# Using VASP at NERSC

May 14, 2025

Phillip Thomas
NERSC Application Performance Group

# Outline

- Getting started with VASP at NERSC
- Running VASP on Perlmutter
- Troubleshooting VASP
- VASP performance on Perlmutter
- Some VASP sample jobs
- Compiling VASP for Perlmutter
- Summary

Links (also in invite e-mail):
Q&A:
https://tinyurl.com/3byu44c8
Survey:
https://forms.gle/gdBKo2LM9E7hnFE98

# Getting Started with VASP at NERSC

# VASP Access at NERSC

- Confirm your VASP license to access the pre-compiled VASP binaries at NERSC
  - Instructions: https://docs.nersc.gov/applications/vasp/#access
  - Use VASP License Confirmation Request Form at: https://bit.ly/3Ca8kXc
- VASP access is controlled by unix groups
  - **`vasp5`** for all VASP 5
  - **`vasp6`** for VASP 6 through 6.4.3
  - Type the **`groups`** command to check your access
  - **Future**: vasp 6.5+ separate from earlier VASP 6
- No need to confirm licenses if you build your own

# Available VASP Modules on Perlmutter

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

| Build Type* | Pure MPI | MPI+OpenMP | MPI+OpenMP+OpenACC |
|---|---|---|---|
| Standard Distributions | vasp/5.4.4-cpu<br>(*default std*) | vasp/6.4.3-cpu<br>vasp/6.4.2-cpu | vasp/6.4.3-gpu<br>vasp/6.4.2-gpu |
| Builds with **third party codes**:<br>VTST, VASPSol | vasp-tpc/5.4.4-cpu<br>(*default tpc*) | vasp-tpc/6.4.2-cpu | vasp-tpc/6.4.2-gpu |

- All builds include precompiler-enabled plugins when possible[†]:
  Wannier90, DFTD4, LibXC, vdw-BEEF

*) **-cpu**: build for CPUs; **-gpu**: build for GPUs; **-tpc** stands for 'third party codes'
†) Some plugins work only in **cpu-only** builds or do not function with certain versions of VASP

# Using VASP Modules

- Type "`module show vasp/<version-str>`" to see what a module does

```
perlmutter:login34:~> module show vasp/6.4.3-gpu
-------------------------------------------------------------------------------------------------------------------
   /global/common/software/nersc/pe/modulefiles_hotfixes/vasp/6.4.3-gpu.lua:
-------------------------------------------------------------------------------------------------------------------
depends_on("gpu")
depends_on("nccl/2.18.3-cu12")
whatis("The Vienna Ab initio Simulation Package (VASP) is a computer program for atomic scale materials modelling.")
add_property("arch","gpu")
...
setenv("PSEUDOPOTENTIAL_DIR","/global/common/software/nersc9/vasp/dependencies/pseudopotentials")
setenv("VDW_KERNEL_DIR","/global/common/software/nersc9/vasp/dependencies/vdw_kernel")
setenv("MAKEFILE_INCLUDE_PATH","/global/common/software/nersc9/vasp/vasp/6.4.3-gpu")
setenv("NO_STOP_MESSAGE","1")
setenv("MPICH_NO_BUFFER_ALIAS_CHECK","1")
setenv("OMP_NUM_THREADS","1")
prepend_path("PATH","/global/common/software/nersc9/vasp/dependencies/vtstscripts-1033")
prepend_path("PATH","/global/common/software/nersc9/vasp/dependencies/bader")
prepend_path("PATH","/global/common/software/nersc9/vasp/vasp/6.4.3-gpu/bin")
```

path to **`makefile.include`** used to build this module

path to POTCAR files

some useful utilities

# Using VASP Modules (Cont.)

- Type "`ls -l <bin directory>`" to see available VASP binaries

```
perlmutter:login34:~> ls -l /global/common/software/nersc9/vasp/vasp/6.4.3-gpu/bin
total 152068
-rwxr-x--- 1 swowner vasp6 51307464 Oct  7  2024 vasp_gam
-rwxr-x--- 1 swowner vasp6 52229992 Oct  7  2024 vasp_ncl
-rwxr-x--- 1 swowner vasp6 52178704 Oct  7  2024 vasp_std
```

vasp_gam: the gamma point only version
vasp_std: the standard kpoint version
vasp_ncl: the non-collinear version

Increasing resource use

```
ERROR: noncollinear calculations require that VASP is compiled without the flag -DNGXhalf and -DNGZhalf
```

This error occurs when one attempts to run a noncollinear calculation using the `vasp_std` or `vasp_gam` executable. To solve, use the `vasp_ncl` executable instead.

- Type "`module load <vasp/version>`" to access VASP binaries

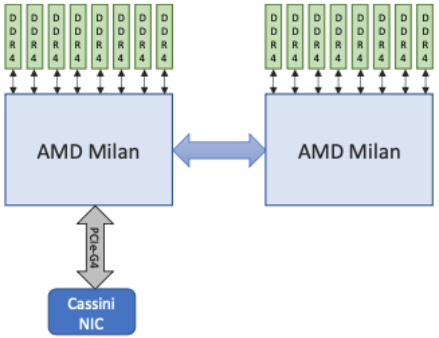
```
perlmutter:login34:~> module load vasp/6.4.3-gpu

perlmutter:login34:~> which vasp_std
/global/common/software/nersc9/vasp/vasp/6.4.3-gpu/bin/vasp_std
```

# Running VASP on Perlmutter

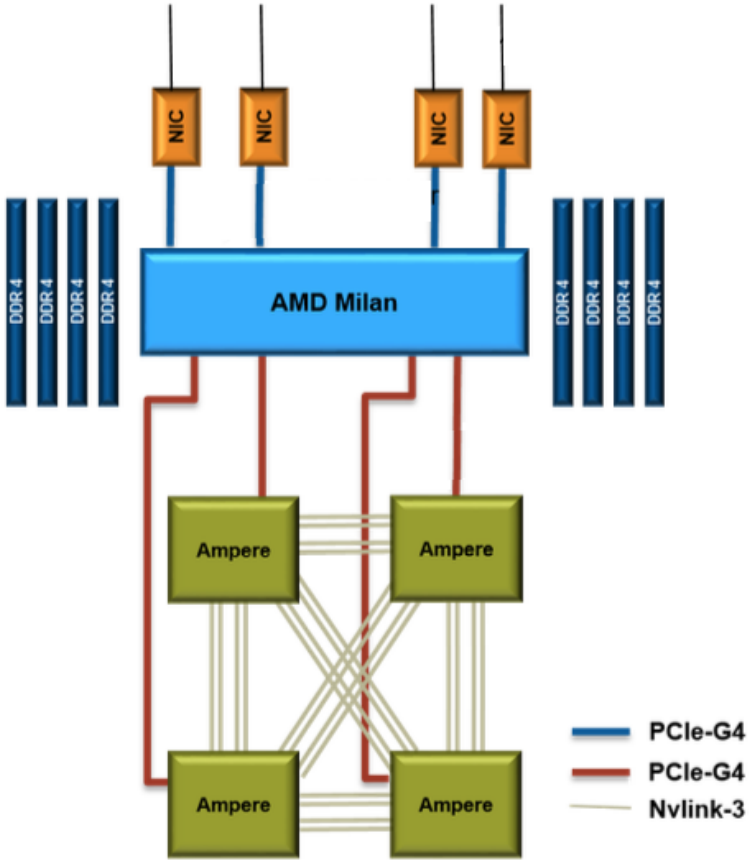
# Perlmutter System Configuration

| **System** | | **# of cores/node; # of gpus/node** | **# of CPUs per core** | **# of sockets per node** | **# of NUMA domains** | **Base Clock Speed (GHz)** | **Memory** per node**; per GPU (GB)** |
|---|---|---|---|---|---|---|---|
| GPU accelerated nodes | GPU (A100) (1536 nodes) | 64<br>4 | 2 | 1 | 4 | 2.45 | 256<br>**40** |
| | GPU (A100) (256 nodes) | 64<br>4 | 2 | 1 | 4 | 2.45 | 256<br>**80** |
| CPU only nodes | CPU (Milan) (3072 nodes) | 128 | 2 | 2 | 8 | 2.45 | 512 |
| GPU login nodes | GPU (A100) (40 nodes) | 128<br>1 | 2 | 2 | 2 | 2.45 | 512<br>**40** |

# Compute Nodes




AMD EPYC 7763 (Milan)




NVIDIA A100 GPU

# Perlmutter Queues

- **regular** QOS – for routine use
  - Max nodes entire system, max walltime is now 48 hours
- **interactive** QOS – for real-time testing
  - Can use up to 4 nodes for 4 hours; run limit is 2
  - Starts jobs immediately or cancels in 5 minutes
- **shared** QOS – for jobs using < 1 node
- **preempt** QOS – for long, interruptible jobs
- Two job aging policy:
  - Only two jobs per QOS per user accrue priority in the queue
- Note: Perlmutter queue limits are subject to change

| QOS | Max nodes | Max time (hrs) | Min time (hrs) | Submit limit | Run limit | Priority |
|---|---|---|---|---|---|---|
| regular | - | 48 | - | 5000 | - | medium |
| interactive[1] | 4 | 4 | - | 2 | 2 | high |
| shared_interactive | 0.5 | 4 | - | 2 | 2 | high |
| jupyter | 4 | 6 | - | 1 | 1 | high |
| debug | 8 | 0.5 | - | 5 | 2 | medium |
| shared[2] | 0.5 | 48 | - | 5000 | - | medium |
| preempt[3] | 128 | 48 (preemptible after two hours) | 2 | 5000 | - | medium |
| debug_preempt | 2 | 0.5 (preemptible after five minutes) | - | 5 | 2 | medium |
| premium | - | 48 | - | 5 | - | high |
| overrun | - | 48 (preemptible after two hours) | - | 5000 | - | very low |
| shared_overrun | 0.5 | 48 (preemptible after two hours) | - | 5000 | - | very low |
| realtime | custom | custom | custom | custom | custom | very high |

# Running VASP Interactively on Perlmutter Nodes

- The **`interactive`** QOS allows quick access to compute nodes
  - use **`salloc`** command to request interactive session
  - use **`srun`** to launch run from inside session

```
pthomas@perlmutter:login31:/global/cfs/cdirs/nstaff/pthomas/run> salloc -N 2 -q interactive -C cpu -t 30:00
salloc: Granted job allocation 38170327
salloc: Waiting for resource configuration
salloc: Nodes nid[004254-004255] are ready for job

pthomas@nid004254:~> module load vasp/6.4.3-cpu
pthomas@nid004254:~> export OMP_NUM_THREADS=8
pthomas@nid004254:~> srun -u -n 32 -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   32 mpi-ranks, with    8 threads/rank
...
```

# Sample Job Script to Run VASP 5 on CPU Nodes

- Start by creating a small script with the job configurations, e.g. **`runvasp.sh`**:

1 node

```
#!/bin/bash
#SBATCH –N 1
#SBATCH -C cpu
#SBATCH –q regular
#SBATCH –t 1:00:00

module load vasp/5.4.4-cpu

srun –n 128 -c 2 --cpu-bind=cores vasp_std
```

2 nodes

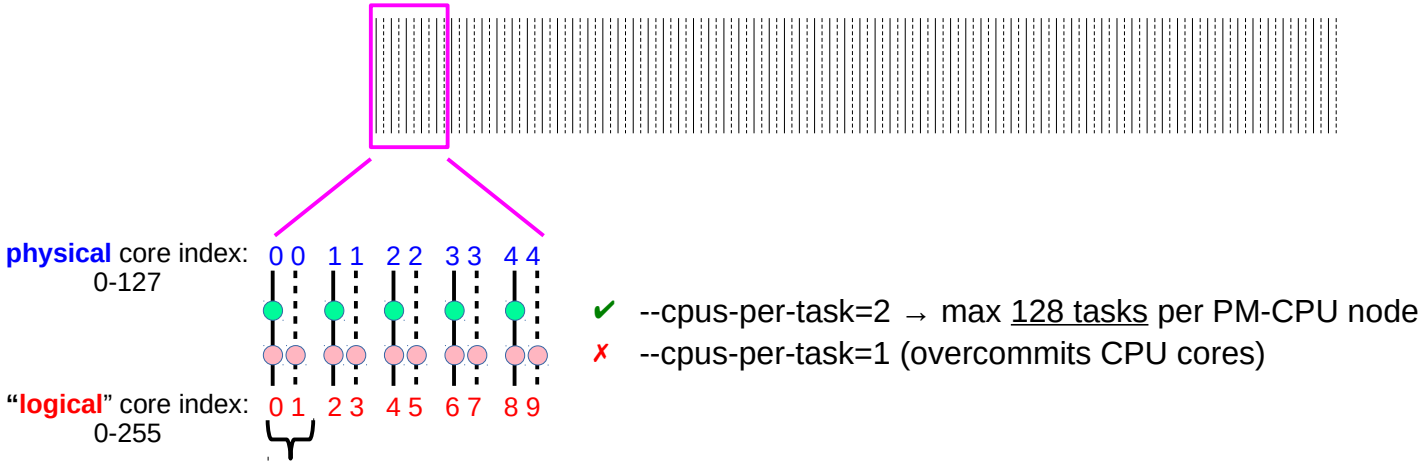
```
#!/bin/bash
#SBATCH –N 2
#SBATCH -C cpu
#SBATCH –q regular
#SBATCH –t 1:00:00

module load vasp/5.4.4-cpu

srun –n 256 –c 2 --cpu-bind=cores vasp_std
```

- Total MPI tasks indicated by '-n <ntasks>'
- The '-c <ncpus>' (--cpus-per-task=<ncpus>) parameter controls spacing of MPI tasks
- Use the --cpu-bind=cores option to bind tasks to CPUs
- Submit the batch script with the `sbatch` command e.g.,

  **`sbatch runvasp.sh`**

# Optimizing task spacing using --cpus-per-task

- The optimal MPI task spacing is 1 task per physical core




✔ --cpus-per-task=2 → max 128 tasks per PM-CPU node

✗ --cpus-per-task=1 (overcommits CPU cores)

# Sample Job Script to Run VASP 6 on CPU Nodes

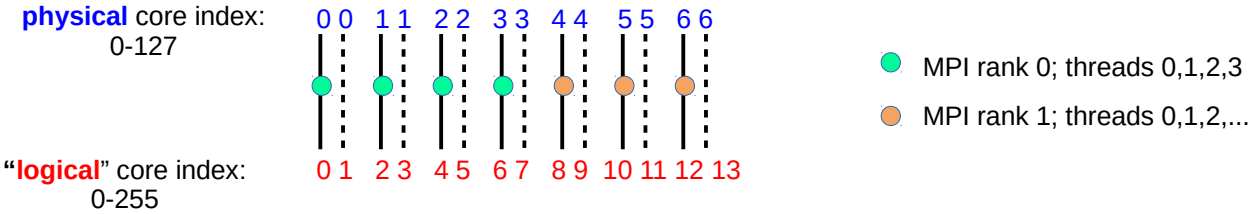
1 node

```
#!/bin/bash
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 1:00:00
#SBATCH -C cpu

module load vasp/6.4.3-cpu

export OMP_NUM_THREADS=4
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

# launch 1 task every 16 logical cores
srun –n 32 –c 8 --cpu-bind=cores vasp_std
```

2 nodes

```
#!/bin/bash
#SBATCH –N 2
#SBATCH –q regular
#SBATCH –t 1:00:00
#SBATCH -C cpu

module load vasp/6.4.3-gpu

export OMP_NUM_THREADS=4
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

# launch 1 task every 16 logical cores
srun –n 64 –c 8 --cpu-bind=cores vasp_std
```

- Each MPI rank spawns number of threads specified by OMP_NUM_THREADS
- Use larger spacing between MPI tasks to accommodate extra processes



# Sample Job Script to Run VASP 6 on **GPU** Nodes

```
#!/bin/bash
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 1:00:00
#SBATCH -C gpu

module load vasp/6.4.3-gpu

export OMP_NUM_THREADS=16
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun –n4 –c32 -G4 --cpu-bind=cores --gpu-bind=none vasp_std
```

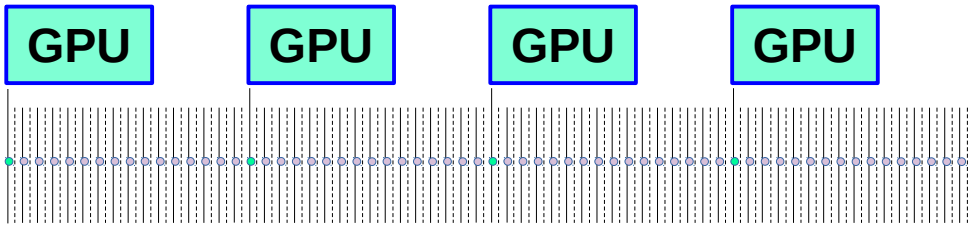
1 node

Ensures that all GPUs are visible to all MPI ranks

```
#!/bin/bash
#SBATCH –N 2
#SBATCH –q regular
#SBATCH –t 1:00:00
#SBATCH -C gpu

module load vasp/6.4.3-gpu

export OMP_NUM_THREADS=16
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun –n8 –c32 -G8 --cpu-bind=cores –gpu-bind=none vasp_std
```

2 nodes

- **Always** use 4 ranks per node with '**-c 32**' to select 1:1 MPI-rank-to-GPU mapping
- To use additional CPU cores, use up to **16** OpenMP threads per MPI rank



# NERSC Job Script Generator

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

CPU

Jobscript Generator

Job Information

This tool generates a batch script template which also realizes specific process and thread binding configurations.

**Machine**

Select the machine on which you want to submit your job.

Perlmutter - CPU

**Application Name**

Specify your application including the full path.

vasp_std

**Job Name**

Specify a name for your job.

test_vasp

**Email Address**

```
#!/bin/bash
#SBATCH -N 4
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J test_vasp
#SBATCH --mail-user=pthomas@lbl.gov
#SBATCH --mail-type=ALL
#SBATCH -t 01:00:00

#OpenMP settings:
export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

#run the application:
srun -n 64 -c 16 --cpu_bind=cores vasp_std
```

GPU

Jobscript Generator

Job Information

This tool generates a batch script template which also realizes specific process and thread binding configurations.

**Machine**

Select the machine on which you want to submit your job.

Perlmutter - GPU

**Application Name**

Specify your application including the full path.

vasp_std

**Job Name**

Specify a name for your job.

test_vasp

**Email Address**

Specify your email address to get notified when the job enters a certain state.

```
#!/bin/bash
#SBATCH -N 4
#SBATCH -C gpu
#SBATCH -G 16
#SBATCH -q regular
#SBATCH -J test_vasp
#SBATCH --mail-user=pthomas@lbl.gov
#SBATCH --mail-type=ALL
#SBATCH -t 01:00:00
#SBATCH -A m1234

#OpenMP settings:
export OMP_NUM_THREADS=1
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

#run the application:
#applications may perform better with --gpu-bind=none instead of --gpu-bind=single:1
srun -n 16 -c 32 --cpu_bind=cores -G 16 --gpu-bind=single:1 vasp_std
```

N.B. use **--gpu-bind=none**

VASP-specific samples at:
https://docs.nersc.gov/applications/vasp/#sample-job-scripts

# Bundle Jobs with Multiple Sruns

- Sample job script bundling multiple VASP jobs

```
#!/bin/bash
#SBATCH --qos=debug
#SBATCH --nodes=10
#SBATCH --time=30:00
#SBATCH --constraint=cpu

module load vasp/5.4.4-cpu

#Assume 5 VASP jobs in run1, run2, …, run5 directories
for j in {1..10}; do
   cd run$j
   srun -N 1 -n 128 -c 2 --cpu_bind=cores vasp_std &
   cd ..
done

wait
```

- The number of sruns in the job script should be small
  - Slurm is **not** good at handling multiple sruns in a single job script
  - Nodes with earlier completed jobs will be idle
- For many jobs, consider using a workflow tool such as Fireworks

# Bundle Jobs with Multiple Sruns + Job Resizing

**Sample job script to bundle VASP jobs with job resizing**

```
#!/bin/bash
#SBATCH -J job_resizing
#SBATCH -q debug
#SBATCH -N 10
#SBATCH -t 30:00
#SBATCH -C cpu

module load vasp/5.4.4-cpu
nodelist="`scontrol show hostname`"

#cd to each run directory, and launch srun on each node
for node in $nodelist ;do
   cd run$j
   ../run_jobstep.sh $node &
   cd ..
done

wait
```

```
cat run_jobstep.sh
#!/bin/bash
mynode=$1
srun -N1 -w $mynode -n128 -c2 --cpu-bind=cores -o %x-%J-${mynode}.out vasp_std

#release $mynode from nodelist if $mynode is not the batch host (head node) or it is not the last remaining node
batchHost=`scontrol show hostname |head -1`
if [[ $mynode != $batchHost ]]; then
   nodelist=`squeue -h -j $SLURM_JOB_ID -o %N`
   remaining_nodelist=`scontrol show hostname $nodelist |grep -v $mynode`
   if [[ -n $remaining_nodelist ]]; then
      scontrol update job $SLURM_JOB_ID Nodelist="`echo $remaining_nodelist`"
   fi
fi
```

- Release the nodes as soon as the job running on them completes while the rest of the jobs are still running
- You should not release the batch host (head node)
- Make the longest job run on the head node (the first job)
- You can modify this script for your needs, e.g., running each job instance on multiple nodes

# Running Long VASP Jobs

- Some VASP runs can be restarted from CONTCAR + {CHGCAR, WAVECAR}
- Use the **`preempt`** QOS to run at a discount after first 2 h!

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -C cpu
#SBATCH -J vasp_job
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH --qos=preempt
#SBATCH --comment=96:00:00
#SBATCH --time=2:00:00
#SBATCH --time-min=0:30:00
#SBATCH --signal=B:USR1@60
#SBATCH --requeue
#SBATCH --open-mode=append

if [ -f "STOPCAR" ]; then
rm STOPCAR
fi

module load vasp/5.4.4-cpu

srun -n 128 -c 2 --cpu_bind=cores vasp_std &
```

**checkpoint command**

***total* job length**

**job *segment* length**

**time needed to checkpoint**

```
ckpt_vasp() {
set -x
restarts=`squeue -h -O restartcnt -j $SLURM_JOB_ID`
echo checkpointing the ${restarts}-th job

restarts_num=$(echo $restarts | sed -e 's/^[ \t]*//')
echo "Restart number: ==${restarts_num}=="

echo LABORT = .TRUE. > STOPCAR
srun_pid=$(ps -fle | grep srun | head -1 | awk '{print $4}')
echo srun pid is $srun_pid
wait $srun_pid

folder="checkpt-$SLURM_JOB_ID-${restarts_num}"
mkdir $folder
echo "In directory $folder"

cp -p CONTCAR POSCAR
cp -p CONTCAR "$folder/POSCAR"
cp -p OUTCAR "$folder/OUTCAR-${restarts_num}"
cp -p OSZICAR "$folder/OSZICAR-${restarts_num}"
cp -p vasprun.xml "vasprun-${restarts_num}.xml"

set +x
}

ckpt_command=ckpt_vasp
max_timelimit=7200
ckpt_overhead=60
.  /global/common/sw/cray/cnl7/haswell/nersc_cr/19.10/etc/env_setup.sh
requeue_job func_trap USR1

wait
```

# A Few Useful Commands

- Commonly used Slurm commands:
  - `sbatch, salloc, scancel, srun, squeue, sinfo, sqs, scontrol, sacct` (check man page)
  - `sinfo --format='%F %b'` for available features of nodes, or `sinfo --format='%C %b'`
  - `scontrol show node <nid>` for node info

# Troubleshooting VASP

# Common Runtime Issues

- Most common issue: hanging due to bad CPU or GPU task binding
- Executable not found (for new users with unconfirmed licenses)
- Out of Memory
- Catastrophic failure (“segmentation fault”, “double linked list corrupted”, “PMPI_Allreduce: Invalid communicator”) due to solver crash
- VASP fails to read input files created with MS-DOS endline characters

- Intermittent issues with bad nodes or file system sometimes occur

# Troubleshooting Tips and Tricks

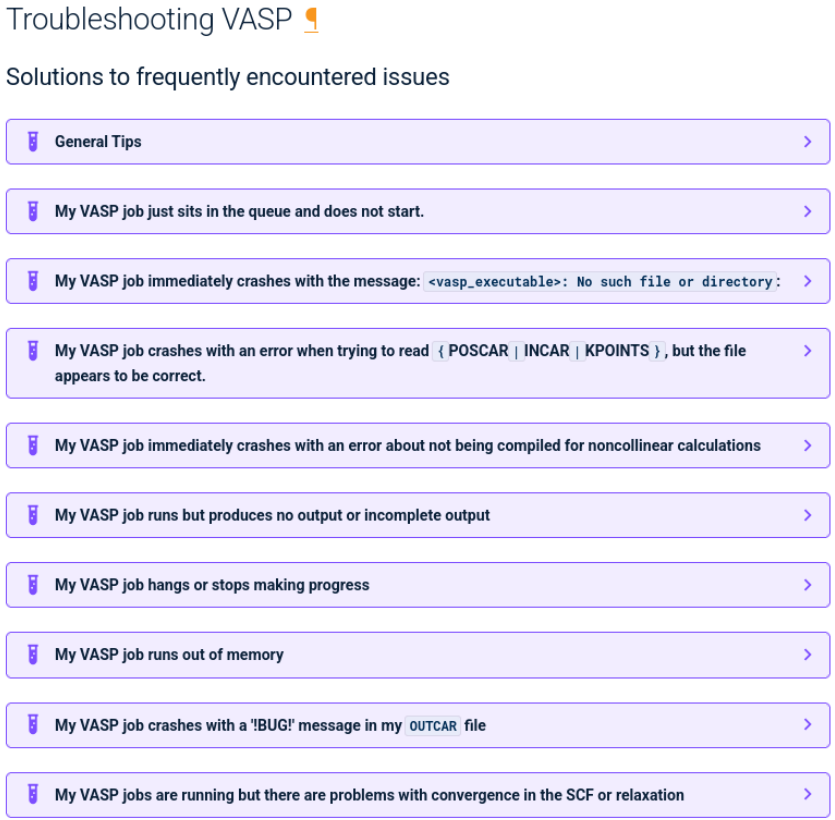
- Check job script, INCAR for mistakes
- Check OUTCAR, standard out, err
- stdbuf output=L releases message in output buffer
- Increase resources
- Run different version or {cpu, gpu} build
- **`ssh nid<xxxx>`** to log into node
  - **`top`** to check CPU activity
  - **`nvidia-smi`** to check GPU
- Repeat run on different node(s)

```
#!/bin/bash
#SBATCH -N 1
## SBATCH -N 2
#SBATCH -C cpu
#SBATCH -j vasp_job
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err

module load vasp/5.4.4-cpu
## module load vasp/6.4.3-cpu
## module load vasp/6.4.3-gpu

srun -n 128 -c 2 –cpu_bind=cores stdbuf --output=L vasp_std
## srun -n 256 -c 2 –cpu_bind=cores vasp_std
```

Check NERSC MOTD for possible outage:
https://www.nersc.gov/live-status/motd/

# Troubleshooting Resources

- Start with NERSC Troubleshooting Guide:
  https://docs.nersc.gov/applications/vasp/#troubleshooting-vasp
- VASP user forums:
  https://www.vasp.at/forum/
- File a support ticket:
  https://help.nersc.gov/
  - Describe issue in detail, including error message, which module, steps attempted to fix
  - Provide path-to-job, not attachment



# VASP Performance on Perlmutter

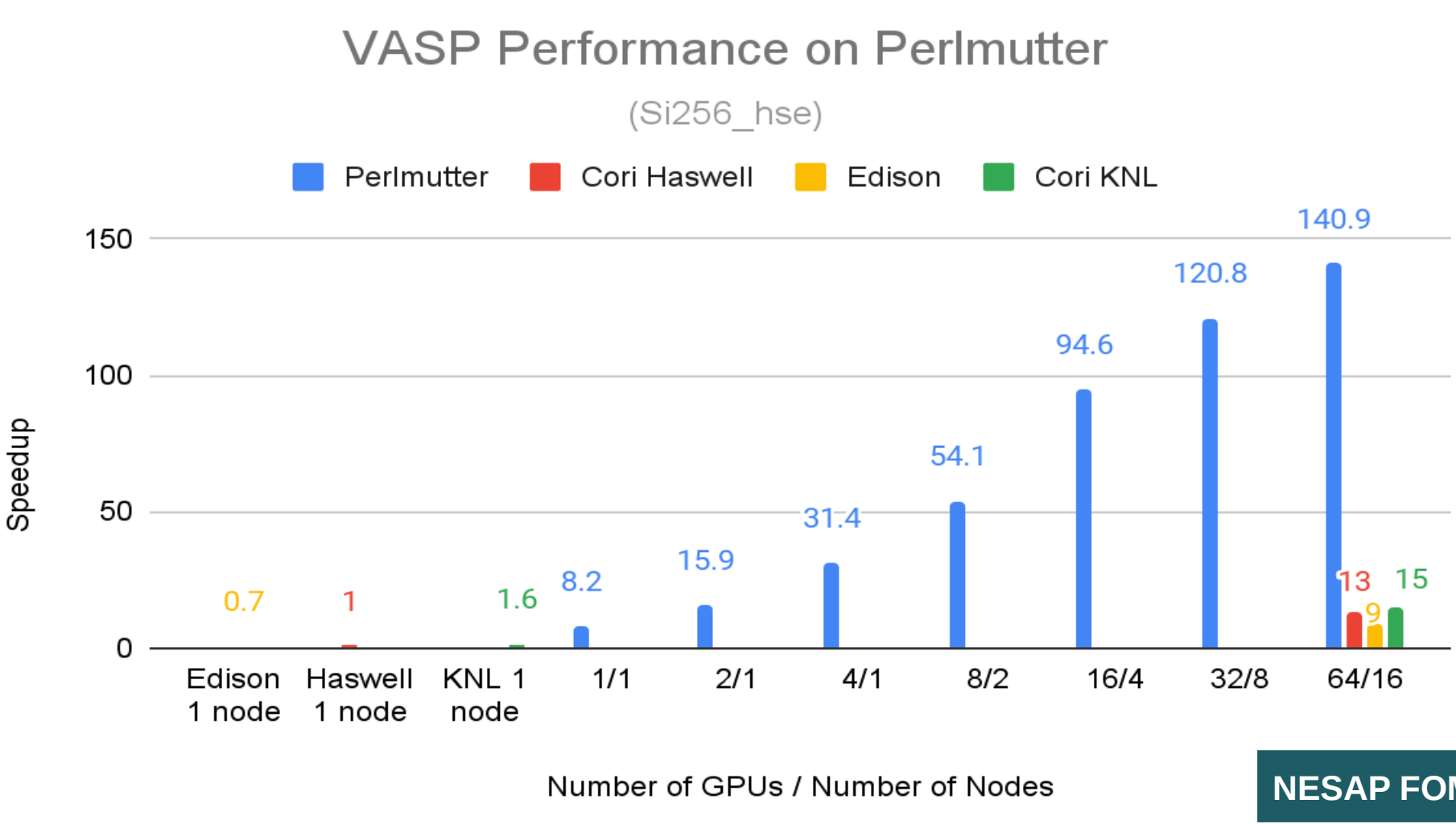
# VASP - #1 Production Code at NERSC

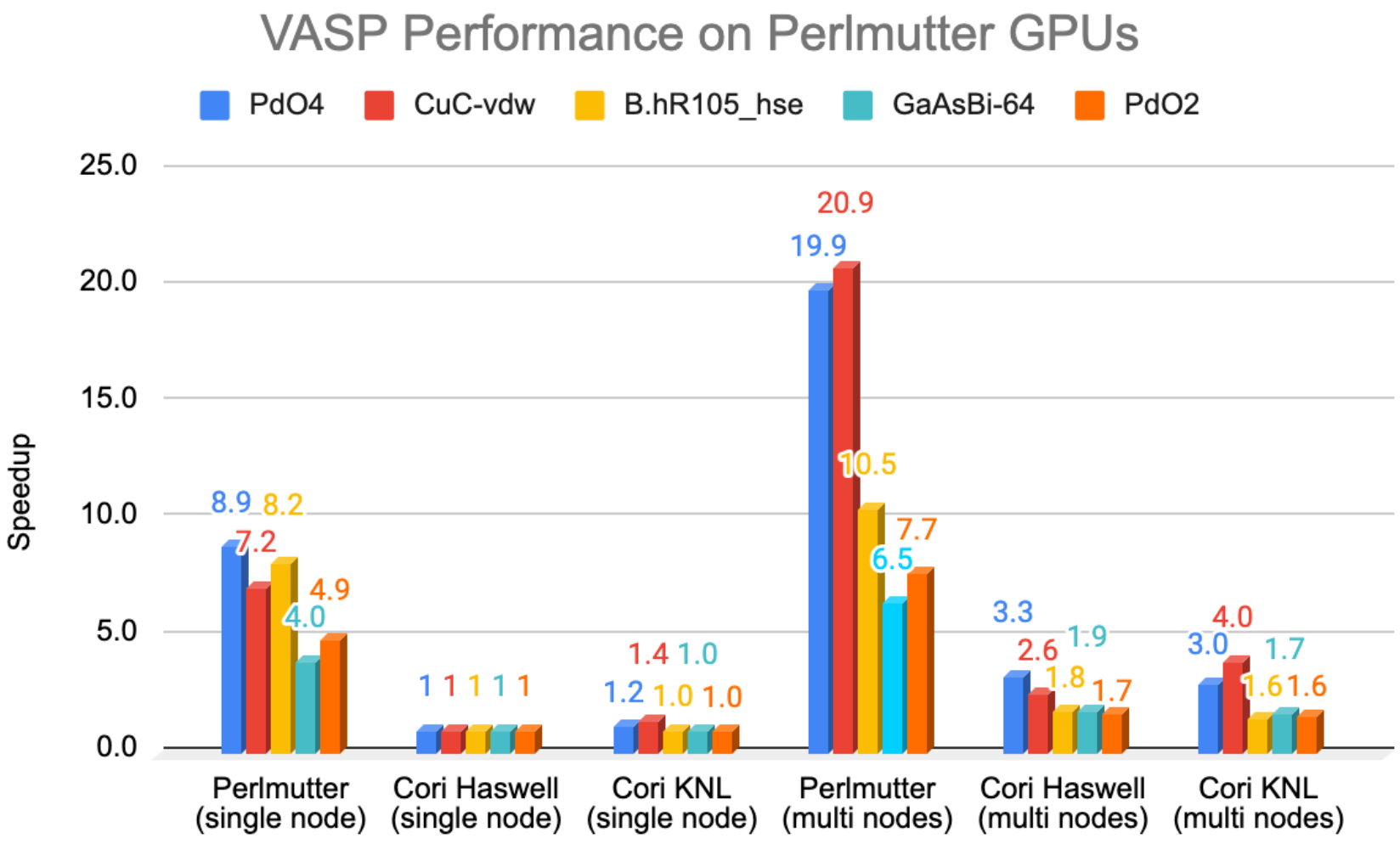
- VASP uses more than 15% of NERSC computing cycles each year
- 849 active VASP users on Perlmutter as of April, 15, 2025
- VASP was a Tier 2 NESAP application, and NVIDIA has been heavily involved with its OpenACC porting
- All major code paths in VASP have been ported to GPUs except the portion using Random Phase Approximation (GW and Adiabatic Connection Fluctuation Dissipation Theorem)

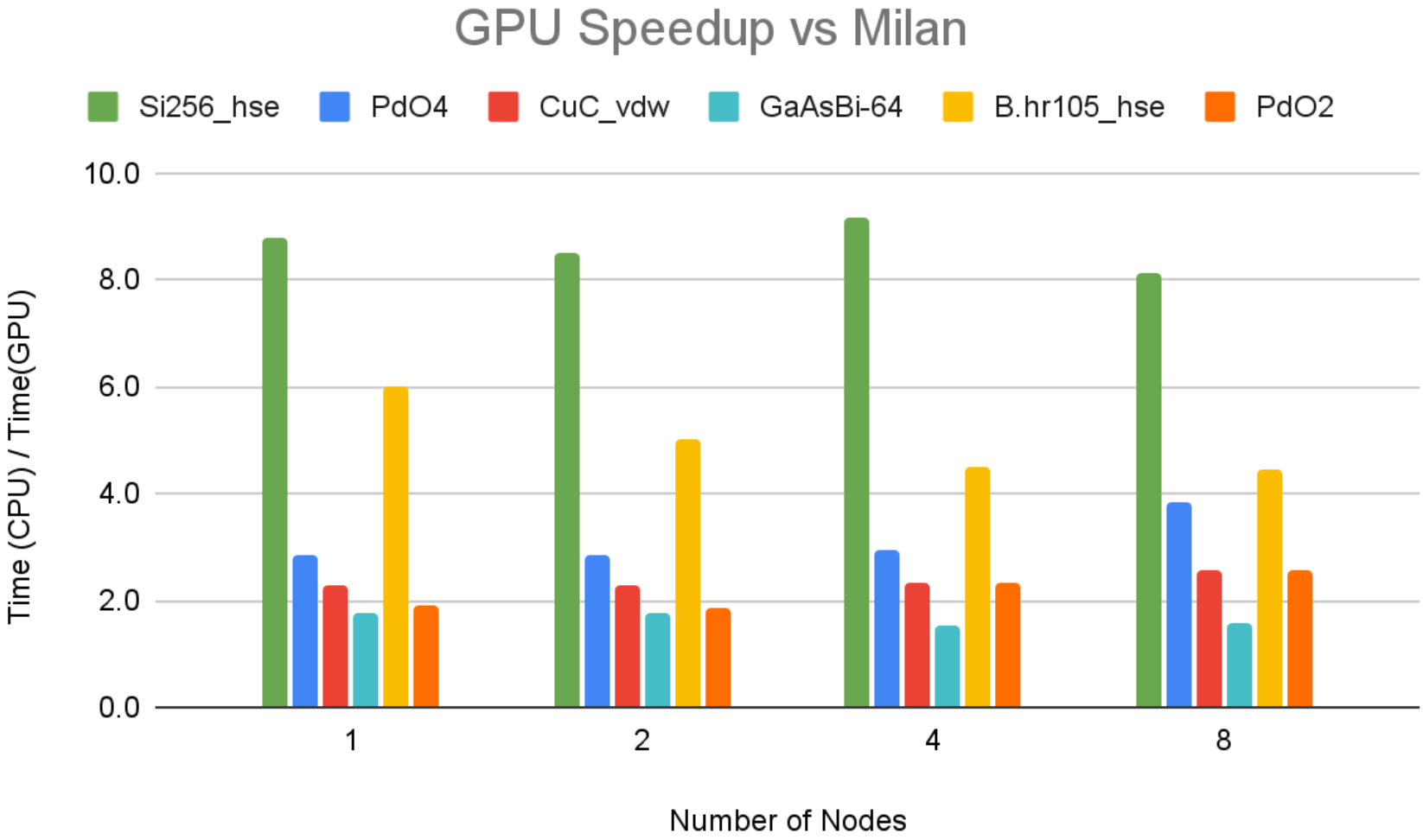
- N.B.: Perlmutter is undergoing continual changes and improvements, thus performance is subject to change

# Benchmarks 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<br>160x240x108 | 70x70x70<br>140x140x140 | 70x70x210<br>120x120x350 | 80x80x80<br>160x160x160 | 48x48x48<br>96x96x96 | 80x60x54<br>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 Performance on Perlmutter
(Si256_hse)
Perlmutter
Cori Haswell
Edison
Cori KNL
Speedup
150
100
50
0
0.7
1
1.6
8.2
15.9
31.4
54.1
94.6
120.8
140.9
13
9
15
Edison 1 node
Haswell 1 node
KNL 1 node
1/1
2/1
4/1
8/2
16/4
32/8
64/16
Number of GPUs / Number of Nodes
NESAP FOM


VASP Performance on Perlmutter GPUs
PdO4
CuC-vdw
B.hR105_hse
GaAsBi-64
PdO2
Speedup
25.0
20.0
15.0
10.0
5.0
0.0
8.9
8.2
7.2
4.9
4.0
1
1
1
1
1
1.4
1.0
1.2
1.0
1.0
20.9
19.9
10.5
7.7
6.5
3.3
2.6
1.9
1.8
1.7
4.0
3.0
1.7
1.6
1.6
Perlmutter (single node)
Cori Haswell (single node)
Cori KNL (single node)
Perlmutter (multi nodes)
Cori Haswell (multi nodes)
Cori KNL (multi nodes)


GPU Speedup vs Milan
Si256_hse
PdO4
CuC_vdw
GaAsBi-64
B.hr105_hse
PdO2
10.0
8.0
6.0
4.0
2.0
0.0
Time (CPU) / Time(GPU)
1
2
4
8
Number of Nodes

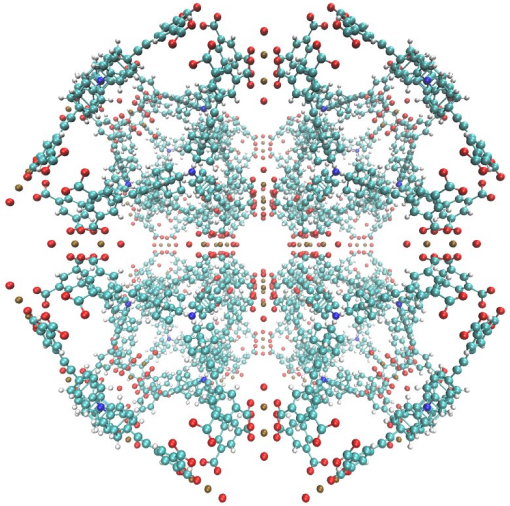
# Tips for Maximizing Performance

- VASP performance can be affected by many parameters, including system dimensions/symmetries and linked software libraries.

  **“Guess nothing, measure everything”** –Anon
- Use a **GPU** build if your application allows
- Use the newest version of VASP allowed by your license
- Do not run on hyperthreads
- (CPU only): measure NCORE=4, 8, 16, … vs. MPI-rank/OpenMP-thread ratios to find optimal values
- Set KPAR to parallelize over irreducible KPOINTS

# Some VASP Sample Jobs

# The Materials Genome (K. Peterson, LBNL)



MOF structure (**3584 atoms**)

VASP Input files provided by Shyam Dwaraknath at LBNL

- Job description (K. Peterson):

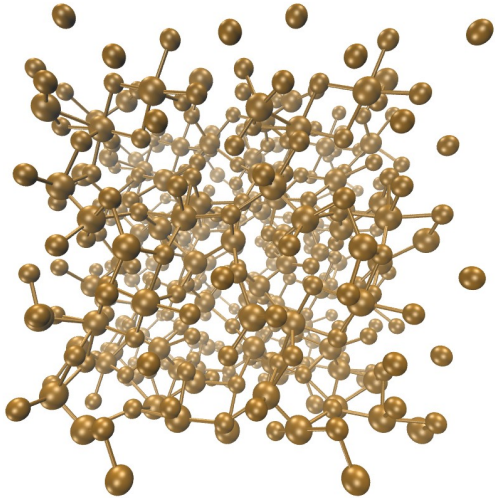
  "We have a large MOF structure that was sent to the Materials Project by a user, and it was too large for us to handle through our normal workflows"

- Computation details:

  Electrons(Ions): 13408 (3584)
  Functional: DFT
  Algo: CG (BD+RMM)
  **NBANDS= 9600**
  **FFT grids: 378x378x378; 756x756x756**
  NPLWV: 54,010,152
  KPOINTS: 1 1 1

- Completed 99 ionic steps in 32.8 hours using 32 nodes (128 GPUs) on Perlmutter

# Discovery of Materials for Energy Applications (C. Wolverton, Northwestern Univ.)



Amorphous VF3 (416 atoms)

VASP Input files provided by Shiqiang Hao Northwestern Univ.

- Job description (S. Hao):

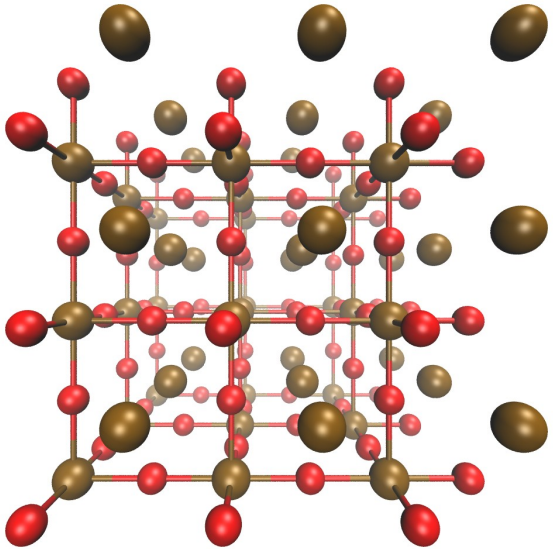
  “This job calculates Li diffusion in an amorphous phase of metal fluorides VF3 using ab initio molecular dynamics methods to investigate if the amorphous VF3 is a good Li conductor.”

- Computation details:

  Electrons (Ions): 2592 (416)
  Functional: DFT
  Algo: CG (BD+RMM)
  NBANDS: 1920
  NPLWV: 1905120
  FFT grids: 126x126x120; 192x180x180
  **KPOINTS: 7 7 7 (172 irreducible kpoints)**

- Completed 66 ionic steps in 17.2 hours using 160 nodes (640 GPUs) on Perlmutter

# Simulating Core-level Spectroscopy of Energy-relevant Materials (D. Prendergast, LBNL)



SrHfO3 (135 atoms)

VASP Input files provided by Ella Banyas at UCB

- Job description:

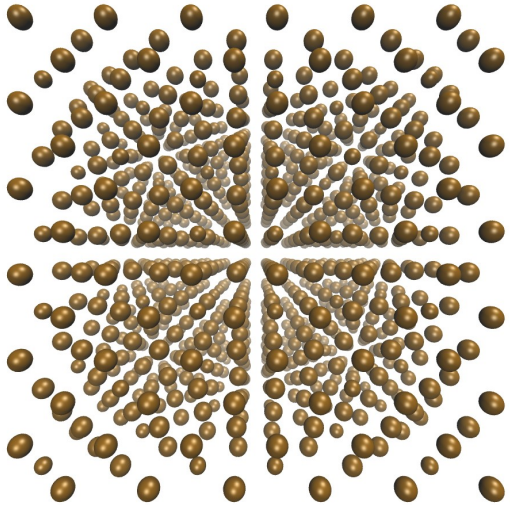
  A large supercell (135 atoms) **phonon calculations with DFPT**

- Computation details:

  Electrons (Ions): 1026 (135)
  Functional: DFT
  Algo: BD, RMM
  IBRION: 7; LEPSILON = .TRUE.
  NBANDS= 616
  NPLWV: 1259712
  FFT grids: 108x108x109; 216x216x216
  Support grids: 432x432x432
  KPOINTS: 3 3 3 (27 irreducible kpoints)

- Completed in 37 hours using 27 nodes (108 GPUs) on Perlmutter

# Material Simulations in Joint Center for Artificial Photosynthesis (L. Wang, LBNL)



NiMoNi (864 atom) alloy

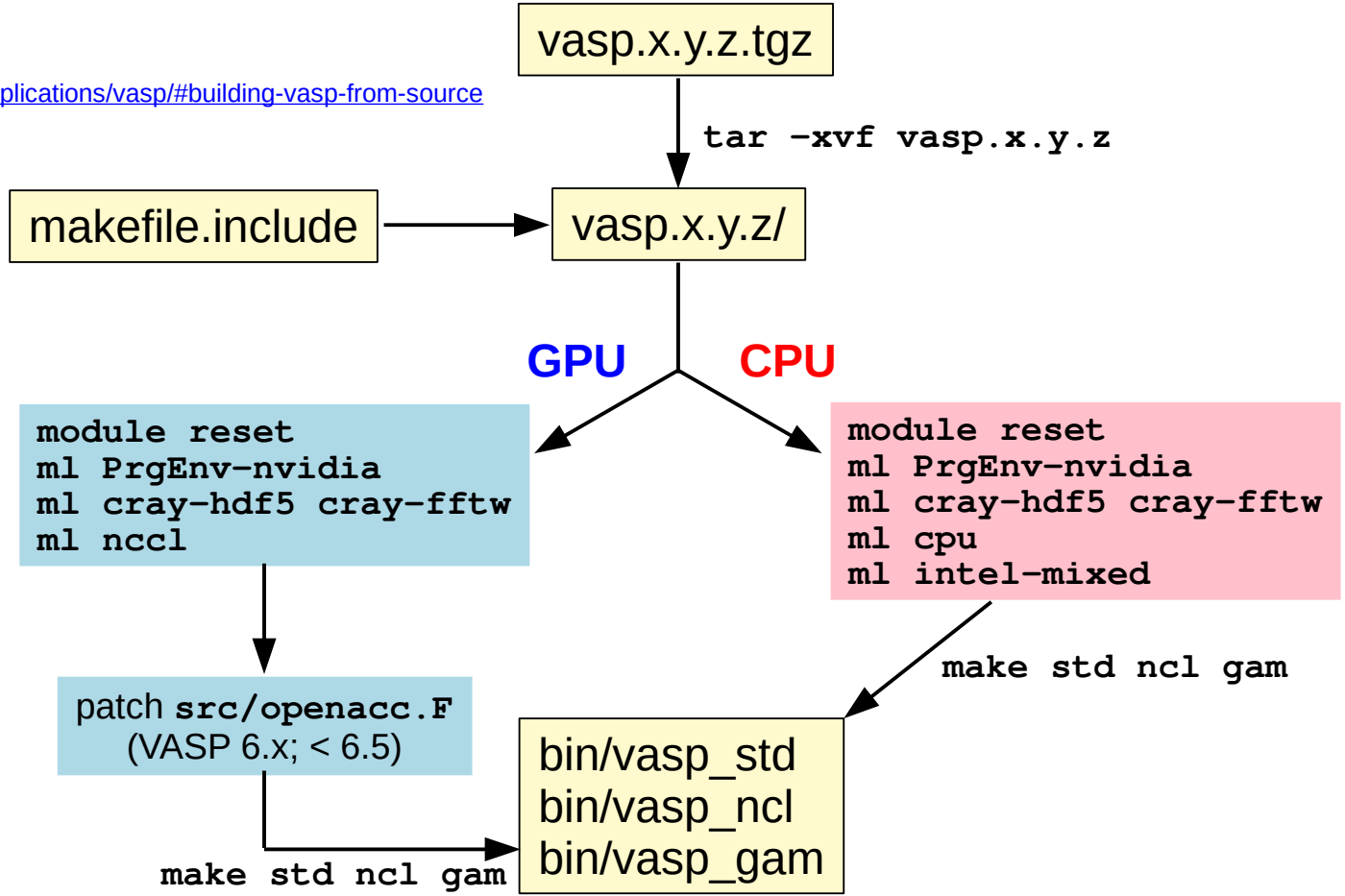
VASP Input files provided by Lin-Wang Wang LBNL

- Job description (L. Wang):
  - "This is an MD simulation for a BiMoNi alloy system which is used for catalysis to reduce Methane to carbon."
- Computation details
  - Electrons (Ions): 4984 (864)
  - Functional: DFT
  - Algo: RMM
  - MDALGO = 3
  - NBANDS= 3008
  - NPLWV: 941192
  - FFT grids: 65x65x65; 196x196x196
  - KPOINTS: 1 1 1
- Completed **3000 MD steps** in 43.3 hours using 16 nodes (64 GPUs) on Perlmutter
  - 52 seconds per MD step

# Compiling VASP for Perlmutter

# VASP Installation Workflow on Perlmutter

https://docs.nersc.gov/applications/vasp/#building-vasp-from-source



# Configuring VASP with **makefile.include**

- Sample ‘makefile.include’ for existing NERSC modules are at **`$MAKEFILE_INCLUDE_PATH/makefile.include`**
- Each programming environment requires its own ‘makefile.include’! NERSC-provided ones are specific to ‘PrgEnv-nvidia’.
- Samples for GNU, Intel can be found at https://www.vasp.at/wiki/index.php/Makefile.include (some modifications required)
- Can construct for other compilers but may require effort!

# Anatomy of a sample **makefile.include**

```
# Precompiler options
CPP_OPTIONS= -DHOST=\"LinuxNV_CrayMPICH\" \
          -DMPI -DMPI_BLOCK=8000 -Duse_collective \
          -DscaLAPACK \
          -DCACHE_SIZE=4000 \
          -Davoidalloc \
          -Dvasp6 \
          -Duse_bse_te \
          -Dtbdyn \
          -Dqd_emulate \
          -Dfock_dblbuf \
          -D_OPENMP \
          -D_OPENACC \
          -DUSENCCL \
          -DVASP_HDF5 \
          -DVASP2WANNIER90 \
          -DDFTD4 \
          -DPROFILING
### Disabled for GPU build:
#           -Dsysv \
#           -DUSELIBXC \
#           -Dlibbeef \

CPP       = nvfortran -Mpreprocess -Mfree -Mextend -E $(CPP_OPTIONS) $*$(FUFFIX)  >
FC        = ftn -mp -acc=gpu -gpu=cc80
FCL       = ftn -v -mp -acc=gpu -gpu=cc80 -c++libs
FREE      = -Mfree
```

- -D_OPENMP to add OpenMP parallelism
- -D_OPENACC to build for GPU; use -DUSENCCL to include NCCL for MPI+GPU performance
- -DVASP2WANNIER90, -DDFTD4, -DUSELIBXC, -DLIBBEEF to include optional plugins
- -DPROFILING to build version which prints timings

The compiler wrapper, **ftn**, is used for automatic inclusion of process specific optimizations and Cray libraries

https://www.vasp.at/wiki/index.php/Precompiler_options

# Anatomy of a sample **makefile.include** (cont'd)

```
# use cray-fftw module for FFTs
# optional packages:

# NCCL (GPU builds only)
LLIBS     += -L$(NCCL_DIR)/lib -Wl,-rpath=$(NCCL_DIR)/lib
INCS      += -I$(NCCL_DIR)/include

# HDF5 (vasp >6.2.0 only)
LLIBS     += -L$(HDF5_ROOT)/lib -lhdf5_fortran
INCS      += -I$(HDF5_ROOT)/include

# Use Intel MKL libraries (CPU builds only)
#MKLROOT    ?= /global/common/software/nersc/pm-stable/sw/intel/oneapi/mkl/latest
#LLIBS      += $(EXTLIBDIR)/mkl-interfaces/2022.1.0/milan/nvidia-22.5/libfftw3xf_nvidia.a
#LLIBS      += -L${MKLROOT}/lib/intel64 -lmkl_scalapack_lp64 -lmkl_intel_lp64 -lmkl_pgi_thread -lmkl_core -lmkl_blacs_intelmpi_lp64 -mp -lpthread
#LLIBS      += -Wl,-rpath=$(MKLROOT)/lib/intel64
#INCS       += -I$(MKLROOT)/include/fftw

# fftlib (with OpenMP only; useful for CPU builds)
#FCL       += fftlib.o
#CXX_FFTLIB  = nvc++ -mp --no_warnings -std=c++11 -DFFTLIB_THREADSAFE
#INCS_FFTLIB = -I./include -I$(FFTW_ROOT)/include
#LIBS       += fftlib
#LLIBS      += -ldl

# PATH FOR PLUGIN BUILDS
EXTLIBDIR   = $(CFS)/omp/local

# libbeef (CPU builds only)
LIBBEEF_ROOT = $(EXTLIBDIR)/libbeef/2020-11-01/nvidia-22.5/milan/lib
LLIBS      += -L$(LIBBEEF_ROOT) -lbeef
...
```

Load NCCL module and include this section to use in GPU builds

Uncomment Intel MKL and FFTlib for use in CPU builds for performance

This directory contains other optional third party codes builds (must be specified to use)

# Summary

# Summary and Looking Ahead

- While speed-ups depend on job sizes and types, VASP achieves **4x-30x** faster node-to-node performance on Perlmutter-GPU vs Cori Haswell with representative VASP workloads
- VASP achieves **1.5x-9x** faster node-to-node performance on Perlmutter GPUs vs Perlmutter CPUs for selected VASP workloads

- VASP 6.5+ builds are planned! If you have access to source then NERSC staff can assist you with building.
- VASP 5.4.4 support to continue through Perlmutter's life cycle, with expected deprecation in N10
- Containerized VASP planned for Perlmutter, likely will replace module environment in N10

# Thank You!

# Questions? We are here to help!

Q&A link:
https://tinyurl.com/3byu44c8
Survey link:
https://forms.gle/gdBKo2LM9E7hnFE98