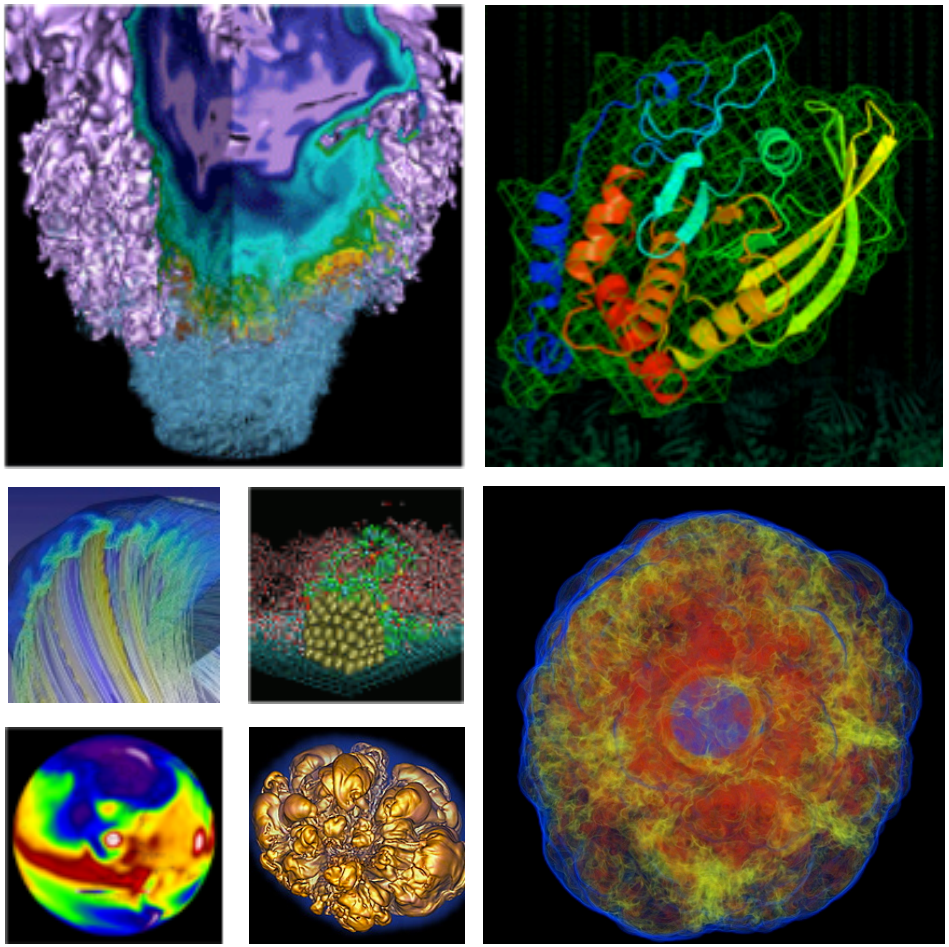
Using VASP at NERSC



Zhengji Zhao

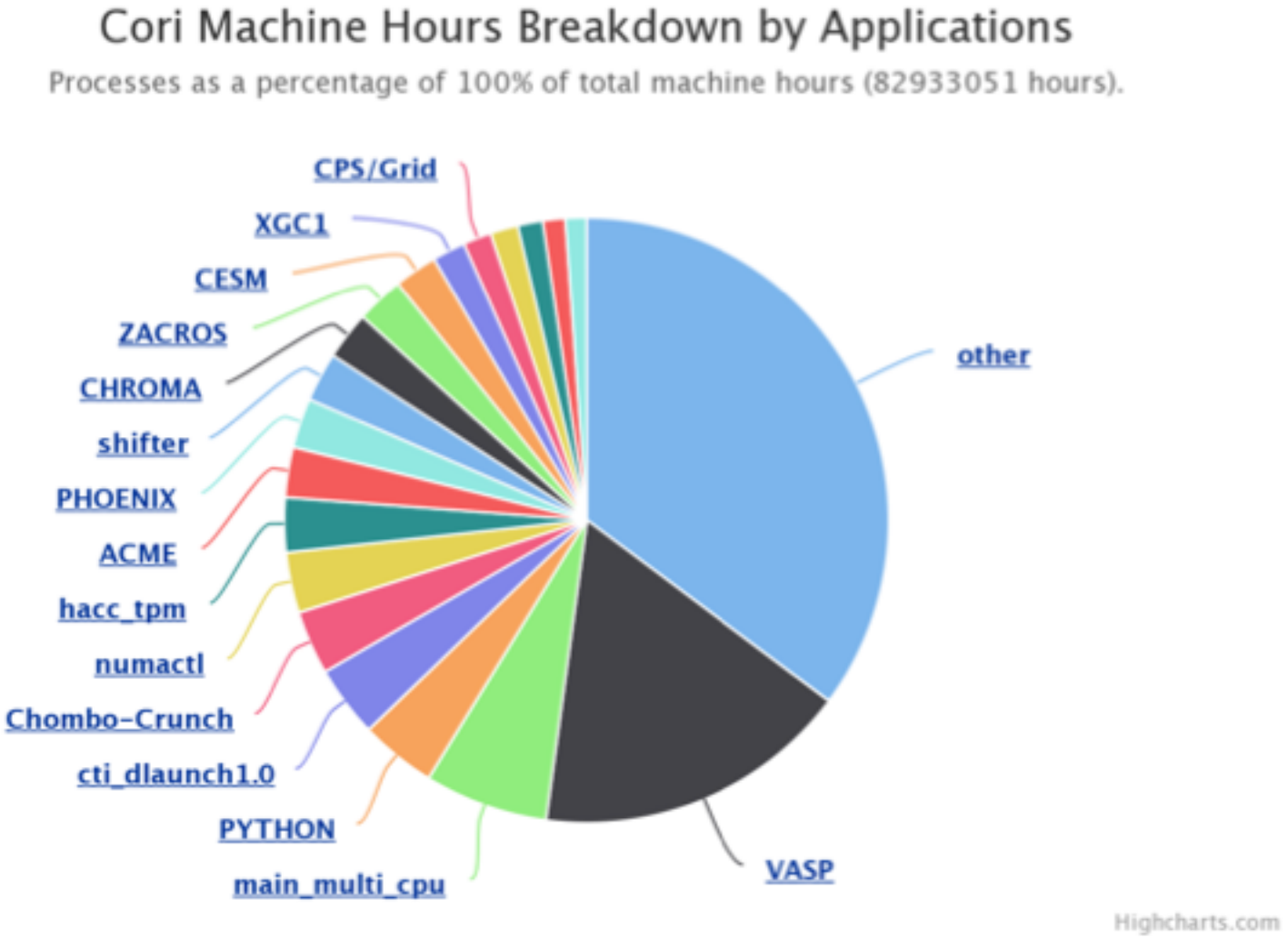
VASP Training Webinar,
June 29, 2018, Berkeley CA

NERSC 40 YEARS at the FOREFRONT 1974-2014

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

VASP used more than 15% of Cori machine time last year (6/27/17 – 6/26/2018)

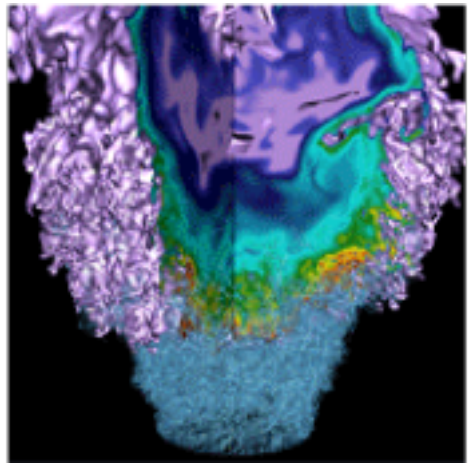


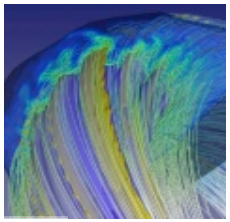
https://my.nersc.gov/application_usage_page.php

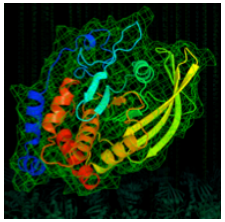
Outline

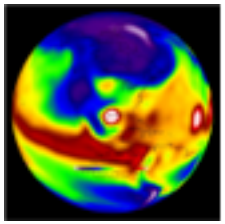
- VASP access at NERSC
- Available VASP modules
- Running VASP on Cori and Edison
- Performance of Hybrid VASP
- Best practices

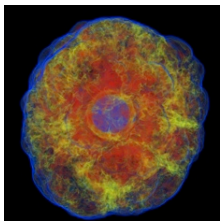
VASP Access at NERSC

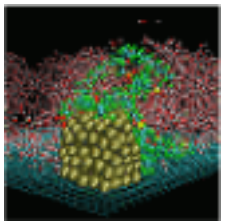












NERSC 40 YEARS at the FOREFRONT

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

All users who want to access the VASP binaries provided at NERSC need to confirm their licenses.

- VASP is open to the users who have licenses by themselves.
- Instructions on how to confirm VASP licenses is available at http://www.nersc.gov/users/software/applications/materials-science/vasp#toc-anchor-2
- Also available in the vasp modules (type module show vasp)

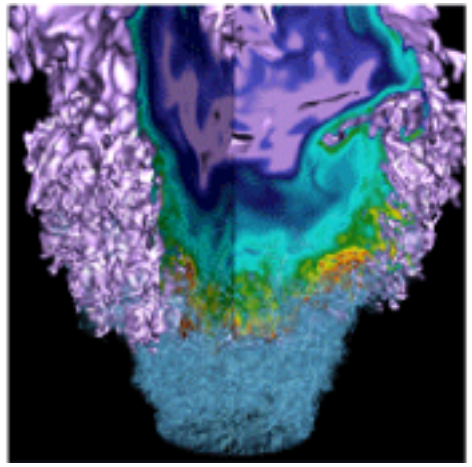
```
zz217@cori05:~> module show vasp
-------------------------------------------------------------------
/usr/common/software/modulefiles/vasp/5.4.4-hsw:

module-whatis    VASP: Vienna Ab-initio Simulation Package

Access to the vasp suite is allowed only for research groups with existing
licenses for VASP. If you have a VASP license please email

 vasp.Materialphysik@univie.ac.at and CC: vasp_licensing@nersc.gov

with the information on which research group your license derives from.
The PI of the group as well as the institution and license number will help
speed the process.

setenv           PSEUDOPOTENTIAL_DIR /global/common/sw/cray/cnl6/haswell/vasp/pseudopotentials
setenv           VDW_KERNAL_DIR /global/common/sw/cray/cnl6/haswell/vasp/vdw_kernal
setenv           NO_STOP_MESSAGE 1
setenv           MPICH_NO_BUFFER_ALIAS_CHECK 1
prepend-path     PATH /global/common/sw/cray/cnl6/haswell/vasp/vtstscripts/r933
prepend-path     PATH /global/common/sw/cray/cnl6/haswell/vasp/5.4.4/intel/17.0.2.174/4bqi2il/bin
-------------------------------------------------------------------
```

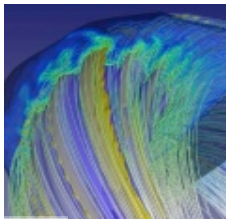
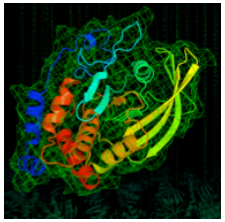
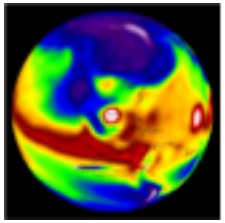
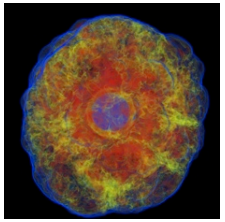
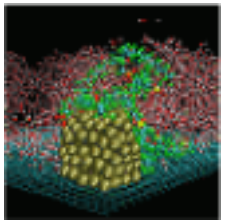
How to check if you have the VASP access or not

- **Usually it takes a few business days to confirm a VASP license.**
  - The license owner (PI) must register you under his/her license.
  - Each individual user needs to confirm, no group (repo) access
- **VASP access is controlled by a Unix file group vasp5.**
  - Type the "groups" command on Cori or Edison, if you do not see "vasp5" in your group list, then you don't have the VASP access at NERSC.
  - Attempting to run VASP would run into the following error:

```
fbench@nid00126:~> module load vasp
fbench@nid00126:~> srun -n 4 -c 16 --cpu_bind=cores vasp_std
srun: fatal: Can not execute vasp_std
Aborted
```

Pre-compiled VASP Modules



The precompiled VASP binaries are available via modules.

module avail vasp #to see the available modules
module show vasp # to see what vasp modules do
module load vasp #to access the VASP binaries

Available vasp modules on Cori

- **Type "module avail vasp" to see the available VASP modules**

	Standard distribution	VASP with third party codes (tpc) (Wannier90,VTST,BEEF,VASPSol)
Hybrid MPI+OpenMP VASP	vasp/20171017-hsw	
	vasp/20171017-knl	
	vasp/20170629-hsw	vasp-tpc/20170629-hsw (experimental)
	vasp/20170629-knl	vasp-tpc/20170629-knl (experimental)
Pure MPI VASP	vasp/5.4.4-hsw (default)	vasp-tpc/5.4.4-hsw (default)
	vasp/5.4.4-knl	vasp-tpc/5.4.4-knl
	vasp/5.4.1-hsw	vasp-tpc/5.4.1-hsw
	vasp/5.4.1-knl	vasp-tpc/5.4.1-knl

- **To be removed modules:**

vasp/5.4.4
vasp/5.4.1
vasp/5.4.1-cce
vasp/5.4.1-gcc
vasp/5.3.5
vasp/5.4.1_vtst
vasp/5.4.1_vtst-gcc
vasp-tpc/5.4.1
vasp/5.3.5_vtst
vasp/20170323_NMAX_DEG=128-hsw
vasp/20170323_NMAX_DEG=128-knl

Available vasp modules on Edison

	Standard distribution	VASP with third party codes (tpc) incorporated
Pure MPI VASP	vasp/5.4.4 (default)	vasp-tpc/5.4.4
	vasp/5.4.1	vasp-tpc/5.4.1
	vasp/5.3.5	vasp-tpc/5.4.4.NMAX_DEG=128

- **Type module show vasp to see what vasp modules do**

```
zz217@cori03:~> module show vasp
-------------------------------------------------------------------
/usr/common/software/modulefiles/vasp/5.4.4-hsw:
…

setenv PSEUDOPOTENTIAL_DIR /global/common/sw/cray/cnl6/haswell/vasp/pseudopotentials
setenv VDW_KERNAL_DIR /global/common/sw/cray/cnl6/haswell/vasp/vdw_kernal
setenv NO_STOP_MESSAGE 1
setenv MPICH_NO_BUFFER_ALIAS_CHECK 1
prepend-path PATH /global/common/sw/cray/cnl6/haswell/vasp/vtstscripts/r933
prepend-path PATH /global/common/sw/cray/cnl6/haswell/vasp/5.4.4/intel/17.0.2.174/4bqi2il/bin
-------------------------------------------------------------------
```

- **Type ls –l to see the available VASP binaries**

```
zz217@cori03:~> ls -l /global/common/sw/cray/cnl6/haswell/vasp/5.4.4/intel/17.0.2.174/4bqi2il/bin
total 326064
-rwxrwxr-x 1 swowner swowner 110751840 Feb 10 14:59 vasp_gam
-rwxrwxr-x 1 swowner swowner 111592800 Feb 10 14:59 vasp_ncl
-rwxrwxr-x 1 swowner swowner 111541384 Feb 10 14:59 vasp_std
zz217@cori03:~>
```

- **Do module load vasp to access the VASP binaries**

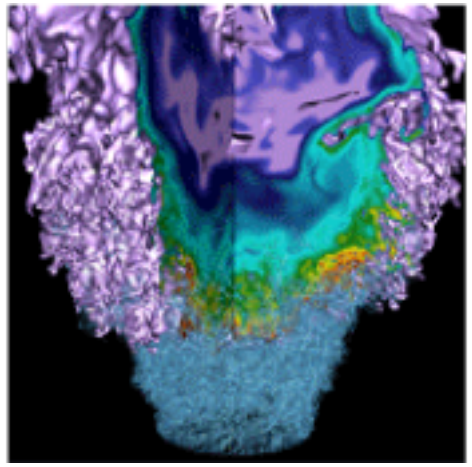
```
zz217@cori03:~> module load vasp

zz217@cori03:~> which vasp_std
/global/common/sw/cray/cnl6/haswell/vasp/5.4.4/intel/17.0.2.174/4bqi2il/bin/vasp_std
```

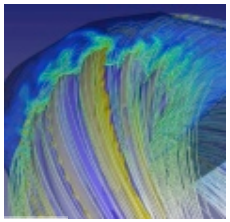
- **The VTST Scripts are made available, as well as pseudo files and makefiles for users who want to build VASP by themselves**

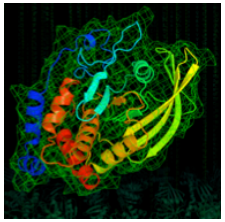
Notes

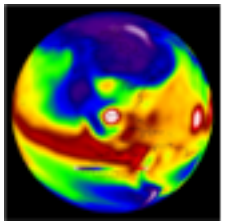
- **The hybrid MPI+OpenMP VASP does not use NCORE/NPAR for multi-thread runs. The code sets NCORE =1 internally if any NCORE !=1 is encountered.**
- **However, when only 1 thread per task is used (OMP_NUM_THREADS=1), then the NCORE/NPAR values that users provided in the INCAR files are honored.**
- **The hybrid VASP + third party contributed code builds are experimental, meaning they may not work with all cases. If you run into any issues, please run with OMP_NUM_THREADS=1, this seems to as well as the Pure MPI verion +TPC.**

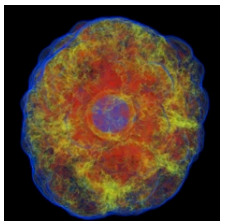
Running VASP on Cori and Edison

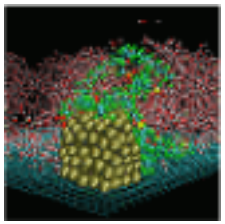












NERSC 40 YEARS at the FOREFRONT

System Configurations

System	# of cores/ CPUs per node	# of sockets per node	Clock Speed (GHz)	Memory /node	Memory/core
Cori KNL (9688 nodes)	68/272	1	1.4	96 GB DDR4 @2400 MHz 16GB MCDRAM as cache	1.4 GB DDR 325 MB MCDRAM
Cori Haswell (2388 nodes)	32/64	2	2.3	128 GB DDR4 @2133 MHz	4.0 GB
Edison (5586 nodes)	24/48	2	2.4	64 GB DDR3 @1866 MHz	2.67 GB

Machine charge factors:

Edison:48
Cori Haswell: 80
Cori KNL: 96

Running interactive VASP jobs on Cori

- **The interactive QOS allows a quick access to the compute nodes up to 4 hours and 64 nodes. The run limit is 1.**
  - It either allocates the requested nodes in less than 5 minutes or it cancels the job.

```
zz217@cori03:/global/cscratch1/sd/zz217/PdO4> salloc -N4 -C knl -q interactive -t 4:00:00
salloc: Granted job allocation 13460931

zz217@nid02305:/global/cscratch1/sd/zz217/PdO4> module load vasp/20171017-knl

zz217@nid02305:/global/cscratch1/sd/zz217/PdO4> export OMP_NUM_THREADS=4

zz217@nid02305:/global/cscratch1/sd/zz217/PdO4> srun -n64 -c16 --cpu_bind=cores vasp_std
 ----------------------------------------------------------
    OOO  PPPP  EEEEE N   N M   M PPPP
   O   O P   P E     NN  N MM MM P   P
   O   O PPPP  EEEEE N N N M M M PPPP   -- VERSION
   O   O P     E     N  NN M   M P
    OOO  P     EEEEE N   N M   M P
 ----------------------------------------------------------
 running   64 mpi-ranks, with    4 threads/r
 distrk:  each k-point on   64 cores,    1 g
 distr:  one band on    1 cores,   64 groups

  ...
```

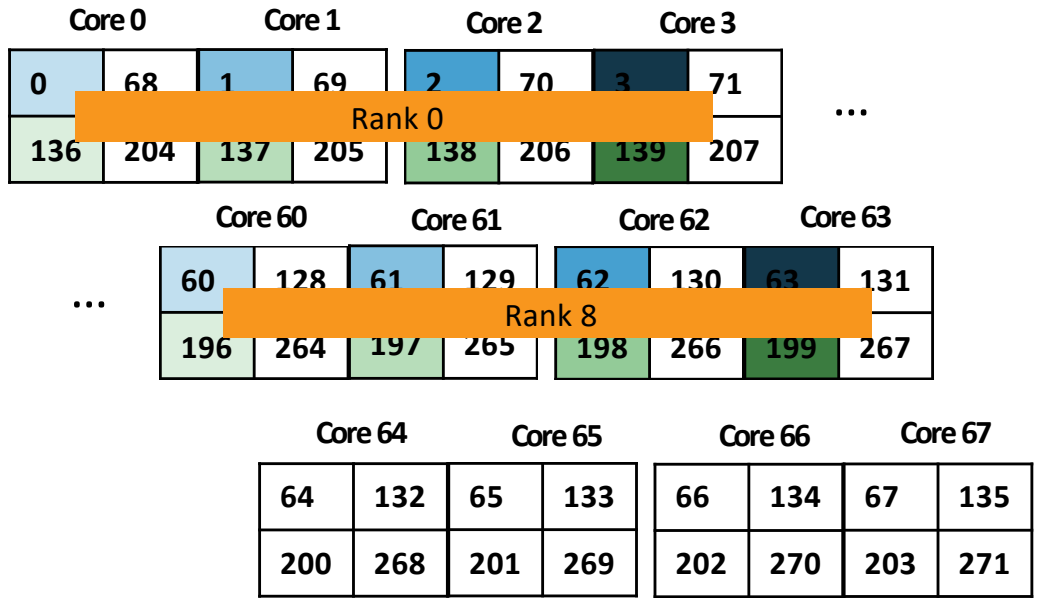
```
zz217@cori11:~> sacct -j 13460931 -o reserved,start,submit
  Reserved               Start              Submit
---------- ------------------- -------------------
  00:00:00 2018-06-28T23:31:07 2018-06-28T23:31:07
```

- **No interactive QOS available on Edison. Use the debug QOS.**

Sample job script to run the hybrid MPI+OpenMP VASP with 8 threads per MPI task on Cori KNL nodes

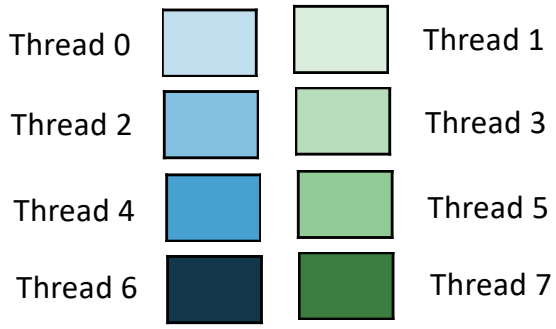
```
#!/bin/bash -l
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 2:00:00
#SBATCH -C knl
#SBATCH –L SCRATCH

module load vasp/20171017-knl
export OMP_NUM_THREADS=8

srun –n8 –c32 --cpu_bind=cores vasp_std
```

Process/thread affinity outcome



The above script requests 1 KNL node for two hours using the regular QOS and using the scratch file system.

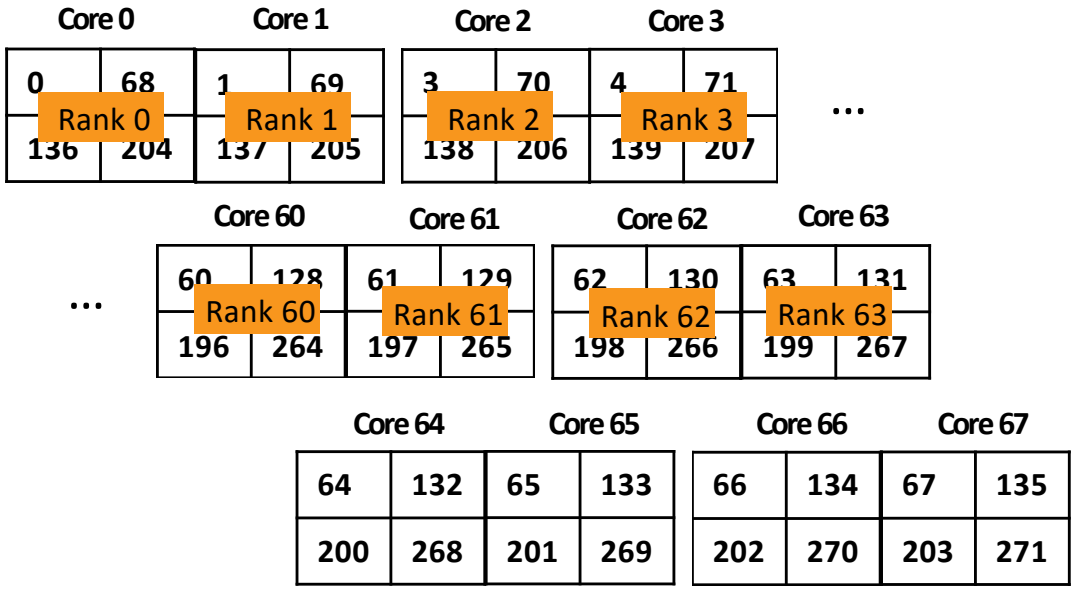
The hybrid VASP will run with 8 threads per MPI task using 64 cores out of available 68 cores. The srun command line options, –c32 --cpu_bind=cores, together with the two OpenMP environment variables set inside the vasp/20171017-knl module, OMP_PROC_BIND=spread and OMP_PLACES=threads, allows optimal process/thread affinity.



Sample job script to pure MPI VASP on Cori KNL nodes

```
#!/bin/bash -l
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 2:00:00
#SBATCH -C knl
#SBATCH –L SCRATCH

module load vasp

srun –n64 –c4 --cpu_bind=cores vasp_std
```

Process affinity outcome



This job script requests 1 KNL node in the quad,cache mode. The srun command launches 64 MPI tasks on the node, allocating 4 CPUs per task, and binds processes to cores. The resulting task placement is shown in the right figure. The Rank 0 will be pinned to Core0, Rank1 to Core1, …, Rank63 will be pinned to Core63. Each MPI task may move within the 4 CPUs in the cores.

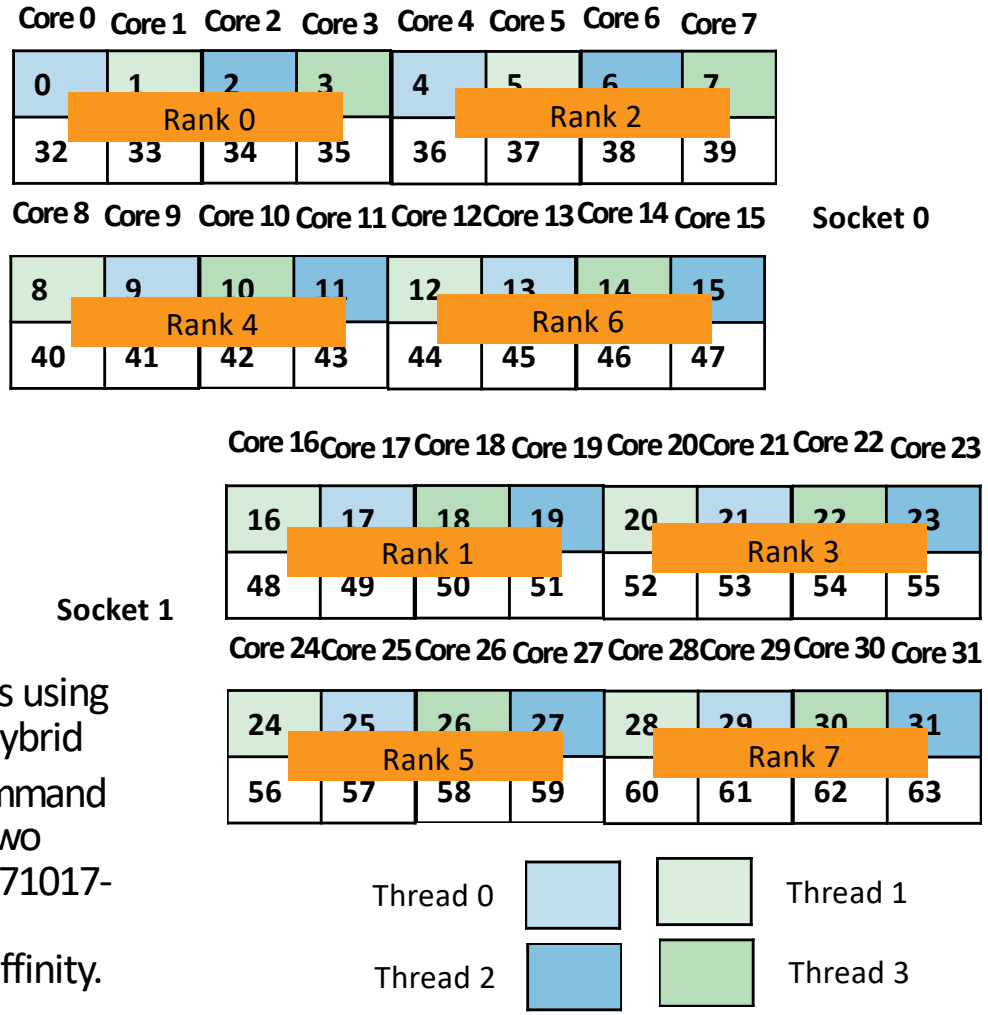
Each 2x2 box above is a core with 4 CPUs (hardware threads). The numbers shown in each CPU box is the CPU ids. The last 4 cores are not used in this example. The cores 4-59 were not be shown.

Sample job script to run the hybrid MPI+OpenMP VASP with 4 threads per MPI task on Cori Haswell nodes

```
#!/bin/bash -l
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 2:00:00
#SBATCH -C haswell
#SBATCH –L SCRATCH

module load vasp/20171017-hsw
export OMP_NUM_THREADS=4

srun –n8 –c8 --cpu_bind=cores vasp_std
```

The above script requests 1 Haswell nodes for two hours using the regular QOS and using the scratch file system. The hybrid VASP will run with 4 threads per MPI task. The srun command line options, –c8 --cpu_bind=cores, together with the two OpenMP environment variables set inside the vasp/20171017-hsw module, OMP_PROC_BIND=spread and OMP_PLACES=threads, allows optimal process/thread affinity.

Process/thread affinity outcome



Sample job script to pure MPI VASP on Cori Haswell nodes

```
#!/bin/bash -l
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 2:00:00
#SBATCH -C hawell
#SBATCH –L SCRATCH

module load vasp

srun –n32 –c2 --cpu_bind=cores vasp_std
#or srun –n 32 vasp_std
```

This job script requests 1 haswell node for 2 hours. The srun command launches 32 MPI tasks on the node, allocating 2 CPUs per task, and binds processes to cores. The resulting task placement is shown in the right figure. The Rank 0 will be pinned to Core0, Rank1 to Core1, …, Rank31 will be pinned to Core31. Each MPI task may move within the 2 CPUs in the cores.

Process affinity outcome

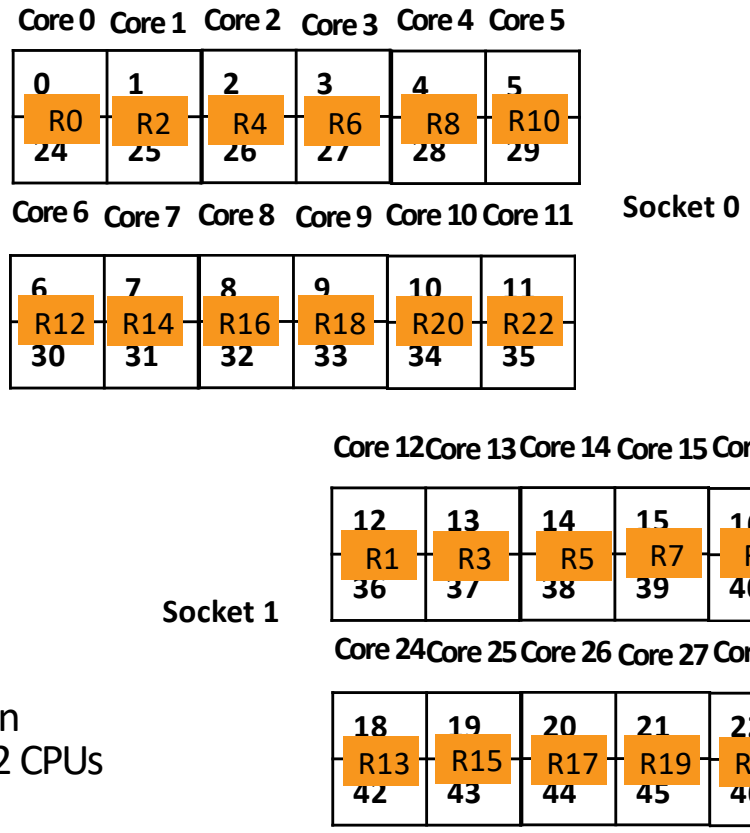
Each 2x1 box above is a core with 2 CPUs (hardware threads). The numbers shown in each CPU box is the CPU ids.

Sample job script to run pure MPI VASP on Edison

```
#!/bin/bash -l
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 2:00:00
#SBATCH –L SCRATCH

module load vasp

srun –n24 –c2 --cpu_bind=cores vasp_std
#or srun –n 24 vasp_std
```

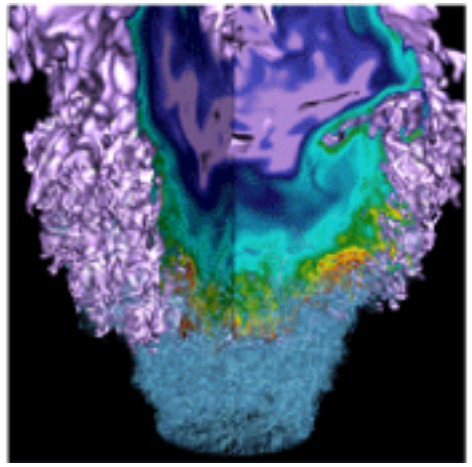
This job script requests 1 Edison node for 2 hours. The srun command launches 24 MPI tasks on the node, allocating 2 CPUs per task, and binds processes to cores. The resulting task placement is shown in the right figure. The Rank 0 will be pinned to Core0, Rank1 to Core12, …, Rank23 will be pinned to Core23. Each MPI task may move within the 2 CPUs in the cores.

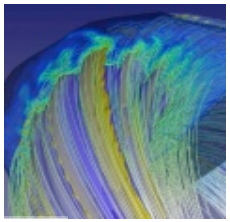
Process affinity outcome

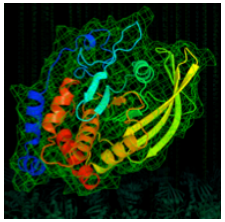


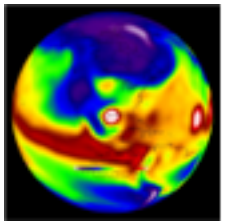
Each 2x1 box above is a core with 2 CPUs (hardware threads). The numbers shown in each CPU box is the CPU ids.

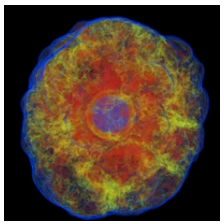
Performance of Hybrid VASP on Cori

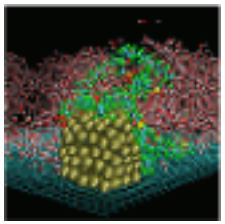










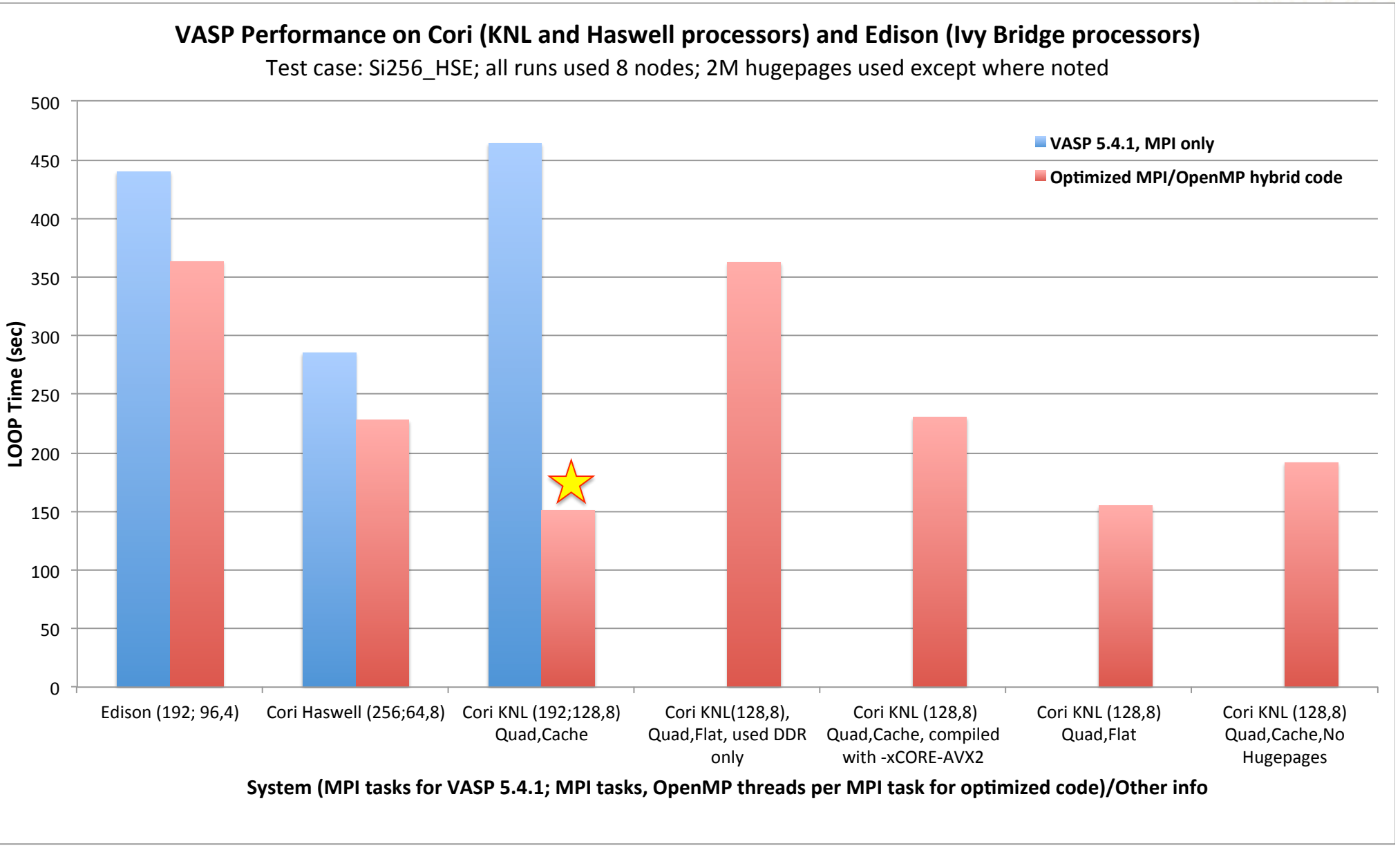


Selected 6 benchmarks cover representative VASP workloads, exercising different code paths, ionic constituent and problem sizes

	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

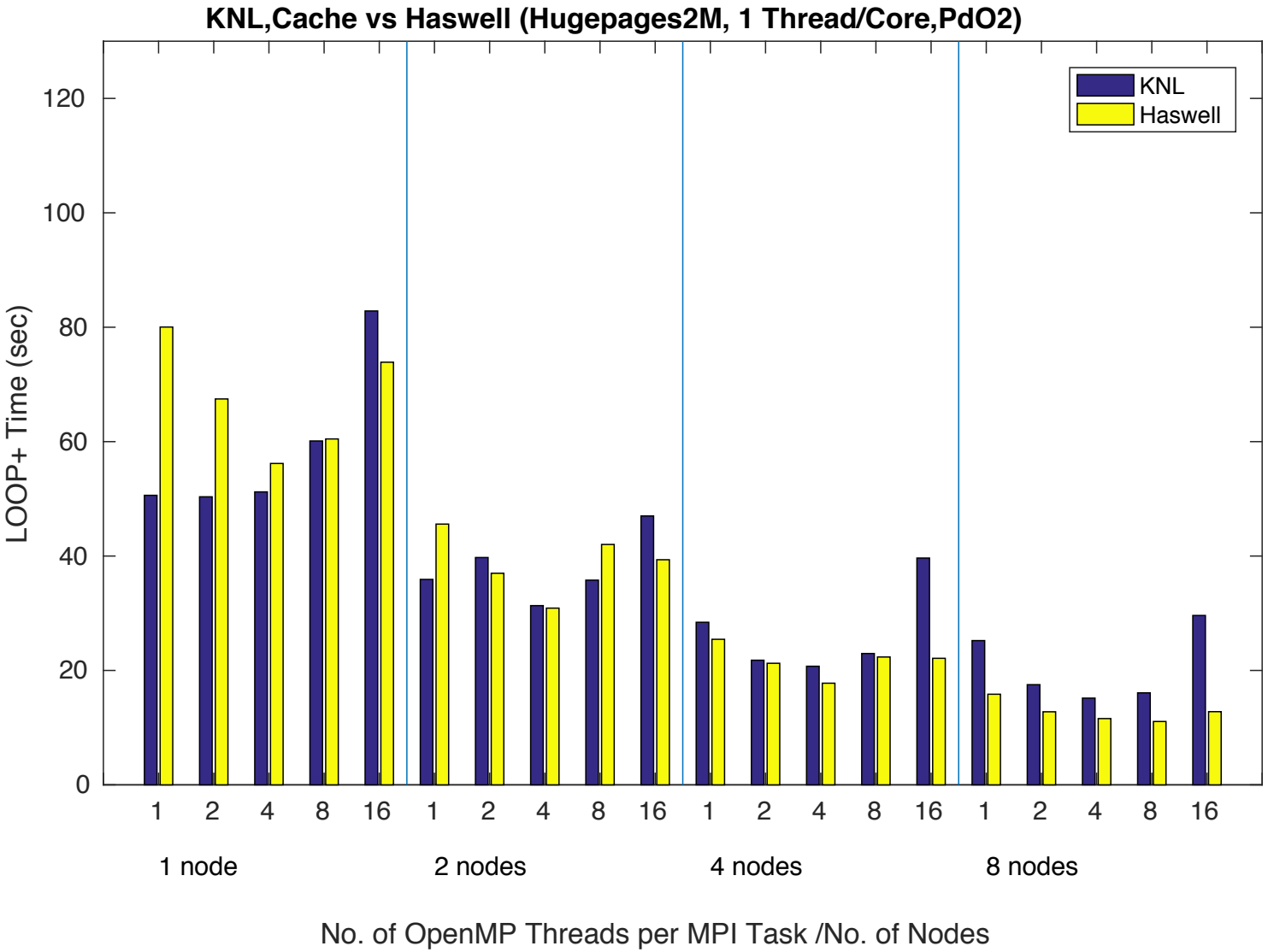
VASP versions, compilers and libraries used

- 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)
- Cori runs CLE 6.3 Update 4, and SLURM 2017.02.

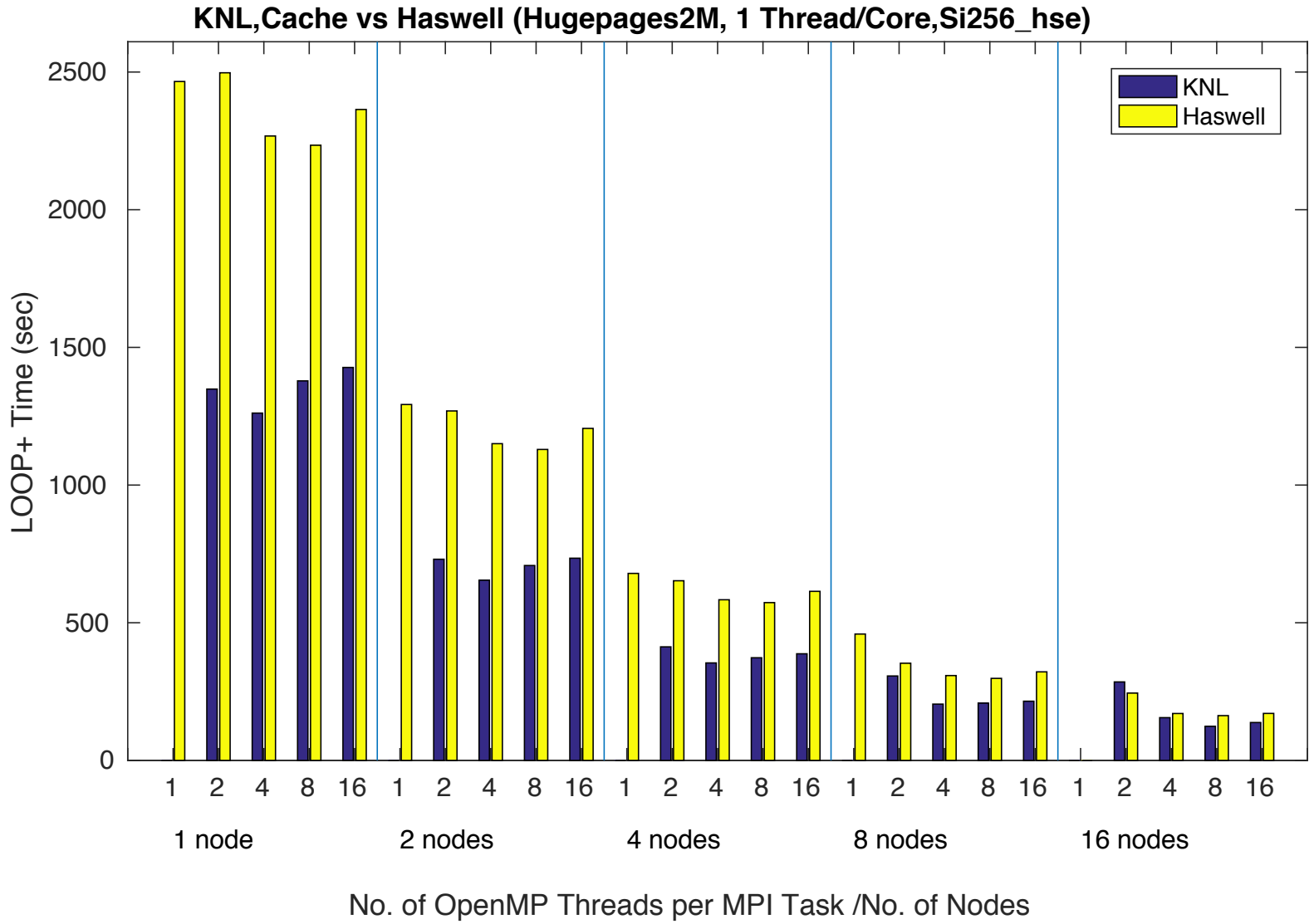


The optimized MPI/OpenMP hybrid code performs about 3x faster (HSE) than the MPI only code on Cori KNL nodes.

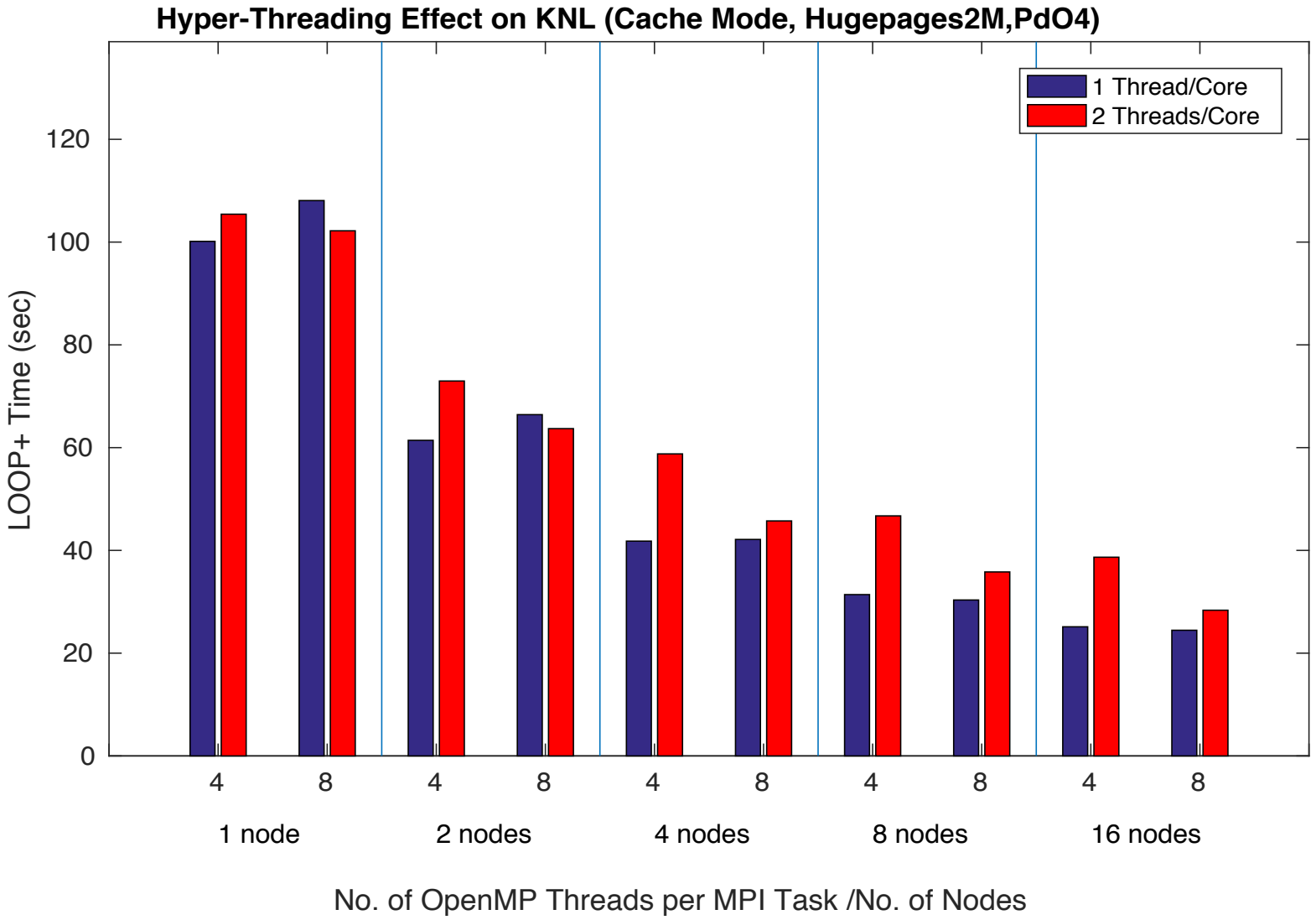
Hybrid VASP performs best with 4 or 8 OpenMP threads/task



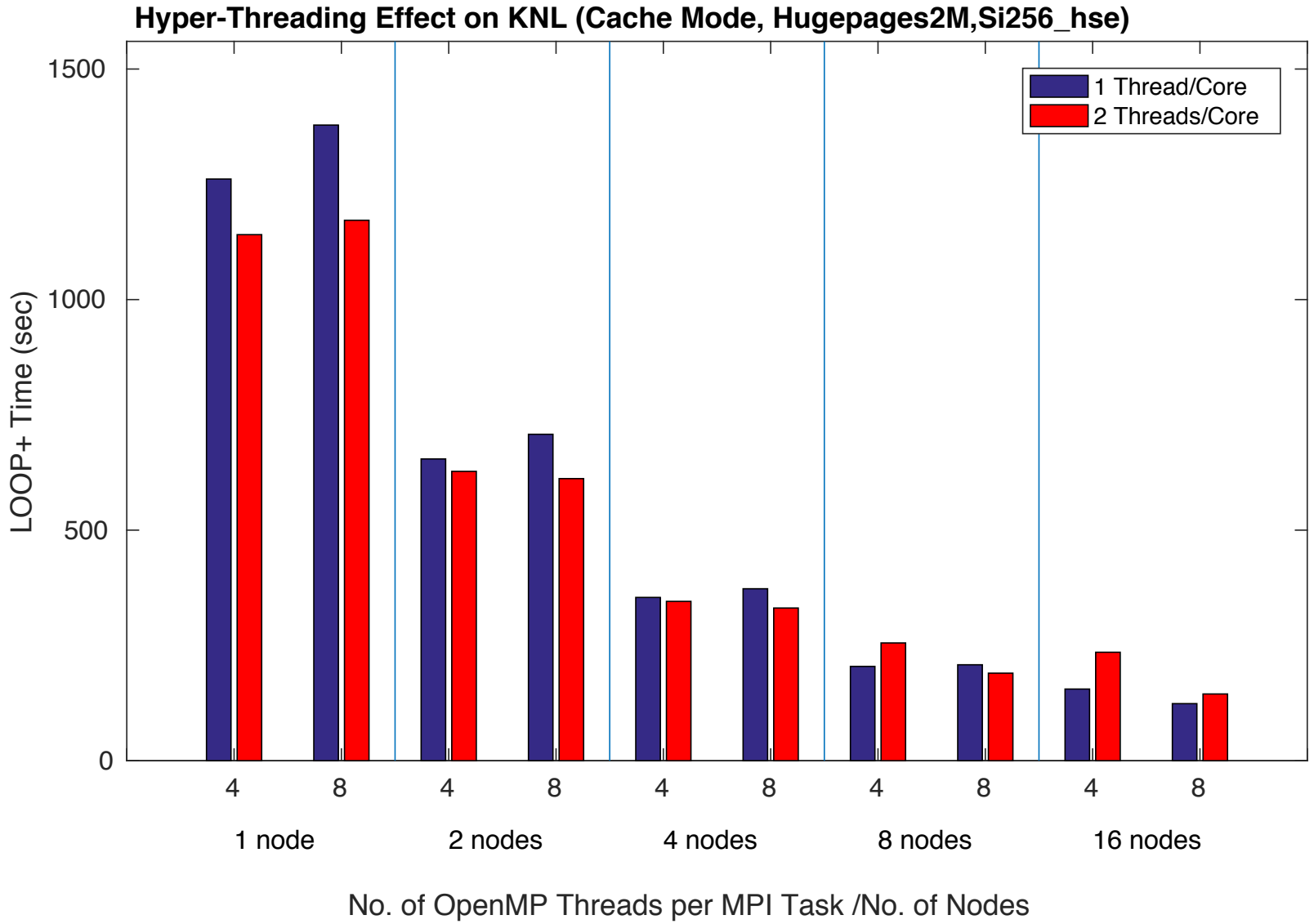
Hybrid VASP performs best with 4 or 8 OpenMP threads/task



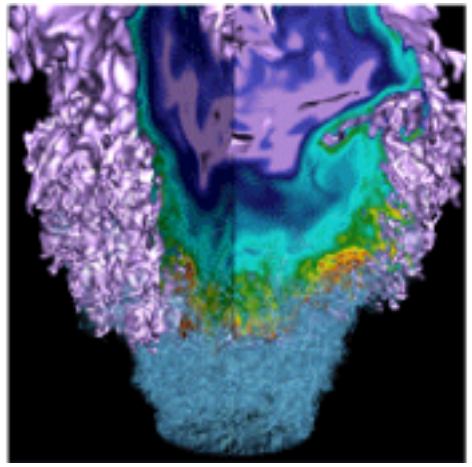
Hyper-Threading helps HSE workloads (arguably), but not other workloads in the parallel scaling regions on KNL

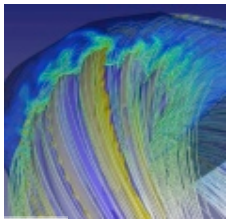


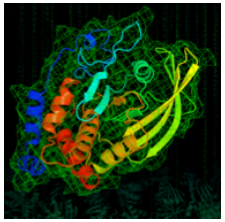
Hyper-Threading helps HSE workloads (arguably), but not other workloads in the parallel scaling regions on KNL

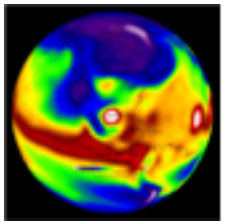


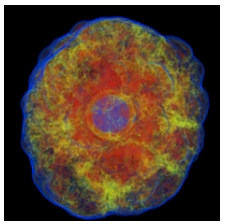
Best Practices

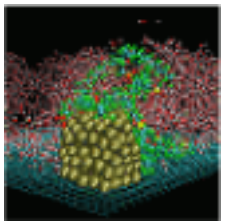












NERSC 40 YEARS at the FOREFRONT

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

Best practices

- On the KNL nodes, the hybrid MPI+OpenMP VASP is strongly recommended as it could outperform the pure MPI code by 2-3 times, depending on the workload (code paths).
- For the hybrid version, 4 or 8 OpenMP threads per MPI task is recommended.
- 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.
- Using 64 cores out of 68 available were recommended.

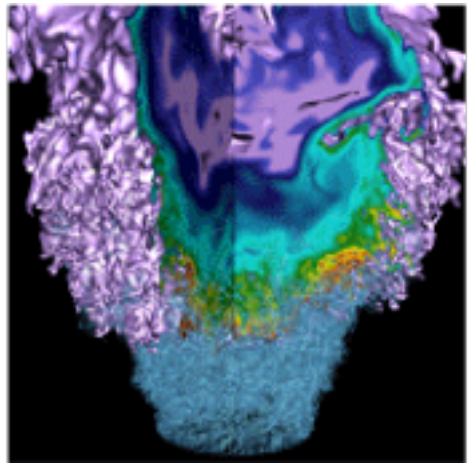
Best practices – continued

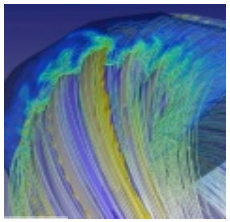
- The performance benefit from using MCDRAM is significant, so using more nodes to fit your data into the MCDARAM cache could be beneficial.
- A reference number when choosing the number of MPI tasks to use for a given system is 1/8 - 1/4 of the atoms in the system (assuming using eight threads/tasks) for a single k-point calculation.
- For pure MPI code, 1 core/atom is a good reference when selecting how many cores to use for your VASP jobs.

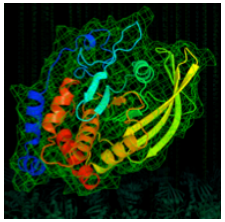
Best practices – continued

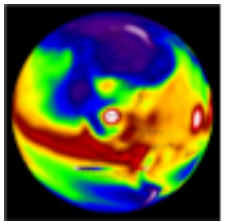
- **Use the K-point parallel when your systems have many k-points (KPAR) as long as the memory fits**
- **Use the gamma point only VASP binary when system contains only Gamma point.**

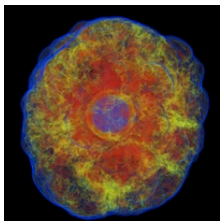
Compiling VASP by yourselves

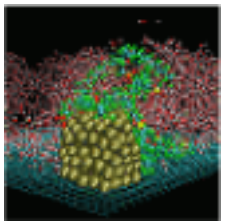












VASP makefiles are available in the VASP installation directories for the standard VASP distributions.

- **Makefiles are available in the VASP installation directories for the standard VASP distributions.**

```
swowner@cori02:~> module show vasp
-------------------------------------------------------------------
/usr/common/software/modulefiles/vasp/5.4.4-hsw:
…

setenv PSEUDOPOTENTIAL_DIR /global/common/sw/cray/cnl6/haswell/vasp/pseudopotentials
setenv VDW_KERNAL_DIR /global/common/sw/cray/cnl6/haswell/vasp/vdw_kernal
setenv NO_STOP_MESSAGE 1
setenv MPICH_NO_BUFFER_ALIAS_CHECK 1
prepend-path PATH /global/common/sw/cray/cnl6/haswell/vasp/vtstscripts/r933
prepend-path PATH /global/common/sw/cray/cnl6/haswell/vasp/5.4.4/intel/17.0.2.174/4bqi2il/bin
-------------------------------------------------------------------


swowner@cori02:~> ls -l /global/common/sw/cray/cnl6/haswell/vasp/5.4.4/intel/17.0.2.174/4bqi2il
total 3
drwxrwx--- 2 swowner vasp5    512 Feb 10 14:59 bin
-rw-r--r-- 1 swowner swowner 1687 Feb 10 14:31 makefile.include
swowner@cori02:~>

#move the makefile.include to your VASP source tree, e.g., vasp.5.4.4,
and then cd to that directory and type make all
```

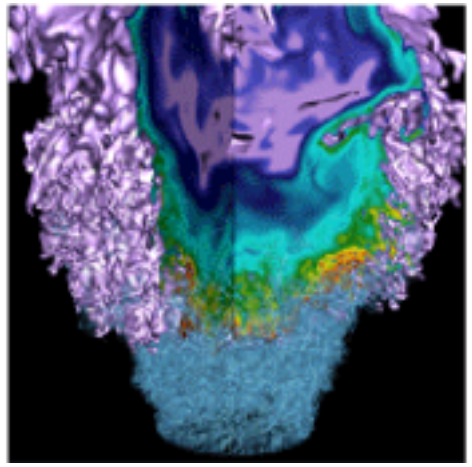
- **The source code of the hybrid MPI+OpenMP version may not be available to all users. If you need the makefile for the hybrid VASP, please let us know. (email to consult@nersc.gov)**

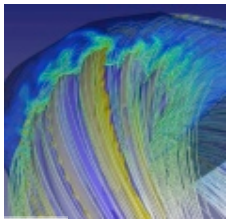
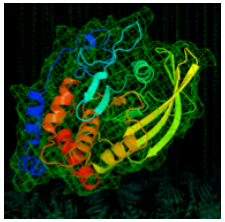
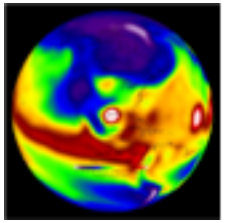
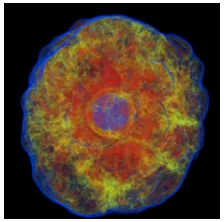
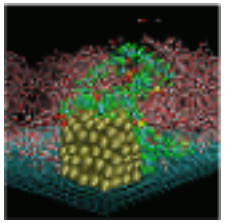
Acknowledgement

- **Martijn Marsman** (martijn.marsman@univie.ac.at)**, Florian Wende** (wende@zib.de)**, and Jeongnim Kim** (jeongnim.kim@intel.com)
- Steve Leak at NERSC

Thank you!

Parallelizations in VASP

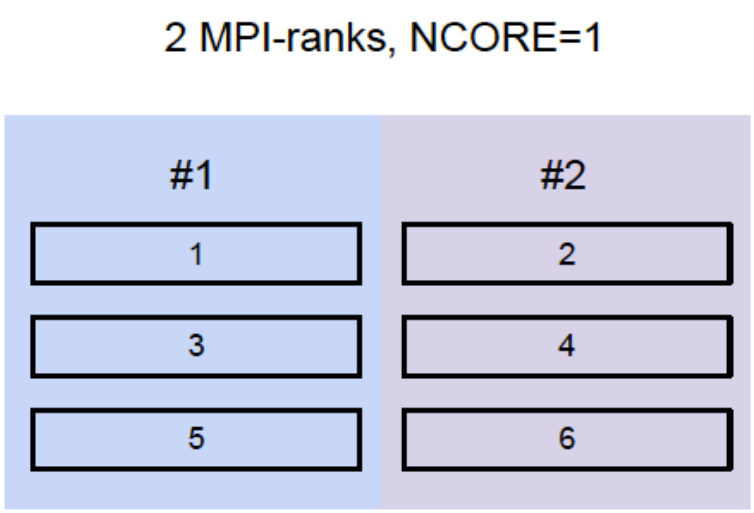


The following 5 slides are from Martijn Marshman's training slides,

http://www.vasp.at/vasp-workshop/lectures/VASP_lecture_HPC.pdf

Distribution of work and data



Distribute work and data “over-orbitals”

- Default
- NCORE = 1
  (or equivalently: NPAR = #-of-MPI-ranks)
- KPAR = 1

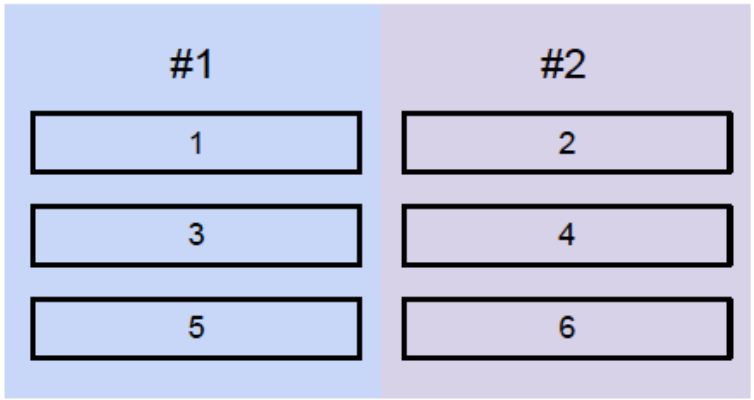
The Kohn-Sham equation:

$$\left(-\frac{1}{2}\Delta + V_{\mathrm{ext}}(\mathbf{r}) + V_{\mathrm{H}}(\mathbf{r}) + V_{\mathrm{xc}}(\mathbf{r})\right)\psi_{n\mathbf{k}}(\mathbf{r}) = \epsilon_{n\mathbf{k}}\psi_{n\mathbf{k}}(\mathbf{r})$$

- Orbital index *n*

Distribution of work and data

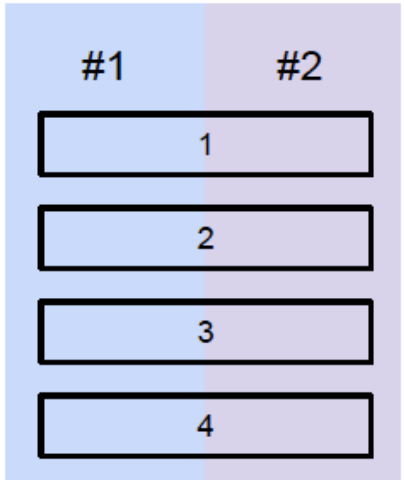
2 MPI-ranks, NCORE=1



Distribute work and data “over-orbitals”

- Default
- NCORE = 1
  (or equivalently: NPAR = #-of-MPI-ranks)
- KPAR = 1

2 MPI-ranks, NCORE=2

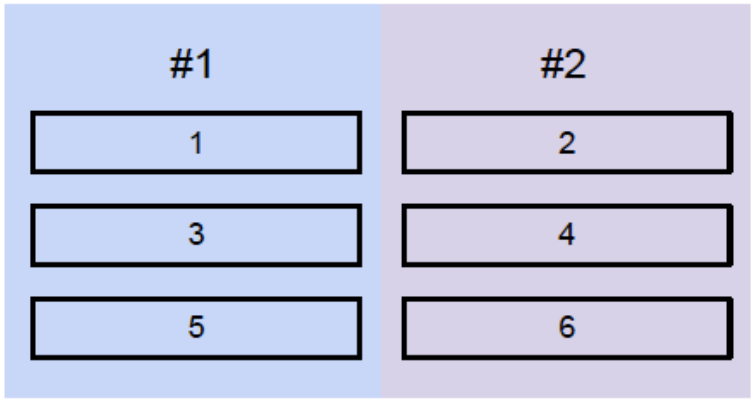


Distribution work and data “over-plane-waves”

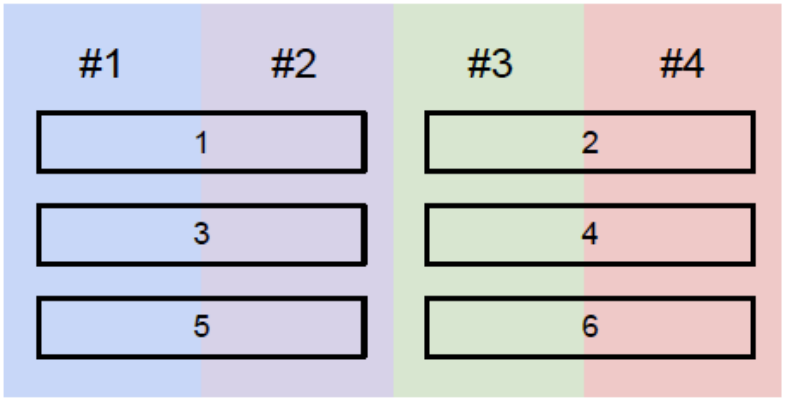
- NCORE = #-of-MPI-ranks
  (or equivalently: NPAR = 1)
- KPAR = 1

Distribution of work and data

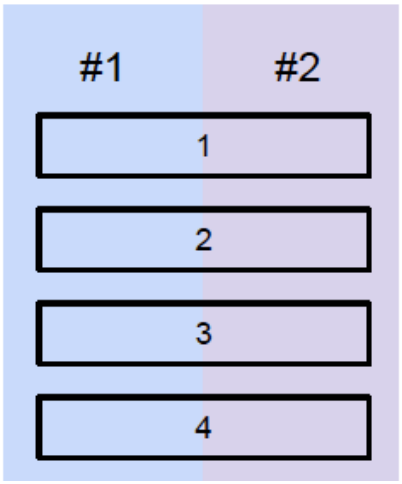
2 MPI-ranks, NCORE=1



4 MPI-ranks, NCORE=2



2 MPI-ranks, NCORE=2



Combinations of ”over-orbitals” and ”over-plane-wave” distributions are allowed as well

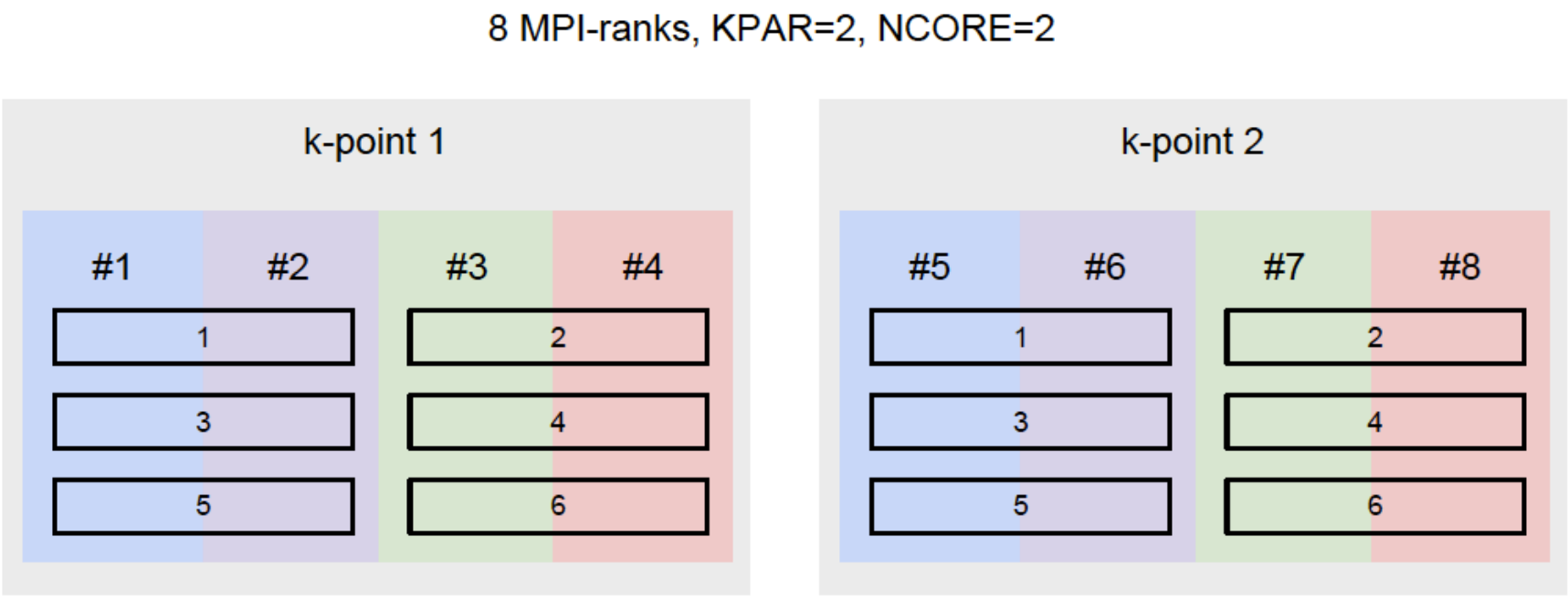
Distribution of work and data

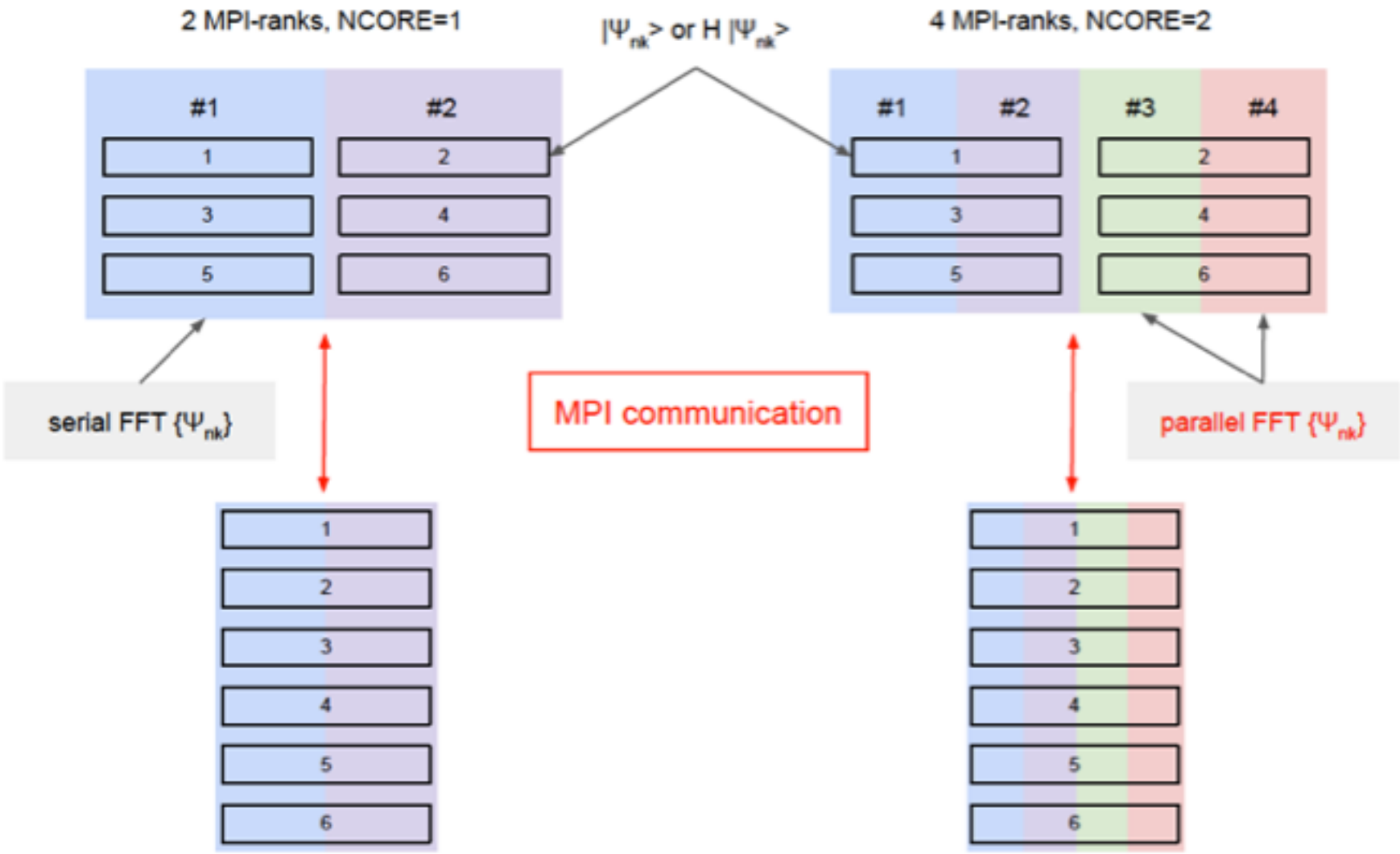
Additionally work may be distributed ”over-**k**-points”

- KPAR = *n* (*n*>1)
- *m* = (#-of-MPI-ranks / *n*) must be an integer
- Work is distributed in a round-robin fashion over groups of *m* MPI-ranks
- Data is duplicated!

$$\left(-\frac{1}{2}\Delta + V_{\mathrm{ext}}(\mathbf{r}) + V_{\mathrm{H}}(\mathbf{r}) + V_{\mathrm{xc}}(\mathbf{r})\right)\psi_{n\mathbf{k}}(\mathbf{r}) = \epsilon_{n\mathbf{k}}\psi_{n\mathbf{k}}(\mathbf{r})$$

- Orbital index *n*, k-point index **k**

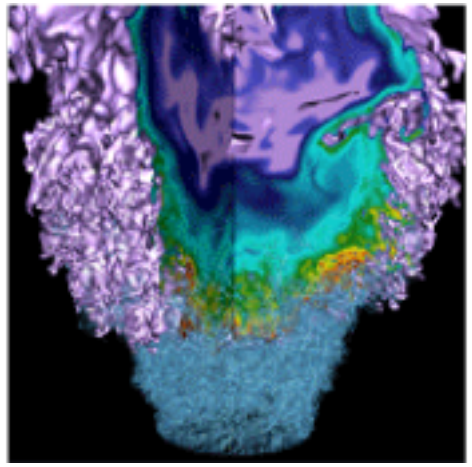


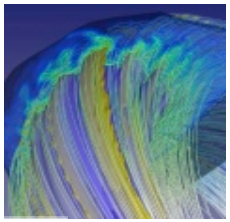


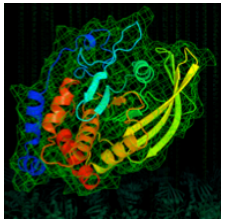
$C_{nm} = \langle \Psi_{mk} | H | \Psi_{nk} \rangle$

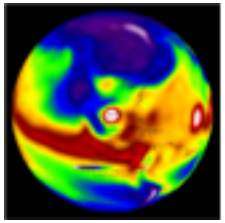
Each MPI-rank contracts over its subset of G-vector
followed by a global sum of C_{nm} over all MPI-ranks

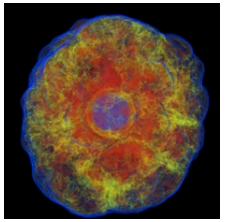
Process/Thread affinity

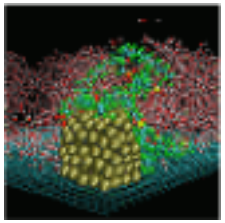












Using srun's --cpu_bind option and OpenMP environment variables to achieve desired process/thread affinity

- **Use srun --cpu_bind to bind tasks to CPUs**
  - Often needs to work with **the –c option of srun** to evenly spread MPI tasks on the CPUs on the nodes
  - **The srun –c <n> (or --cpus-per-task=n) allocates (reserves) n number of CPUs per task (process)**
  - --cpu_bind=[{verbose,quiet},]type, type: cores, threads, map_cpu:<list of CPUs>, mask_cpu:<list of masks>, none, …
- **Use OpenMP envs, OMP_PROC_BIND and OMP_PLACES to fine pin each thread to a subset of CPUs allocated to the host task**
  - Different compilers may have different default values for them. The following are recommended, which yield a more compatible thread affinity among Intel, GNU and Cray compilers:

    OMP_PROC_BIND=true # Specifying threads may not be moved between CPUs

    OMP_PLACES=threads # Specifying a thread should be placed in a single CPU
  - Use **OMP_DISPLAY_ENV=true** to display the OpenMP environment variables set (useful when checking the default compiler behavior)

Default Slurm behavior with respect to process/thread/memory binding

- By Slurm default, a decent CPU binding is set only when the MPI tasks per node x CPUs per task = the total number of CPUs allocated per node, e.g., 68x4=272
- Otherwise, Slurm does not do anything with CPU binding. The srun's --cpu_bind and –c options must be used explicitly to achieve optimal process/thread affinity.
- No default memory binding is set by Slurm. Processes can allocate memory from all NUMA nodes. The --mem_bind (or numactl) should be used explicitly to set memory bindings.

Default Slurm behavior with respect to process/thread/memory binding (continued)

- **The default distribution, the –m option of srun, is block:cyclic on Cori.**
  - The cyclic distribution method distributes allocated CPUs for binding to a given task consecutively from the same socket, and from the next consecutive socket for the next task, in a round-robin fashion across sockets.
- **The –m block:block also works. You are encouraged to experiment with –m block:block as some applications perform better with the block distribution.**
  - The block distribution method distributes allocated CPUs consecutively from the same socket for binding to tasks, before using the next consecutive socket.
- **The –m option is relevant to the KNL nodes when they are configured in the sub-NUMA cluster modes, e.g., SNC2, SNC4, etc. Slurm treats "NUMA nodes with CPUs" as "sockets", although KNL is a single socket node.**

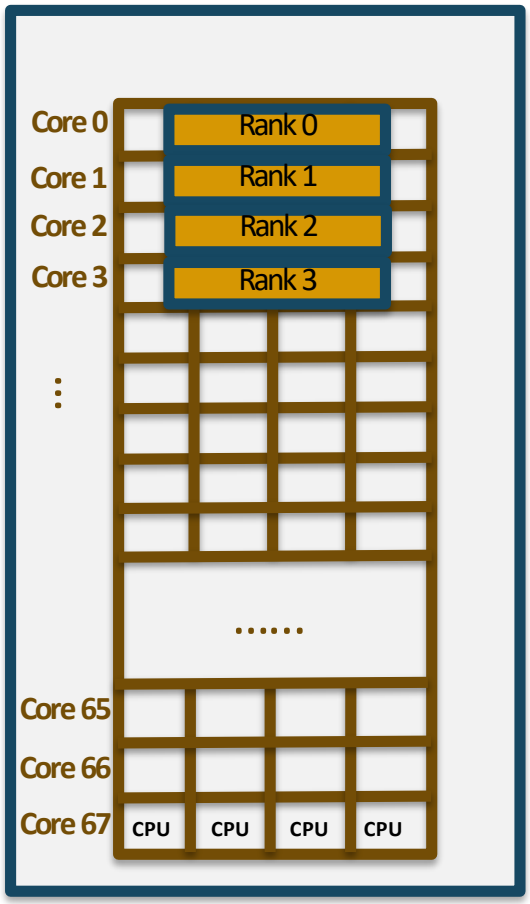
The --cpu_bind option of srun enables CPU bindings

salloc –N 1 –p debug –C knl,quad,flat

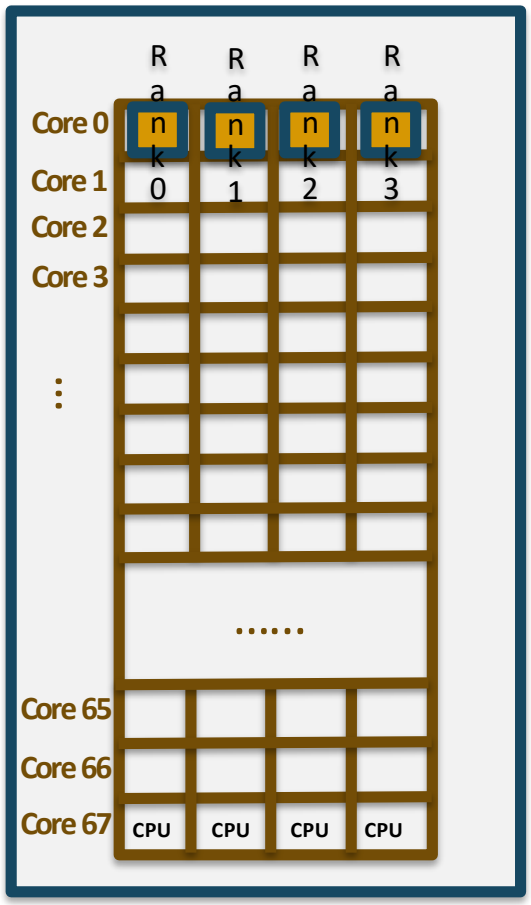
…

srun –n 4 ./a.out # no CPU bidings. Tasks can move around within 68 cores/272 CPUs

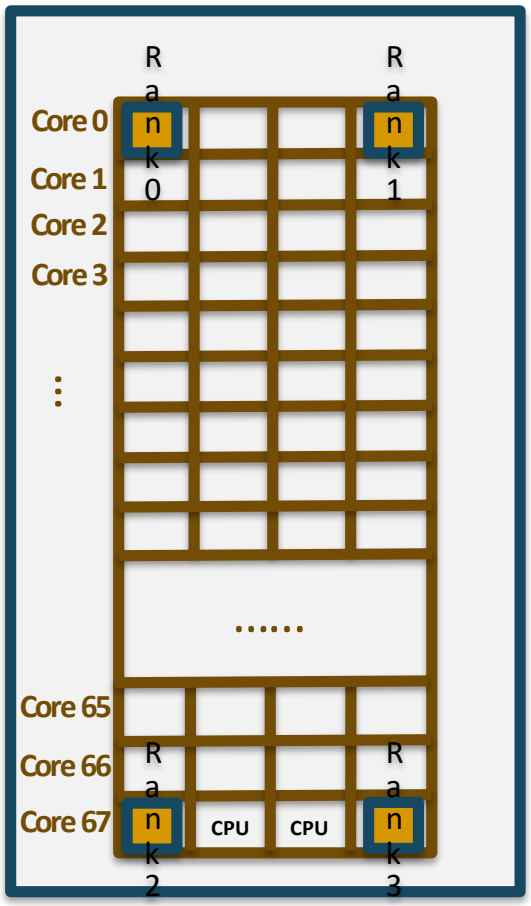
srun –n 4 –cpu_bind=
cores ./a.out



srun –n 4 --cpu_bind=
threads ./a.out



srun –n 4 --cpu_bind=
map_cpu:0,204,67,271 ./a.out

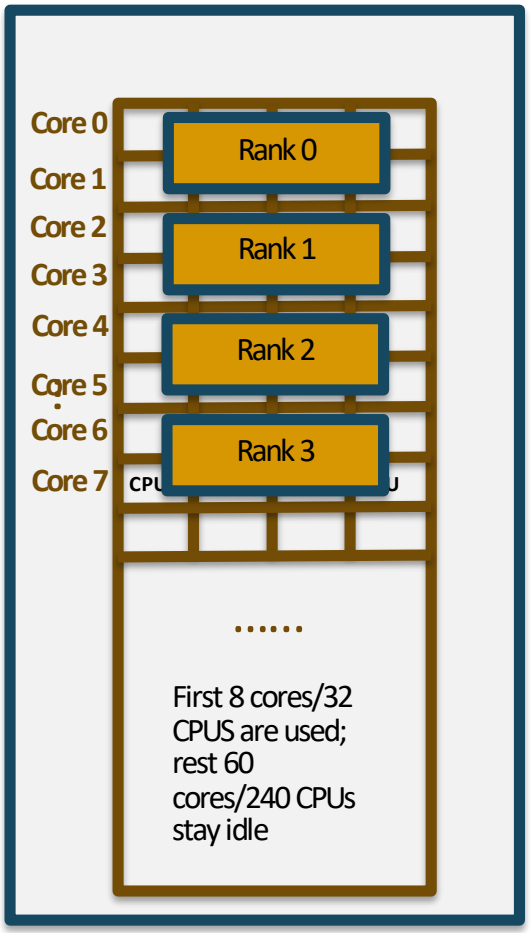


The --cpu_bind option: the -c option spreads tasks (evenly) on the CPUs on the node

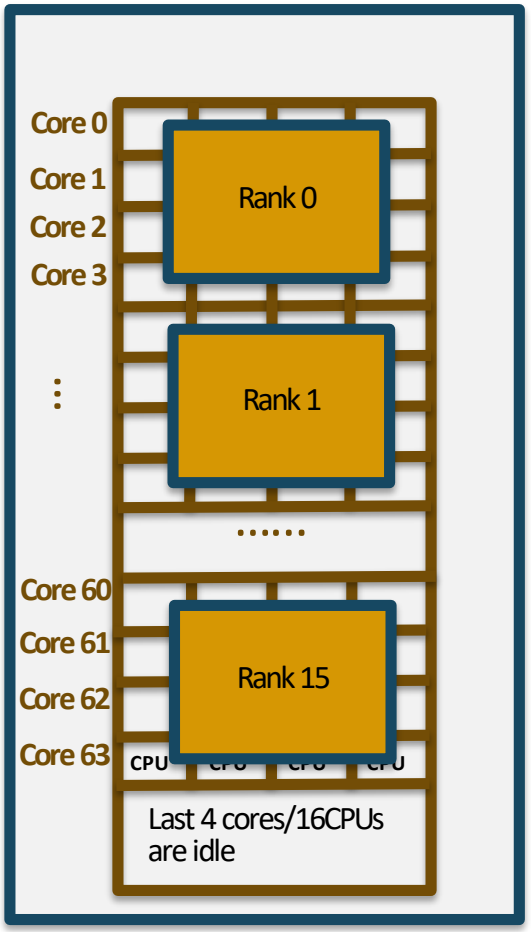
salloc –N 1 –p debug –C knl,quad,flat

…

srun –n 4 –c8 –cpu_bind=cores ./a.out



srun –n 16 –c16 –cpu_bind=cores ./a.out

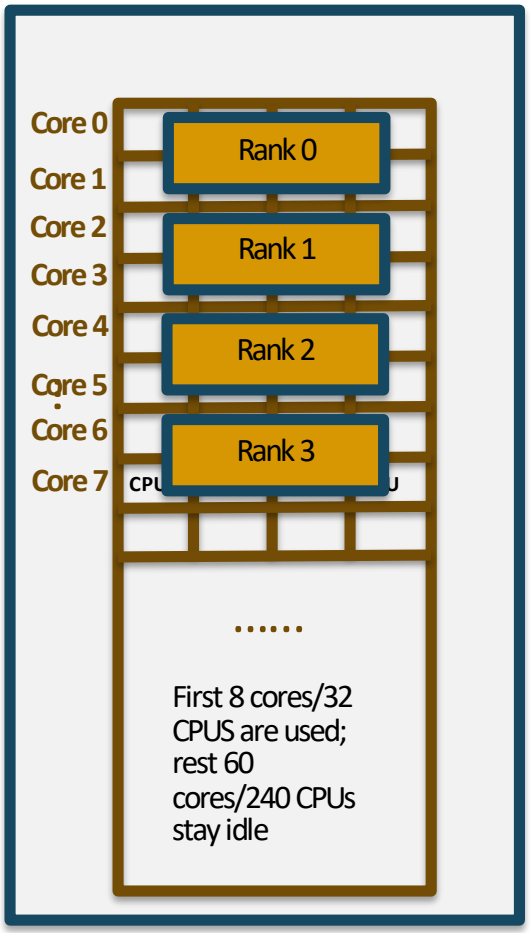


The --cpu_bind option (continued): the –c option spread tasks (evenly) on the CPUs on the node

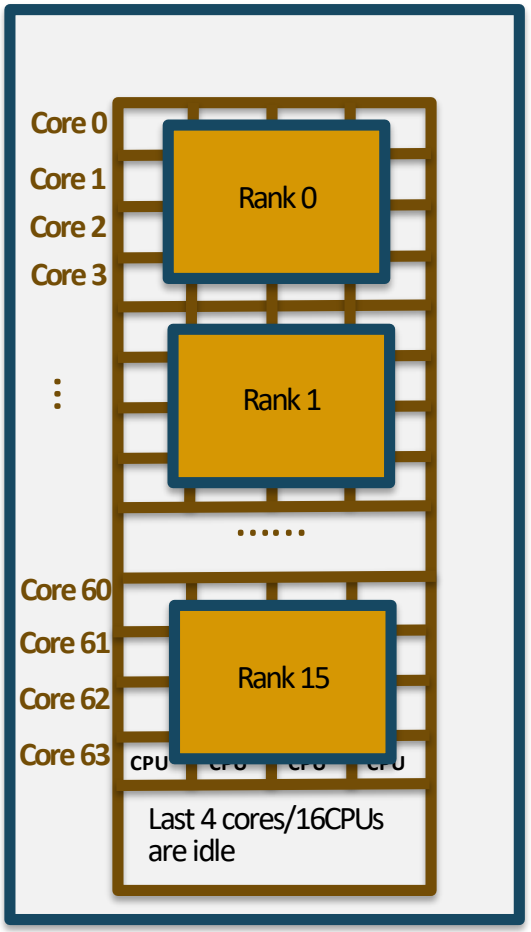
salloc –N 1 –p debug –C knl,quad,flat

…

srun –n 4 –c8 –cpu_bind=threads ./a.out



srun –n 16 –c16 –cpu_bind=threads ./a.out

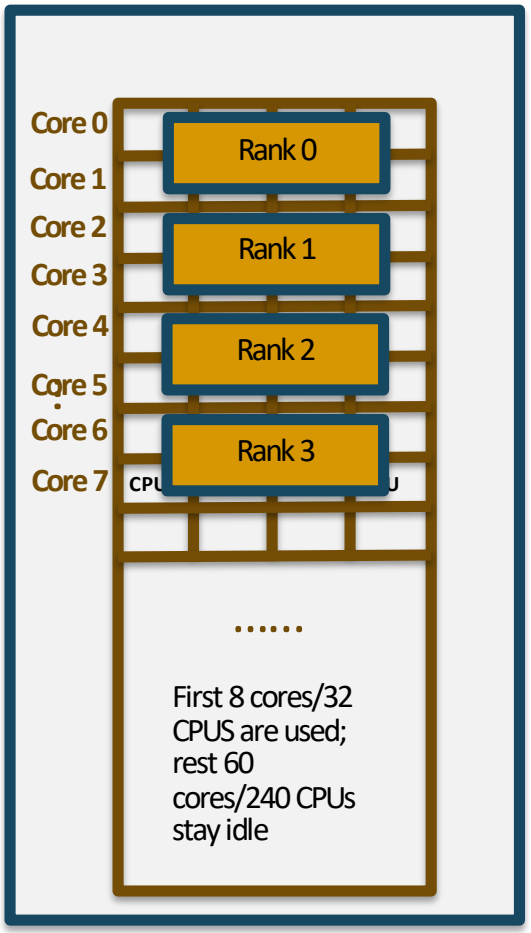


The -c option: --cpu_bind=cores vs –cpu_bind=threads

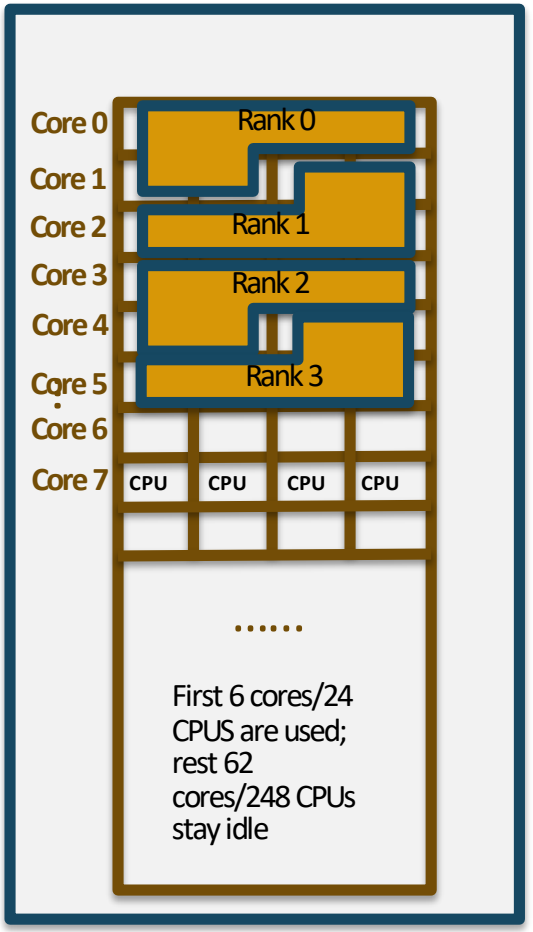
salloc –N 1 –p debug –C knl,quad,flat

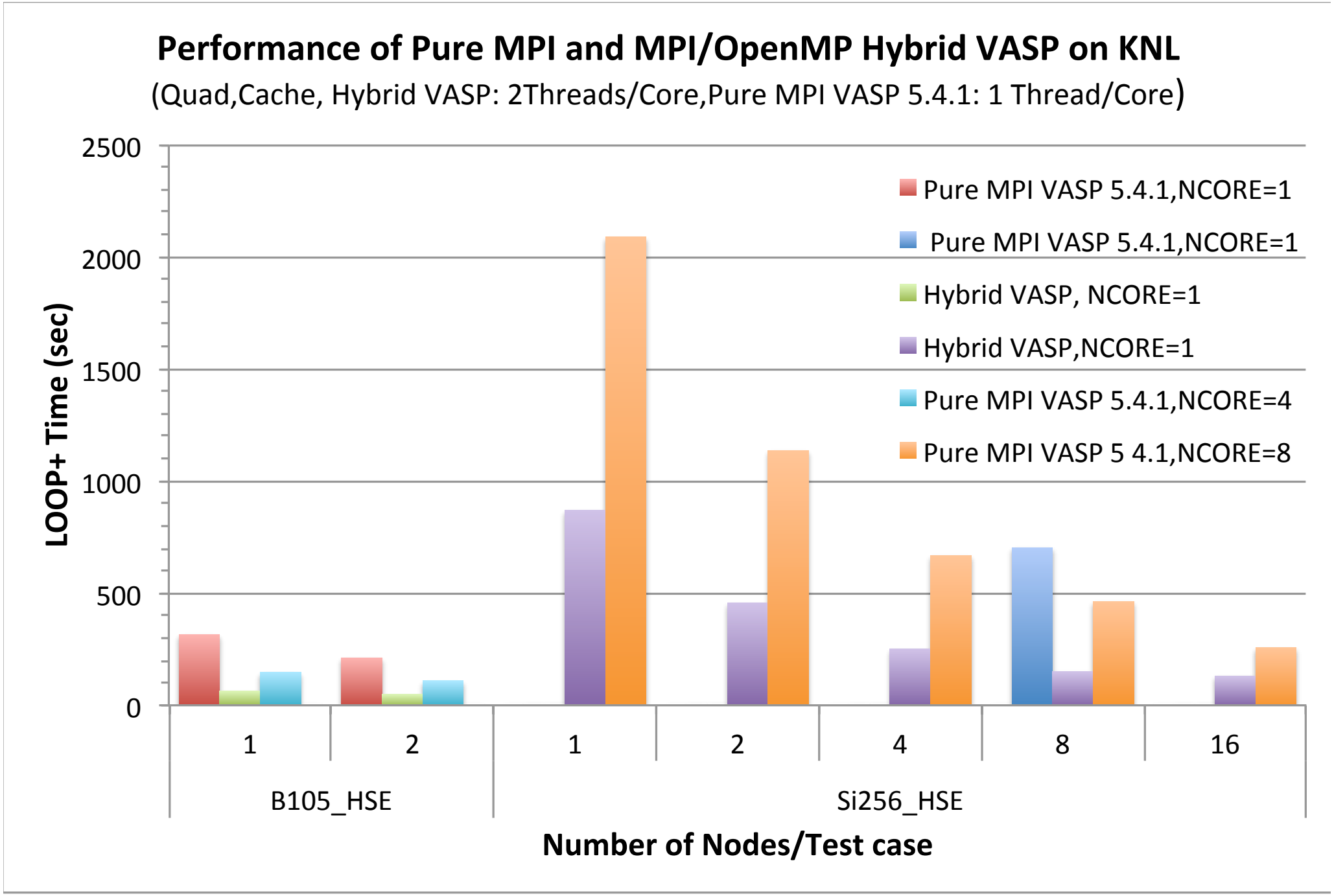
…

srun –n 4 –c 6 –cpu_bind=cores ./a.out



srun –n 4 –c 6 –cpu_bind=threads ./a.out





Hybrid VASP is about 2-3 times faster on Cori KNL nodes