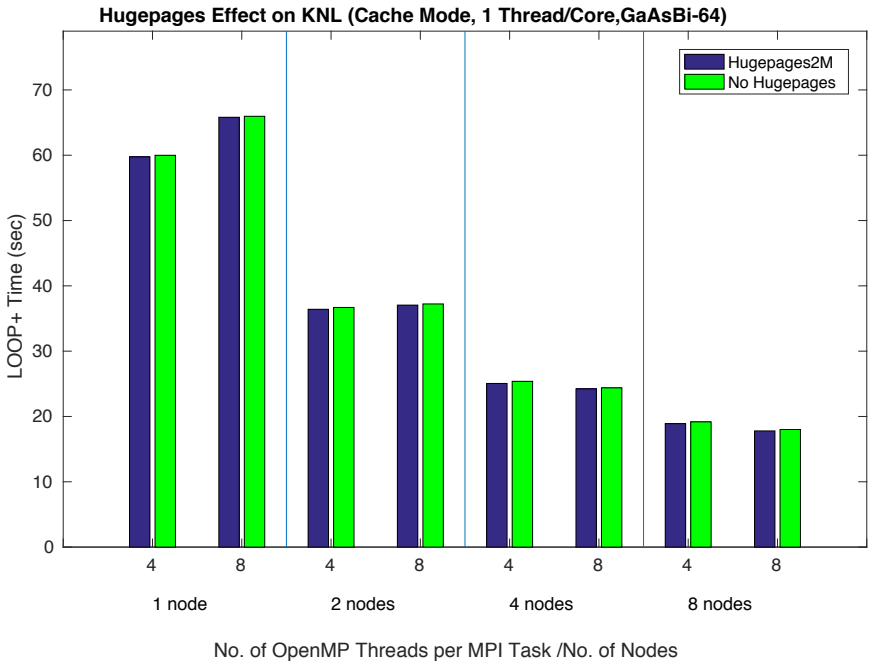


*) VASP ran out of 2M hugepage memory with 1 thread/task runs for Si256_hse.

Hugepages

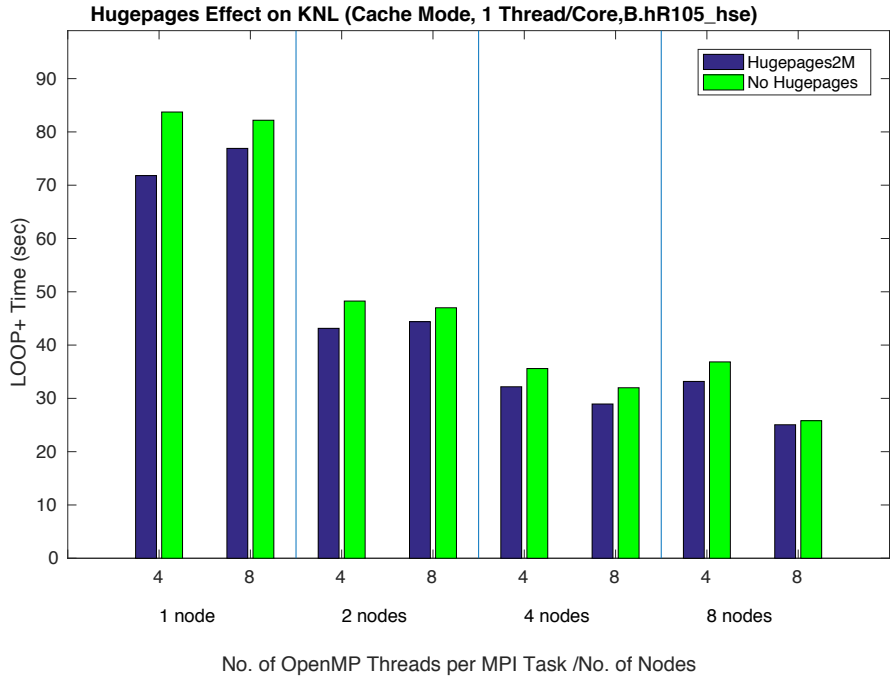
The use of hugepage memory does not slow down the code for the workloads it does not help significantly

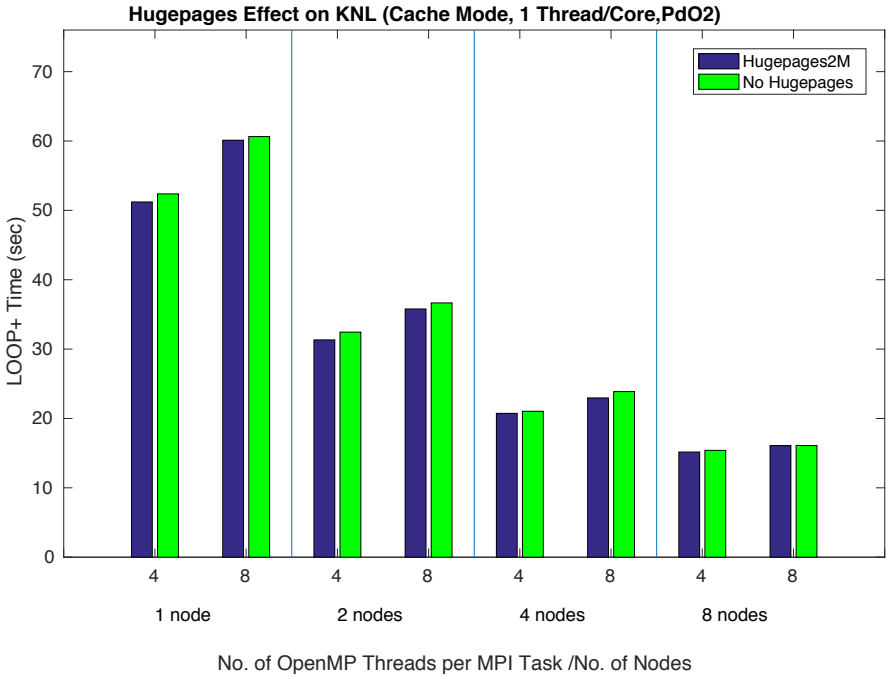




Hugepages

Hugepage memory helps hybrid VASP performance

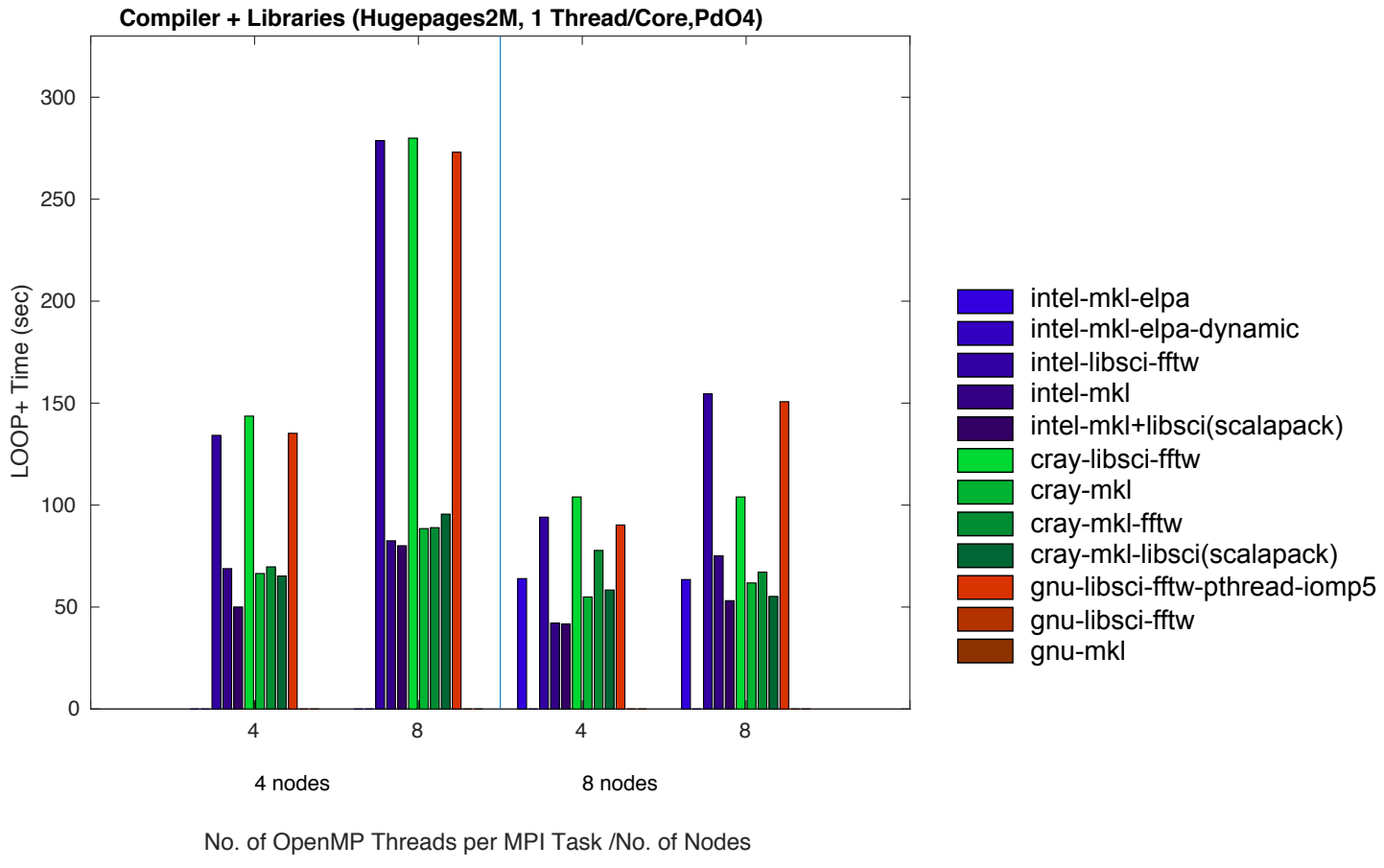




Hugepages

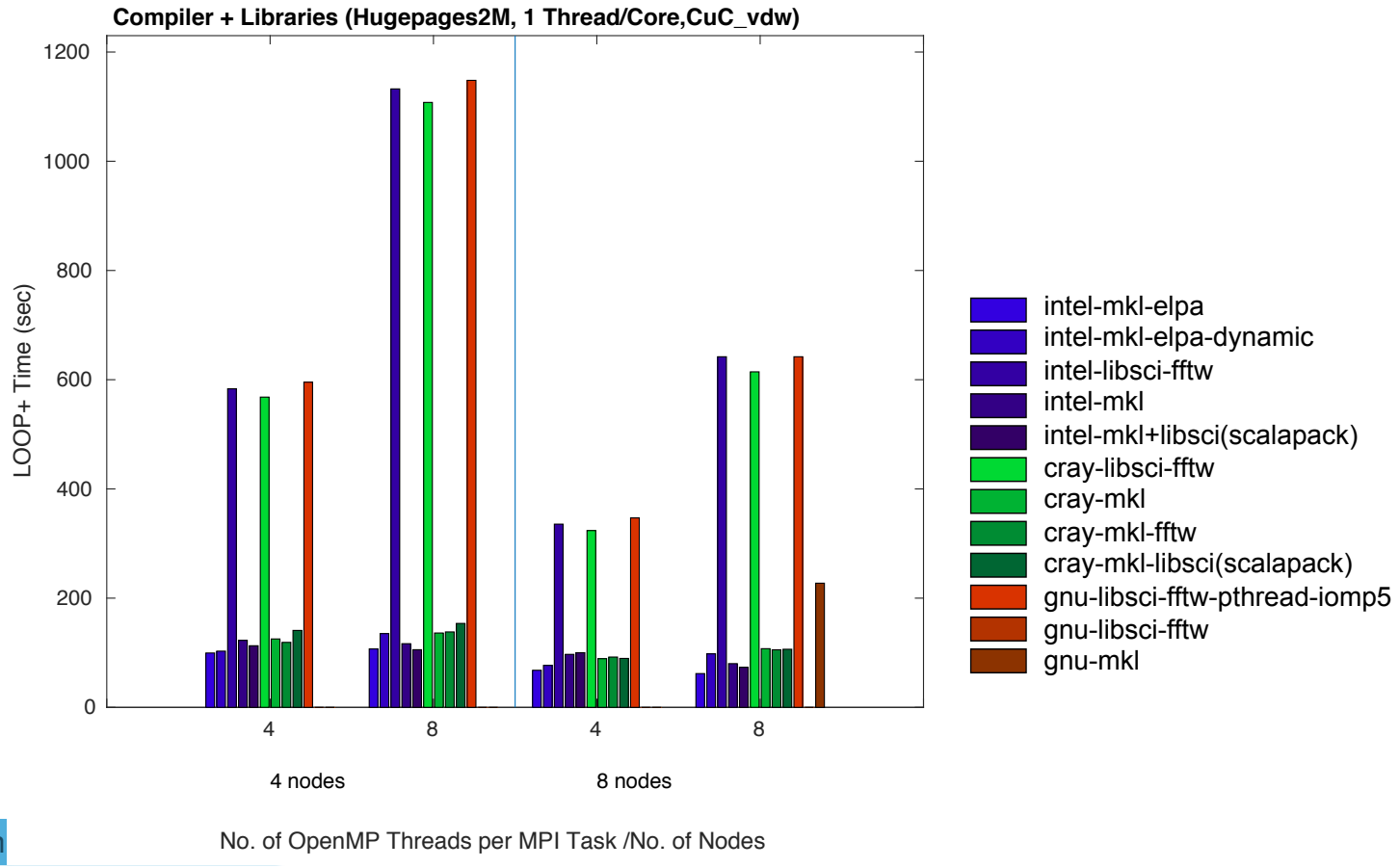
Compilers and Libraries

Hybrid VASP linked to MKL outperforms that linked to Libsci + FFTW for all three compilers (Intel, Cray and GNU) on KNL



Com

Hybrid VASP linked to MKL outperforms that linked to Libsci + FFTW for all three compilers (Intel, Cray and GNU) on KNL



Com

Summary and Future work

Conclusions and best practice tips

We studied the parallel/thread scaling of the MPI/OpenMP hybrid code with representative VASP workloads on Cori KNL system and tested the performance impact from a few build/boot/run time options. Our study shows that

1. The hybrid code performs best at 4 or 8 threads per MPI task. Using 8 threads per task in production runs is recommended.
2. Intel compilers + MKL (and FFTW interface wrappers from MKL) delivers the best performance among other compiler and library combinations, e.g., Intel, Cray and GNU compilers + Libsci and FFTW.

Best practice

3. Hugepages helps (or no hinder to) the performance almost in all cases, so the use of hugepages is recommended.
4. For the workloads that fit into MCDRAM, the cache and flat mode performs similarly. We recommend to run the hybrid VASP under the cache mode for simplicity.
5. Hybrid VASP gets most performance benefit from using MCDRAM. So it could be beneficial to use more nodes and threads (8 threads) to reduce the memory requirement per node.

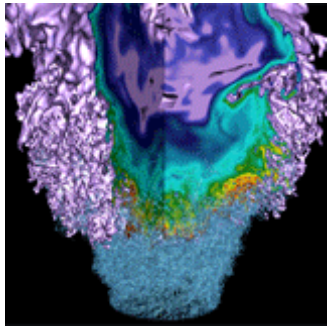
Best practice

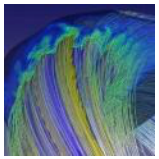
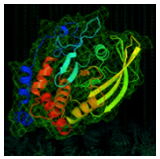
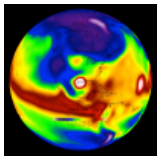
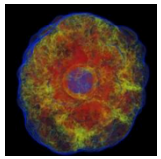
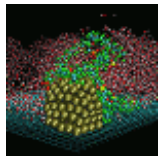
6. Using 1 hardware thread per core is recommended in general. However, hyper-threading could help the VASP performance with the HSE workloads, especially when running at a smaller node count.
7. Using 64 cores out of 68 available cores were used.

Issues and future work

- Further investigation is needed to understand the reproducible spikes in the performance data (at OpenMP thread counts 1 and 16)

Thank you!



U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

NERSC

universität wien

intel Look Inside.™

ZIB