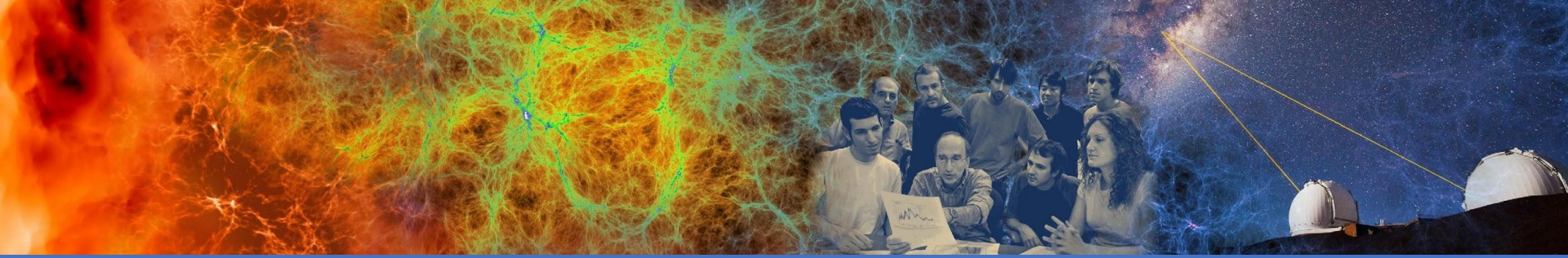
Using VASP at NERSC

VASP Hands-on Training
June 30, 2020

Zhengji Zhao
NERSC User Engagement Group

Outline

- Getting started with VASP at NERSC
- How to run VASP on Cori
- Best practices
- Running VASP with variable-time job scripts
- Summary



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
  - Note the email address change: licensing@vasp.at
- VASP access is controlled by unix groups
  - vasp5 for VASP 5
  - vasp6 for VASP 6
  - Type the "`groups`" command to check if you have access
- No need to confirm licenses if you build your own

Available VASP Modules on Cori

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

Build Type	Pure MPI	MPI+OpenMP	Comment
Standard Distributions	vasp/5.4.4-hsw(default) vasp/5.4.4-knl	vasp/20181030-hsw, vasp/20181030-knl vasp/6.1.0-hsw, vasp/6.1.0-knl	
Builds with the third party codes, Wannier90, VTST, BEEF, VASPSol enabled	vasp-tpc/5.4.4-hsw(default) vasp-tpc/5.4.4-knl	vasp-tpc/20170629-hsw vasp-tpc/20170629-knl	
Builds with NMAX_DEG=128		vasp/20170323_NMAX_DEG=128-hsw vasp/20170323_NMAX_DEG=128-knl	
An MPI wrapper for VASP to bundle many similar VASP jobs	mvasp/5.4.4-hsw mvasp/5.4.4-knl		Click here for more details

*) -hsw: optimized build for Haswell; -knl: optimized build for KNL; tpc stands for third party codes

Using VASP Modules

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

```
cori08:~> 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

 licensing@vasp.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/cnl7/haswell/vasp/pseudopotentials
setenv       VDW_KERNAL_DIR /global/common/sw/cray/cnl7/haswell/vasp/vdw_kernal
setenv       NO_STOP_MESSAGE 1
setenv       MPICH_NO_BUFFER_ALIAS_CHECK 1
prepend-path PATH /global/common/sw/cray/cnl7/haswell/vasp/vtstscripts/r933
prepend-path PATH /global/common/sw/cray/cnl7/haswell/vasp/5.4.4/intel/18.0.1.163/w5vq7o2/bin
```

Using VASP Modules (Cont.)

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

```
cori08:~> ls -l /global/common/sw/cray/cnl7/haswell/vasp/5.4.4/intel/18.0.1.163/w5vq7o2/bin
total 450600
-rwxr-xr-x 1 zz217 nstaff 156475624 Jul 24  2019 vasp_gam
-rwxr-xr-x 1 zz217 nstaff 152500944 Jul 24  2019 vasp_ncl
-rwxr-xr-x 1 zz217 nstaff 152433416 Jul 24  2019 vasp_std
```

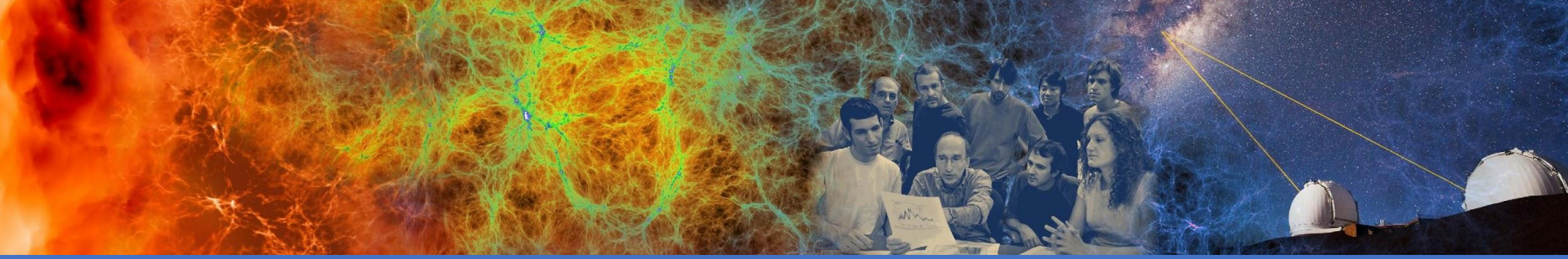
vasp_gam: the Gamma point only version
vasp_ncl: the non-collinear version
vasp_std: the standard kpoint version

- Type "`module load vasp`" to access VASP binaries

```
cori08:~> module load vasp

cori08:~> which vasp_std
/global/common/sw/cray/cnl7/haswell/vasp/5.4.4/intel/18.0.1.163/w5vq7o2/bin/vasp_std
```

- VTST scripts (UT Austin), pseudo potential files, and makefiles are available (check the installation directories)



Running VASP on Cori

Cori System Configuration

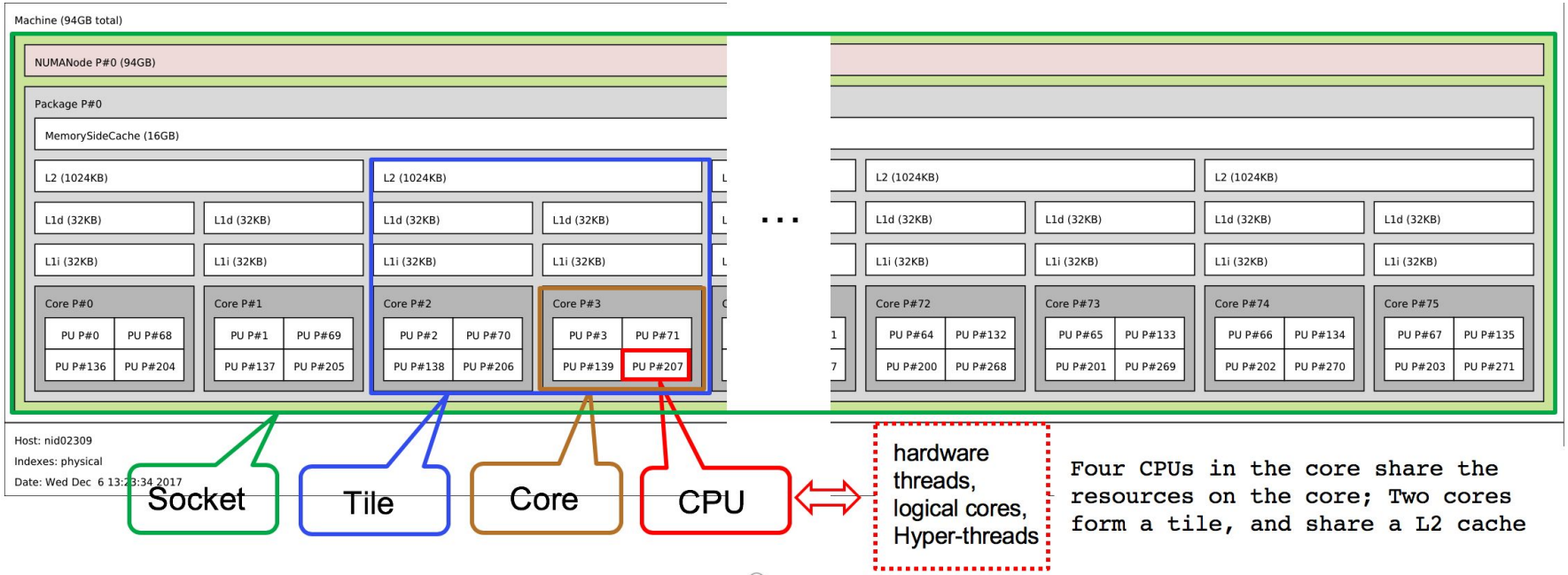
System	# of cores/ CPUs per node	# of CPUs per core	# of sockets per node	Clock Speed (GHz)	Memory/node	Memory/core
Cori KNL (9688 nodes)	68/272	4	1	1.4	96 GB DDR4 @2400 MHz; 16GB MCDRAM as cache	1.4 GB DDR 325 MB MCDRAM
Cori Haswell (2388 nodes)	32/64	2	2	2.3	128 GB DDR4 @2133 MHz	4.0 GB

- The memory available for user applications is 87GB (out of 96GB) per KNL node, and 118GB (out of 128GB) per Haswell node

Terminology: Cori Haswell Node Has 2 Sockets, 32 Cores and 64 CPUs

2 CPUs in the Haswell core share the resources (execution units, caches, etc.) on the core

Terminology: Cori KNL Node Has 1 Socket, 68 Cores, and 272 CPUs



For optimal performance:

- Placing 1 task or thread per core for VASP
  - Hyperthreading does not help VASP performance in most cases
- Evenly divide the available cores/CPUs on the node over the MPI tasks

Cori KNL Queue Policy

QOS	Max nodes	Max time (hrs)	Submit limit	Run limit	Priority	QOS Factor	Charge per Node-Hour
regular	9489	48	5000	-	4	1	80[5]
interactive[4]	64	4	2	2	-	1	80
debug	512	0.5	5	2	3	1	80
premium	9489	48	5	-	2	2	160[5]
low	9489	48	5000	-	5	0.5	40[6]
flex	256	48	5000	-	5	0.25	20
overrun[2]	9489	48	5000	-	7	0	0

Cori KNL Queue Policy (Cont.)

- interactive QOS - for interactive testing
  - Can use up to 64 nodes for 4 hours
  - Starts jobs immediately or cancels in 5 minutes
- flex QOS - for checkpoint/restart capable jobs
  - Must use `sbatch`'s `--time-min` 2 hours or less and `--time` > 2 hours
  - Can use up to 256 KNL nodes for 48 hours
  - Get a 75% charging discount and a faster queue turnaround
- regular QOS
  - Jobs that use 1024+ nodes on Cori KNL get a 50% charging discount and a higher scheduling priority
- Two job aging policy
  - Only two jobs per QOS per user accrue priority in the queue

Running VASP Interactively on Cori

- The interactive QOS allows quick access to compute nodes

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

nid02305:/global/cscratch1/sd/zz217/PdO4> module load vasp/20181030-knl
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/rank
  …
```

Example: using the interactive QOS

Sample Job Scripts to Run Pure MPI VASP Jobs

Cori KNL:

1 node

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -C knl
#SBATCH -q regular
#SBATCH -t 6:00:00

module load vasp/5.4.4-knl
srun -n64 -c4 --cpu-bind=cores vasp_std
```

Cori Haswell:

1 node

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -t 6:00:00

module load vasp #or module load vasp/5.4.4-hsw
srun -n32 -c2 --cpu-bind=cores vasp_std
```

- Launch 1 task per core for optimal performance
  - i.e., place 1 task every 4 CPUs (1 core) on KNL; Place 1 task every 2 CPUs (1 core) on Haswell
- Use the "`-c <ncpus>`" option to evenly divide the node's available CPUs over the MPI tasks
  - `-c, --cpus-per-task=<ncpus>` request that `ncpus` be allocated per task (or process)
- Use the "`--cpu-bind=cores`" option to bind tasks to CPUs
- Submit the batch script with `sbatch`, e.g., `sbatch run.slurm`

Sample Job Scripts to Run Pure MPI VASP Jobs (Cont.)

# of Nodes		Srun command line	Comment
KNL	nnodes	srun –n <ntasks> -c <ncpus> --cpu-bind=cores vasp_std ntasks = nnodes * (256/ncpus) where ncpus=4 is recommended for optimal performance (it can be 4, 2 or 1)	• Place 1 MPI task per Core (4 CPUs) • Using 64 cores (256 CPUs) out of 68 (272 CPUs) available per KNL node
	1	srun –n64 -c4 --cpu-bind=cores vasp_std	
	2	srun –n128 -c4 --cpu-bind=cores vasp_std	
	4	srun –n256 -c4 --cpu-bind=cores vasp_std	
Haswell	nnodes	srun -n <nnodes*64/ncpus> -c <ncpus> --cpu-bind=cores vasp_std where ncpus=2 is recommended	• Place 1 MPI task per Core (2 CPUs) • There are 64 CPUs per Haswell node
	1	srun –n32 –c2 --cpu-bind=cores vasp_std	
	2	srun –n64–c2 --cpu-bind=cores vasp_std	
	4	srun –n128 –c2 --cpu-bind=cores vasp_std	

Sample Job Scripts to Run Hybrid MPI + OpenMP VASP Jobs

Cori KNL:

```
#!/bin/bash -l
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 6:00:00
#SBATCH -C knl

module load vasp/20181030-knl
export OMP_NUM_THREADS=4

# launching 1 task every 4 cores (16 CPUs)
srun –n16 –c16 --cpu-bind=cores vasp_std
```

Cori Haswell:

```
#!/bin/bash -l
#SBATCH –N 1
#SBATCH –q regular
#SBATCH –t 6:00:00
#SBATCH -C haswell

module load vasp/20181030-hsw
export OMP_NUM_THREADS=4

# launching 1 task every 4 cores (8 CPUs)
srun –n8 –c8 --cpu-bind=cores vasp_std
```

- Launch 1 thread per core for optimal performance
  - i.e., place 1 task every 4 cores (16 CPUs) when running 4 OpenMP threads per MPI task on KNL
  - Place 1 task every 4 cores (8 CPUs) when running 4 OpenMP threads per MPI task on Haswell
- Total number of tasks `ntasks`
  - For KNL using 64 cores (256 CPUs) out of 68 (272 CPUs) available
  - 64 cores / 4 cores/task = 16 tasks on KNL; 32 cores/ 4 cores/task = 8 tasks on Haswell
- Use the "`-c <ncpus>`" option to evenly divide the node's CPUs over the MPI tasks
  - `ncpus=16` for KNL; `ncpus=8` for Haswell
- Use the `--cpu-bind=cores` option to bind tasks to CPUs

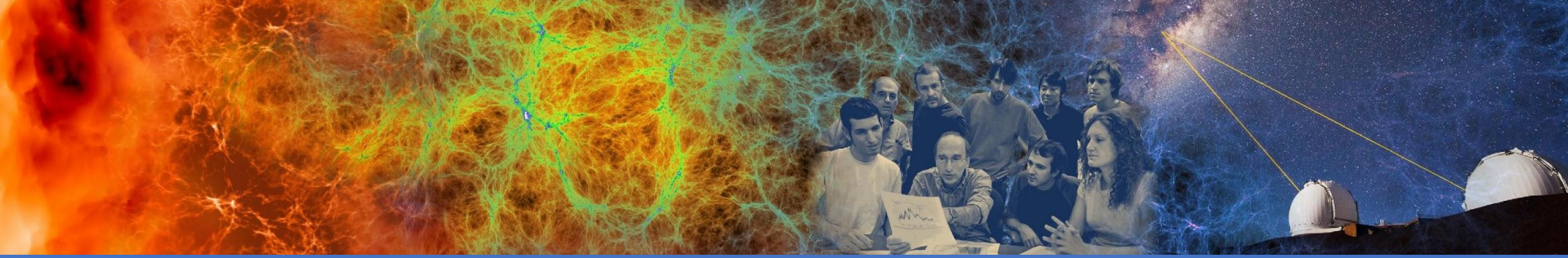
Sample Job Scripts to Run Hybrid MPI + OpenMP VASP Jobs (Cont.)

# of KNL Nodes	OMP_NUM_THREADS	Srun command line	Comment
nnodes	nthreads	#SBATCH -N nnodes module load vasp/20181030-knl export OMP_NUM_THREADS=nthreads `srun –n <ntasks> -c <ncpus> --cpu-bind=cores vasp_std` where ncpus = 4 * nthreads; ntasks = nnodes * (256/ncpus)	• Place 1 thread per core for optimal performance • Usws 64 cores (256 CPUs) out of 68 (272 CPUs) available per KNL node • OMP_NUM_THREADS=4, or 8 is recommended • Hyper-Threading is not recommended for most VASP jobs
1	4	srun –n16 –c16 --cpu-bind=cores vasp_std	
1	8	`srun –n8 –c32 --cpu-bind=cores vasp_std`	
2	4	srun –n32 –c16 --cpu-bind=cores vasp_std	
2	8	`srun –n16 –c32 --cpu-bind=cores vasp_std`	
4	4	`srun –n64 -c16 --cpu-bind=cores vasp_std`	
4	8	`srun –n32 –c32 --cpu-bind=cores vasp_std`	

Job script generator: https://my.nersc.gov/script_generator.php

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
- `ssh_job <jobid>` to ssh to the head compute nodes of your running jobs
  - you can then run your favorite commands to monitor your running jobs, e.g., the `top` command



Best Practices

To Get Through the Queue Faster

- Be aware of the two job aging policy
  - Only two jobs per QOS per user accrue priority in the queue
- Bundle jobs to get improved throughput
  - Multiple sruns + job resizing
  - Use the mvasp modules to bundle many similar VASP jobs
    - Can get a 50% large job charging discount if using more than 1024 nodes
- Run VASP jobs on Cori KNL
  - More nodes and a shorter queue backlog
- Use variable-time job scripts to improve the queue turnaround
  - Automatically makes use of the backfill opportunity
  - If used with the flex QOS, you get a 75% charging discount

Bundle Jobs with Multiple Sruns

- Sample job script bundling multiple VASP jobs

```
#!/bin/bash
#SBATCH --qos=debug
#SBATCH --nodes=5
#SBATCH --time=30:00
#SBATCH --constraint=knl

module load vasp/5.4.4-knl

#Assume 5 VASP jobs in run1, run2, …, run5 directories
for j in {1..5}; do
    cd run$j
    srun -N 1 -n 64 -c 4 --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

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 5
#SBATCH -t 30:00
#SBATCH -C knl

module load vasp/5.4.4-knl
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 $mynode -n64 -c4 --cpu-bind=cores -o %x-%J-${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 Many VASP Jobs Simultaneously with a VASP Wrapper

- An MPI wrapper for VASP 5.4.4 is available on Cori
- To use it, do
  - `module load mvasp`
  - run the "`gen_joblist.sh`" script to generate the job list file, `joblist.in`

Sample job script to bundle VASP jobs with the MPI wrapper for VASP

```
#!/bin/bash
#SBATCH -J test_mvasp
#SBATCH -N 512
#SBATCH -C knl
#SBATCH -q debug
#SBATCH -o %x-%j.out
#SBATCH -t 30:00

module load mvasp/5.4.4-knl

#run 512 VASP jobs simultaneously each running vasp_std with 1 KNL node (64 processes)
sbcast --compress=lz4 `which mvasp_std` /tmp/mvasp_std
srun -n 32768 -c4 --cpu-bind=cores /tmp/mvasp_std
```

  - Sample output is available here
  - The one downside: a single job failure causes all jobs to fail

Parallel Efficiency Considerations

- Running VASP beyond its parallel scaling region may have very limited benefits (even if not running into errors)
  - Not charging efficient
  - For very small jobs, consider the shared QOS
- 1 rank/atom is a good reference when choosing the number of processes to run your VASP jobs
  - To find the most efficient NCORE/NPAR and NSIM values for your jobs, you need to do your own benchmarking
  - Use KPAR if your system contains many kpoints

Performance Considerations

- For small DFT calculations, you may see better performance with the pure MPI VASP on Haswell
- For HSE workloads and VDW calculations, it may be beneficial to run the hybrid MPI+OpenMP VASP on KNL
  - However, running VASP on Cori KNL is recommended for better queue turnaround
- Running the hybrid MPI + OpenMP VASP on Cori KNL is recommended
  - 4 or 8 OpenMP threads per MPI task is recommended
  - Hyperthreads are not recommended with most VASP jobs (except occasionally HSE)
  - Use 64 cores out of 68 on KNL in most cases
  - Note that NCORE /=1 is not supported in the hybrid VASP

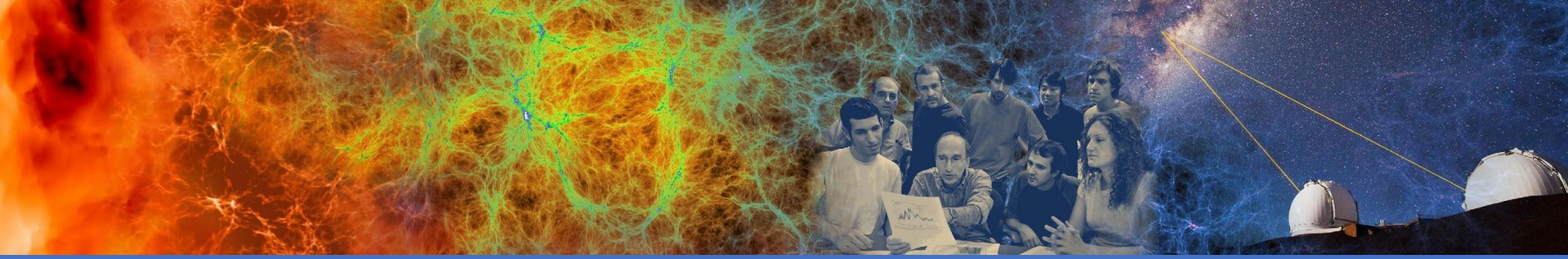
Performance Considerations (Cont.): Process/Thread Affinity

- Slurm imposes CPU bindings (auto-binding) only for limited use cases
  - For example, when MPI tasks per node x CPUs per task = the total number of CPUs allocated per node (e.g., 68x4=272 on KNL)
- Always use srun's `--cpu-bind` and `-c` options explicitly to achieve optimal process/thread affinity (as in the sample job scripts)
  - `-c` is used to evenly divide the node's available CPUs over the MPI tasks
  - `--cpu-bind` is used to pin processes to CPUs
  - Use OMP environment variables to fine control thread affinity
    - `export OMP_PROC_BIND=true`
    - `export OMP_PLACES=threads`
    - These two envs are set in the hybrid MPI+OpenMP VASP modules
- The Job script generator is helpful

Performance Considerations (Cont.):

- Do not run production VASP jobs from your /global/homes directory
  - Use the scratch file system (cscratch1) and community file system (cfs)
  - The Burst Buffer offers the best I/O performance if needed
- For large VASP jobs, copy the vasp binaries to /tmp (memory) and launch them from there

```
sbcast --compress=lz4 `which vasp_std` /tmp/vasp_std
srun -n 4096 -c4 --cpu-bind=cores /tmp/vasp_std
```



Running VASP with Variable-Time Job Scripts

What are Variable-Time Job Scripts?

Variable-Time Job (VTJ) Scripts

- A VTJ script splits a long running job into multiple shorter chunks to make use of the backfill opportunity and automates pre-empted job resubmissions
- Are for jobs that can resume from where they left off
- Adds a few `sbatch` directives and `bash` functions in your job script

sbatch directives:

```
#SBATCH --time-min=2:00:00
#SBATCH --signal=B:USR1@<sig_time>
#SBATCH --requeue
#SBATCH --open-mode=append
```

Bash functions

```
Requeue_job – trap signals
func_trap – action upon trap
parse_job - process job info
```

The core of the automation is trapping signals

Why Use Variable-Time Job Scripts?

- Greatly improved queue turnaround
- Enables jobs of any length
  - e.g., a week or even longer, as long as the jobs can restart by themselves
- Can run with any QOS on both Cori KNL and Haswell
- With flex QOS, you get a 75% charging discount
  - Flex QOS is available on Cori KNL only
- VTJs are an important component in the C/R road map at NERSC
  - C/R is integral to many future plans at NERSC
  - Getting an early start on variable-time jobs would be helpful for you in the future

How Does a Variable-Time Job Work?

A Regular VASP Job Script (Atomic Relaxation Jobs)

Regular QOS VASP jobs

```
#!/bin/bash
#SBATCH -q regular
#SBATCH -N 2
#SBATCH -C knl
#SBATCH -t 48:00:00

module load vasp/20181030-knl
export OMP_NUM_THREADS=4

# launching 1 task every 4 cores (16 CPUs)
srun -n32 -c16 --cpu_bind=cores vasp_std
```

Flex QOS VASP jobs

```
#!/bin/bash
#SBATCH -q flex
#SBATCH -N 2
#SBATCH -C knl
#SBATCH -t 48:00:00
#SBATCH --time-min=2:00:00

module load vasp/20181030-knl
export OMP_NUM_THREADS=4

# launching 1 task every 4 cores (16 CPUs)
srun -n32 -c16 --cpu_bind=cores vasp_std
```

- You can use the flex QOS for a 75% charging discount and a faster queue turnaround
- Manual resubmissions of the pre-terminated jobs are required

What a Variable-Time Job Script looks like

For automatic resubmission of pre-terminated jobs

```
#!/bin/bash
#SBATCH -q flex
#SBATCH -N 2
#SBATCH -C knl
#SBATCH -t 48:00:00
#SBATCH --comment=48:00:00
#SBATCH --time-min=02:00:00
#SBATCH --signal=B:USR1@900
#SBATCH --requeue
#SBATCH --open-mode=append

module load vasp/20181030-knl
export OMP_NUM_THREADS=4

# srun must execute in background and catch signal
on wait command
srun -n32 -c16 --cpu-bind=cores vasp_std &

wait
```

```
# put any commands that need to run to continue
# the next job here
ckpt_vasp() {
    set -x
    restarts=`squeue -h -O restartcnt -j $SLURM_JOB_ID`
    echo checkpointing the ${restarts}-th job

    # to terminate VASP at the next ionic step
    echo LSTOP = .TRUE. > STOPCAR
    # wait until VASP to complete the current ionic step,
    # write WAVECAR file and quit
    srun_pid=`ps -fle|grep srun|head -1|awk '{print $4}'`
    wait $srun_pid

    # copy CONTCAR to POSCAR
    cp -p CONTCAR POSCAR
    set +x
}

ckpt_command=ckpt_vasp
max_timelimit=48:00:00

# requeueing the job if remaining time >0
. /global/common/cori/software/variable-time-job/setup.sh
requeue_job func_trap USR1
```

https://docs.nersc.gov/jobs/examples/#vasp-example

For automatic resubmission of pre-terminated jobs

```
#!/bin/bash
#SBATCH -q flex
#SBATCH -N 2
#SBATCH -C knl
#SBATCH -t 48:00:00
#SBATCH --comment=48:00:00
#SBATCH --time-min=02:00:00
#SBATCH --signal=B:USR1@900
#SBATCH --requeue
#SBATCH --open-mode=append

module load vasp/20181030-knl
export OMP_NUM_THREADS=4

# srun must execute in background and catch signal
on wait command
srun -n32 -c16 --cpu-bind=cores vasp_std &

wait
```

```
# put any commands that need to run to continue
# the next job here
ckpt_vasp() {
    set -x
    restarts=`squeue -h -O restartcnt -j $SLURM_JOB_ID`
    echo checkpointing the ${restarts}-th job

    # to terminate VASP at the next ionic step
    echo LSTOP = .TRUE. > STOPCAR
    # wait until VASP to complete the current ionic step,
    # write WAVECAR file and quit
    srun_pid=`ps -fle|grep srun|head -1|awk '{print $4}'`
    wait $srun_pid

    # copy CONTCAR to POSCAR
    cp -p CONTCAR POSCAR
    set +x
}

ckpt_command=ckpt_vasp
max_timelimit=48:00:00

# requeueing the job if remaining time >0
. /global/common/cori/software/variable-time-job/setup.sh
requeue_job func_trap USR1
```

https://docs.nersc.gov/jobs/examples/#vasp-example

Explaining the VTJ Script Line-by-Line

`#SBATCH --comment=48:00:00`

- A flag to add comments about your job
- Specifies the desired walltime and to track the remaining walltime for the pre-terminated jobs
  - You can specify any length of time, e.g., a week or even longer

`#SBATCH --time-min=02:00:00`

- Specifies the minimum time for your job
- 2 hours or less for Flex QOS

`#SBATCH --signal=B:USR1@<sig_time>`

- Used to request that the batch system sends a user defined signal USR1 to the batch shell (where the job is running) **`sig_time`** seconds before the job hits the wall clock limit
- **`sig_time`** should match the checkpoint overhead of your job

`#SBATCH --requeue`

- Specifies that the batch job should be eligible to be requeued

`#SBATCH --open-mode=append`

- Appends the standard output/error of the requeued job to the same standard output/error files from the previous job
- This is optional; if not used, each requeued job creates its own standard output/error files.

For automatic resubmission of pre-terminated jobs

```
#!/bin/bash
#SBATCH -q flex
#SBATCH -N 2
#SBATCH -C knl
#SBATCH -t 48:00:00
#SBATCH --comment=48:00:00
#SBATCH --time-min=02:00:00
#SBATCH --signal=B:USR1@900
#SBATCH --requeue
#SBATCH --open-mode=append

module load vasp/20181030-knl
export OMP_NUM_THREADS=4

# srun must execute in background and catch signal
on wait command
srun -n32 -c16 --cpu-bind=cores vasp_std &

wait
```

```
# put any commands that need to run to continue
# the next job here
ckpt_vasp() {
    set -x
    restarts=`squeue -h -O restartcnt -j $SLURM_JOB_ID`
    echo checkpointing the ${restarts}-th job

    # to terminate VASP at the next ionic step
    echo LSTOP = .TRUE. > STOPCAR
    # wait until VASP to complete the current ionic step,
    # write WAVECAR file and quit
    srun_pid=`ps -fle|grep srun|head -1|awk '{print $4}'`
    wait $srun_pid

    # copy CONTCAR to POSCAR
    cp -p CONTCAR POSCAR
    set +x
}

ckpt_command=ckpt_vasp
max_timelimit=48:00:00

# requeueing the job if remaining time >0
. /global/common/cori/software/variable-time-job/setup.sh
requeue_job func_trap USR1
```

https://docs.nersc.gov/jobs/examples/#vasp-example

Explaining the VTJ Script Line-by-Line (Cont.)

`ckpt_command=ckpt_vasp`

- The command to run to checkpoint your job (if any)
- It is run inside the function **`requeue_job`** upon receiving the USR1 signal

`max_timelimit=48:00:00`

- Use this to specify the max time for the requeued job
- This can be any time less than or equal to the max time limit allowed by the batch system
  - The default is 48 hours if not specified

`&` and **`wait`**

- To put the `srun` execution to the background so that `srun` continues to run as needed to complete the checkpoint command (**`ckpt_command`**)

`/global/common/cori/software/variable-time-job/setup.sh`

- A few bash functions are defined in this setup script to automate job resubmissions
  - e.g., **`requeue_job`** and **`func_trap`**

For automatic resubmission of pre-terminated jobs

```
#!/bin/bash
#SBATCH -q flex
#SBATCH -N 2
#SBATCH -C knl
#SBATCH -t 48:00:00
#SBATCH --comment=48:00:00
#SBATCH --time-min=02:00:00
#SBATCH --signal=B:USR1@900
#SBATCH --requeue
#SBATCH --open-mode=append

module load vasp/20181030-knl
export OMP_NUM_THREADS=4

# srun must execute in background and catch signal
on wait command
srun -n32 -c16 --cpu-bind=cores vasp_std &

wait
```

```
# put any commands that need to run to continue
# the next job here
ckpt_vasp() {
    set -x
    restarts=`squeue -h -O restartcnt -j $SLURM_JOB_ID`
    echo checkpointing the ${restarts}-th job

    # to terminate VASP at the next ionic step
    echo LSTOP = .TRUE. > STOPCAR
    # wait until VASP to complete the current ionic step,
    # write WAVECAR file and quit
    srun_pid=`ps -fle|grep srun|head -1|awk '{print $4}'`
    wait $srun_pid

    # copy CONTCAR to POSCAR
    cp -p CONTCAR POSCAR
    set +x
}

ckpt_command=ckpt_vasp
max_timelimit=48:00:00

# requeueing the job if remaining time >0
. /global/common/cori/software/variable-time-job/setup.sh
requeue_job func_trap USR1
```

https://docs.nersc.gov/jobs/examples/#vasp-example

Explaining the VTJ Script Line-by-Line (Cont.)

requeue_job

- This function traps the user-defined signal (e.g., `USR1`)
- Upon receiving the signal, it executes a function (e.g., **func_trap** below) that is provided on the command line in the script.

```
requeue_job() {
    parse_job    # calculates the remaining walltime, updates the --comment flag
    if [ -n $remainingTimeSec ] && [ $remainingTimeSec -gt 0 ]; then
       commands=$1
       signal=$2
       trap $commands $signal
    fi
}
```

func_trap

- This function contains the list of commands to be executed to initiate the checkpointing, prepares inputs for the next job, requeues the job, and updates the remaining walltime.

```
func_trap() {
    $ckpt_command
    scontrol requeue ${SLURM_JOB_ID}
    scontrol update JobId=${SLURM_JOB_ID} TimeLimit=${requestTime}
}
```

How Does Automatic Resubmission Work?

1. The user submits the variable-time job script.
2. The batch system looks for a backfill opportunity for the job. If it can allocate the requested number of nodes for this job for any duration (e.g., 6 hours) between the specified minimum time (2 hours) and the time limit (48 hours) before those nodes are used for other higher priority jobs, the job starts execution.
3. The job runs until it receives a signal `USR1` (**`--signal=B:USR1@900`**) 900 seconds before it hits the allocated time limit (6 hours).
4. Upon receiving the signal, the **`func_trap`** function gets executed, which in turn
   a. Executes the **`ckpt_command`** if specified.
   b. Requeues the job and updates the remaining walltime for the requeued job.

```
func_trap() {
    $ckpt_command
    scontrol requeue ${SLURM_JOB_ID}
    scontrol update JobId=${SLURM_JOB_ID}
TimeLimit=${requestTime}
}
```

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

    cp -p CONTCAR POSCAR
}
ckpt_command=ckpt_vasp
```

5. Steps 2-4 repeat until the job runs for the desired amount of time (48 hours) or the job completes.
6. User checks the results.

How to Work with Variable-Time Jobs

Adjust the Sample VTJ Script for Your Jobs

- The `sig_time` should match the checkpoint-overhead of your job
  - The sample script waits for the current ionic step to complete after the signal is sent
  - The sample used 15 minutes (900 seconds) - this may not be enough for your jobs
- Check the commands inside the `vasp_ckpt` function and add/remove anything you need to resume the next job, storing the intermediate results if needed
- Note that not all VASP jobs are currently restartable
  - We are working towards getting an external C/R tool, DMTCP, to work with VASP (target date: by the end of July)

Checking on Variable-Time Jobs

- **sacct –j <jobid> -D**

```
-D, --duplicates

    If Slurm job ids are reset, some job numbers will probably appear more than once in the
    accounting log file but refer to different jobs.  Such jobs can be distinguished by the "submit"
    time stamp in the data records.

    When data for specific jobs are requested with the --jobs option, sacct returns the most recent
    job with that number. This behavior can be  overridden  by specifying --duplicates, in which
    case all records that match the selection criteria will be returned.
```

- **sacct –j <jobid> -D -X**

```
-X, --allocations

    Only show statistics relevant to the job allocation itself, not taking steps into consideration.
```

Example Output of the `sacct -D` Command

```
zz217@cori08:~> sacct -j 30774616 -X -o jobid,jobname,submit,start,end,nnodes,timelimit,elapse,exit,state
       JobID    JobName             Submit               Start                 End   NNodes  Timelimit    Elapsed ExitCode      State
------------ ---------- ------------------- ------------------- ------------------- -------- ---------- ---------- -------- ----------
30774616             md 2020-05-19T13:31:49 2020-05-19T14:46:16             Unknown       32   14:13:00   11:53:23      0:0    RUNNING


zz217@cori08:~> sacct -j 30774616 -X -D -o jobid,jobname,submit,start,end,nnodes,timelimit,elapse,exit,state
       JobID    JobName             Submit               Start                 End   NNodes  Timelimit    Elapsed ExitCode      State
------------ ---------- ------------------- ------------------- ------------------- -------- ---------- ---------- -------- ----------
30774616             md 2020-05-15T23:01:27 2020-05-16T04:33:38 2020-05-16T07:01:51       32 2-00:00:00   02:28:13      0:0   REQUEUED
30774616             md 2020-05-16T07:03:28 2020-05-16T12:08:32 2020-05-16T14:33:37       32 2-00:00:00   02:25:05      0:0   REQUEUED
30774616             md 2020-05-16T14:34:50 2020-05-17T14:10:52 2020-05-17T19:44:05       32 2-00:00:00   05:33:13      0:0   REQUEUED
30774616             md 2020-05-17T19:45:20 2020-05-18T02:50:03 2020-05-18T05:30:10       32 2-00:00:00   02:40:07      0:0   REQUEUED
30774616             md 2020-05-18T05:31:12 2020-05-18T05:42:46 2020-05-18T10:13:47       32 2-00:00:00   04:31:01      0:0   REQUEUED
30774616             md 2020-05-18T10:14:04 2020-05-19T03:38:37 2020-05-19T08:52:46       32 2-00:00:00   05:14:09      0:0   REQUEUED
30774616             md 2020-05-19T08:54:11 2020-05-19T11:01:27 2020-05-19T13:31:28       32 2-00:00:00   02:30:01      0:0   REQUEUED
30774616             md 2020-05-19T13:31:49 2020-05-19T14:46:16             Unknown       32   14:13:00   11:50:02      0:0    RUNNING
```

For more details, run `sacct` without the `-X` option.

If Your Variable-Time Job Fails to Requeue Itself

- Check the standard error file for execution details
- Check if your job ran into any errors
  - If your job failed before the allocated time limit, the `USR1` signal will not be sent so that the job will not requeue
- Check if `<sig_time>` is long enough - it should equal or exceed the checkpoint overhead time
  - If your job runs out of time while executing the `ckpt_command`, then the job will not requeue
  - You can send the USR1 signal to your running job outside the job script any time if needed using `scancel -b -s USR1 <jobid>`.

Example: Standard Error File from a Variable-Time VASP Job

```
zz217@cori08:/global/cscratch1/sd/zz217/vtj/flex_test/sd1-flex1> more md-30774616.err
time remaining $remainingTime: 165:39:00
next timelimit $requestTime: 172800
++ ckpt_vasp
+++ squeue -h -O restartcnt -j 30774616
++ restarts='0                    '
++ echo checkpointing the 0 -th job
checkpointing the 0 -th job
++ echo LSTOP = .TRUE.
+++ ps -fle
+++ grep srun
+++ head -1
+++ awk '{print $4}'
++ srun_pid=134834
++ wait 134834
 PROFILE, used timers:     418
++ cp -p CONTCAR POSCAR
++ trap '' SIGTERM
++ scontrol requeue 30774616
slurmstepd: error: *** JOB 30774616 ON nid06740 CANCELLED AT 2020-05-16T07:01:51 DUE TO JOB REQUEUE ***
++ scontrol update JobId=30774616 TimeLimit=2880
++ trap - SIGTERM
++ echo '$?:' 0
$?: 0
++ set +x
time remaining $remainingTime: 163:23:00
next timelimit $requestTime: 172800
++ ckpt_vasp
...
```

VASP checkpoint function definition

```
# put any commands that need to run to prepare for the next job here
ckpt_vasp() {
    set -x
    #get the restart count
    restarts=`squeue -h -O restartcnt -j $SLURM_JOB_ID`
    echo checkpointing the ${restarts}-th job

    #to terminate VASP at the next ionic step
    echo LSTOP = .TRUE. > STOPCAR

    #wait until VASP to complete the current ionic step
    srun_pid=`ps -fle|grep srun|head -1|awk '{print $4}'`
    wait $srun_pid

    #prepare inputs for next job
    cp -p CONTCAR POSCAR
    set +x
}

ckpt_command=ckpt_vasp
```

The function executed upon receiving signal USR1

```
func_trap() {
    $ckpt_command
    scontrol requeue ${SLURM_JOB_ID}
    scontrol update JobId=${SLURM_JOB_ID} TimeLimit=${requestTime}
}
```

Custom Variable-Time-Job Scripts for Your Needs

- You are encouraged to customize the setup script for your needs
  - Get a local copy of `/usr/common/software/variable-time-job/setup.sh`, and modify just three core functions as needed
- If you don't like long VTJ scripts, you can hide most of them
  - Source the `setup.sh` file and define the `ckpt_vasp` function in your shell startup files, `~/.bashrc`
  - Put some sbatch options to your `~/.slurm/defaults` (your job scripts can overwrite these defaults)

```
zz217@cori04:~> cat ~/.slurm/defaults
account=nstaff
mail-type=BEGIN,END,FAIL,REQUEUE
escori:qos=xfer
cori:constraint=knl
#time-min=2:00:00
#nodes=1
#qos=interactive
```

An example `~/.slurm/defaults` file

```
#!/bin/bash

# user setting goes here
export OMP_NUM_THREADS=4
srun -n32 -c16 --cpu_bind=cores ./a.out &

# requeueing the job if remaining time >0
ckpt_command=ckpt_vasp
requeue_job func_trap USR1

wait
```

A simplified sample VTJ script

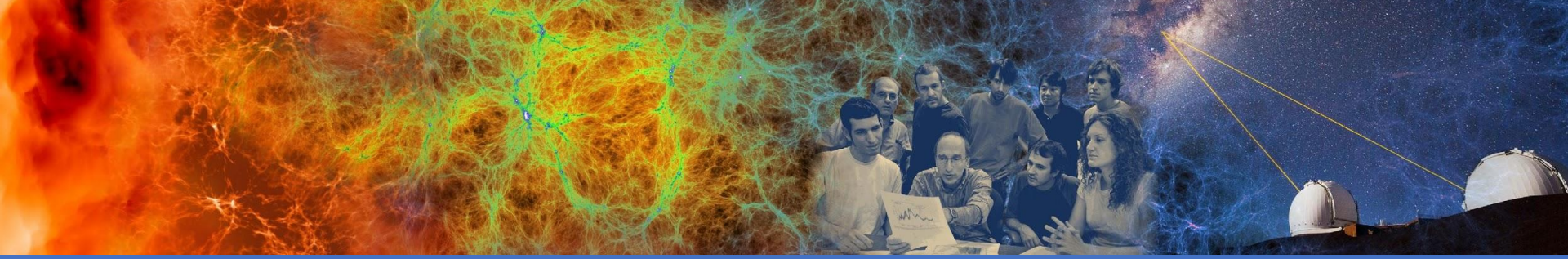
Custom VTJ Scripts for Your Needs (Cont.)

- Efforts to make VTJ scripts more user friendly are underway (still experimental). You may want to try out the `nersc_cr/vasp` module

```
...
#Same SBATCH directives
# nersc_cr module is still experimental, log some debugging info:
export _DEBUG_RESTARTABLE=1
module load nersc_cr/vasp

# options for run_with_cr:
# "-cr vasp" means "use checkpoint/restart strategy that targets Vasp jobs"
# (other strategies will be added eventually)
# "-t 96:00:00" means "run for 96 hours in total (over however many restarts
# that takes)
run_with_cr -cr vasp -t 96:00:00 srun -n 64 -c16 --cpu_bind=cores vasp_std
```

How to use the `nersc_cr/vasp` module



Summary

Summary

- Running VASP jobs on Cori KNL is highly recommended, as it has a much shorter backlog and more nodes in comparison to Cori Haswell
- Using variable-time job scripts
  - Greatly improves queue turnaround
  - Offers a 75% charging discount with flex QOS on Cori KNL
  - Enables long running jobs (e.g., weeks long jobs)
- Bundle VASP jobs to get better throughput
  - Try multiple sruns with job resizing to bundle jobs
  - The mvasp module is available for bundling many similar VASP jobs
- 1 rank/atom is a good reference when choosing the number of processes to run your VASP jobs

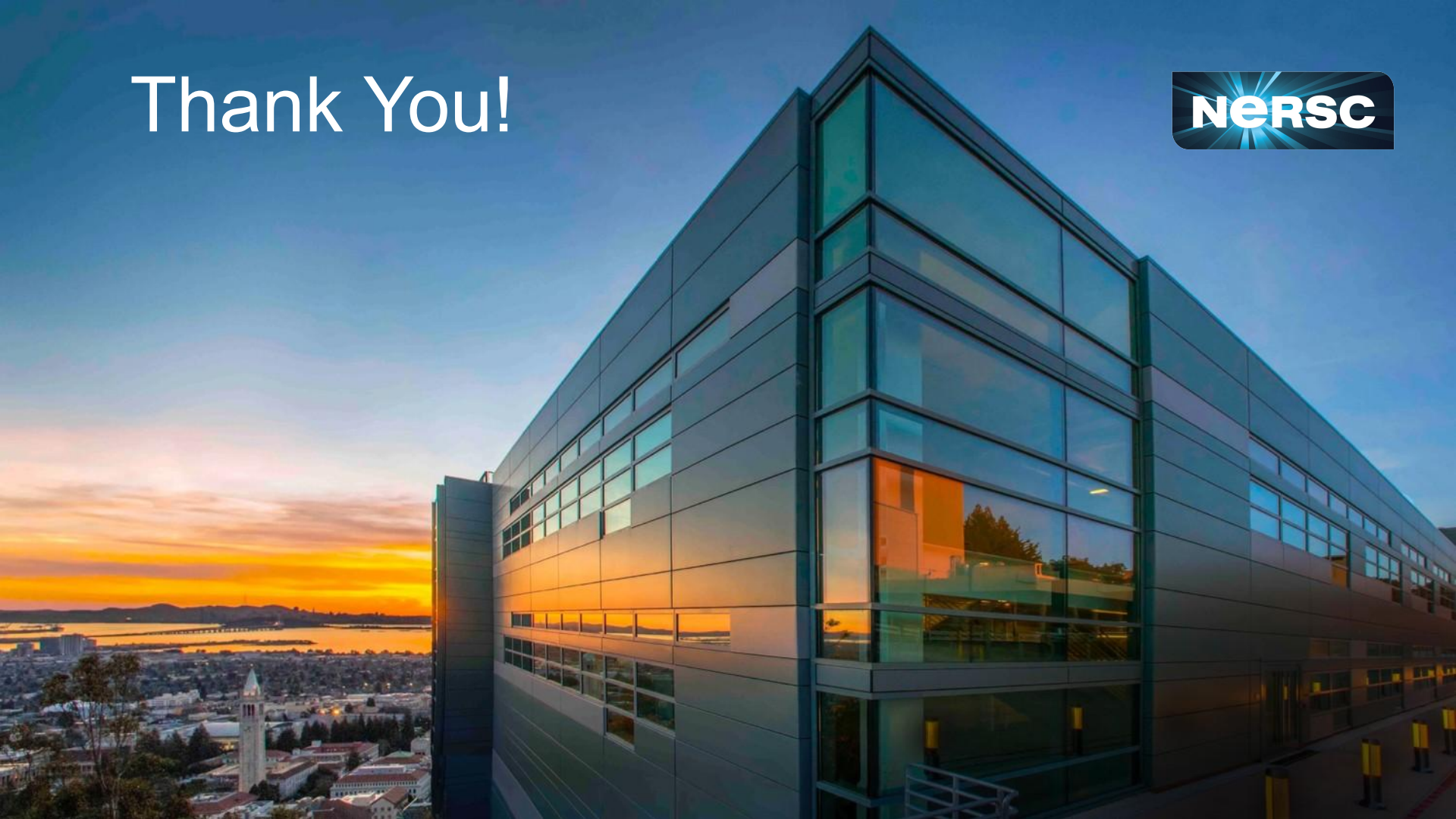
Summary (Cont.)

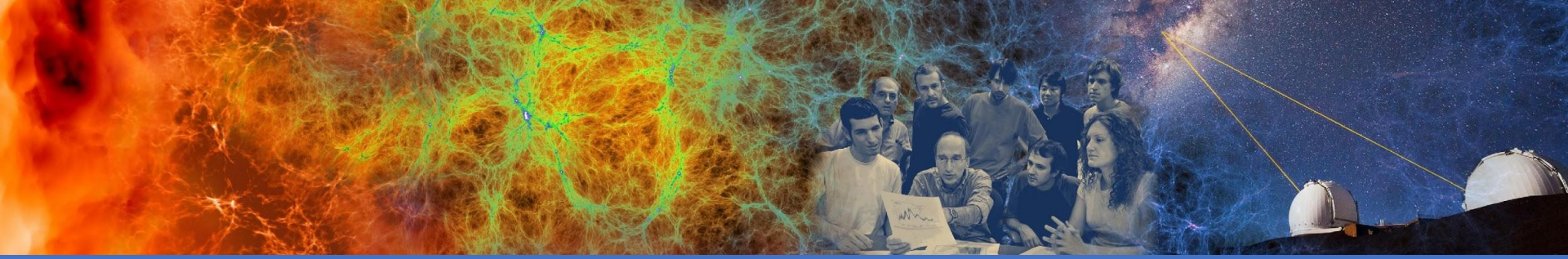
- Running the hybrid MPI + OpenMP VASP on Cori KNL is highly recommended for optimal performance
  - 4 or 8 OpenMP threads per MPI task is recommended
  - Use 64 cores out of 68 on KNL in most cases
- To achieve optimal performance, explicit use of the `srun's --cpu-bind` and `-c` options is recommended to evenly divide the node's CPUs over the MPI tasks and bind tasks to CPUs
- Do not run VASP jobs from your global homes
  - Use cscratch1 and cfs instead

Recommended Readings

- These training slides are available at https://www.nersc.gov/users/training/events/vasp-hands-on-training-june-30-2020/
- Variable-time jobs: https://docs.nersc.gov/jobs/examples/#variable-time-jobs
- MVASP: https://docs.nersc.gov/applications/vasp/#running-multiple-vasp-jobs-simultaneously
- Man pages of `sbatch`, `salloc`, `scancel`, `srun`, `squeue`, `sinfo`, `sqs`, `scontrol`, `sacct`

Thank You!





Hands-on 10:30am - 12:00pm
12:00pm - 1:00pm (Optional)

- Please post questions at here

Use the Reservation vasp_vtj and the Account nintern

- 250 KNL nodes are reserved for the hands-on
  - To run jobs interactively

```
salloc -N 2 -C knl –t 1:00:00 -q regular --reservation=vap_vtj -A nintern
```

  - To run batch jobs

```
#SBATCH -A nintern

#SBATCH --reservation=vasp_vtj
```

- A sample variable-time job script is available at
  - `/global/cscratch1/sd/zz217/vasp_vtj/run.slurm`