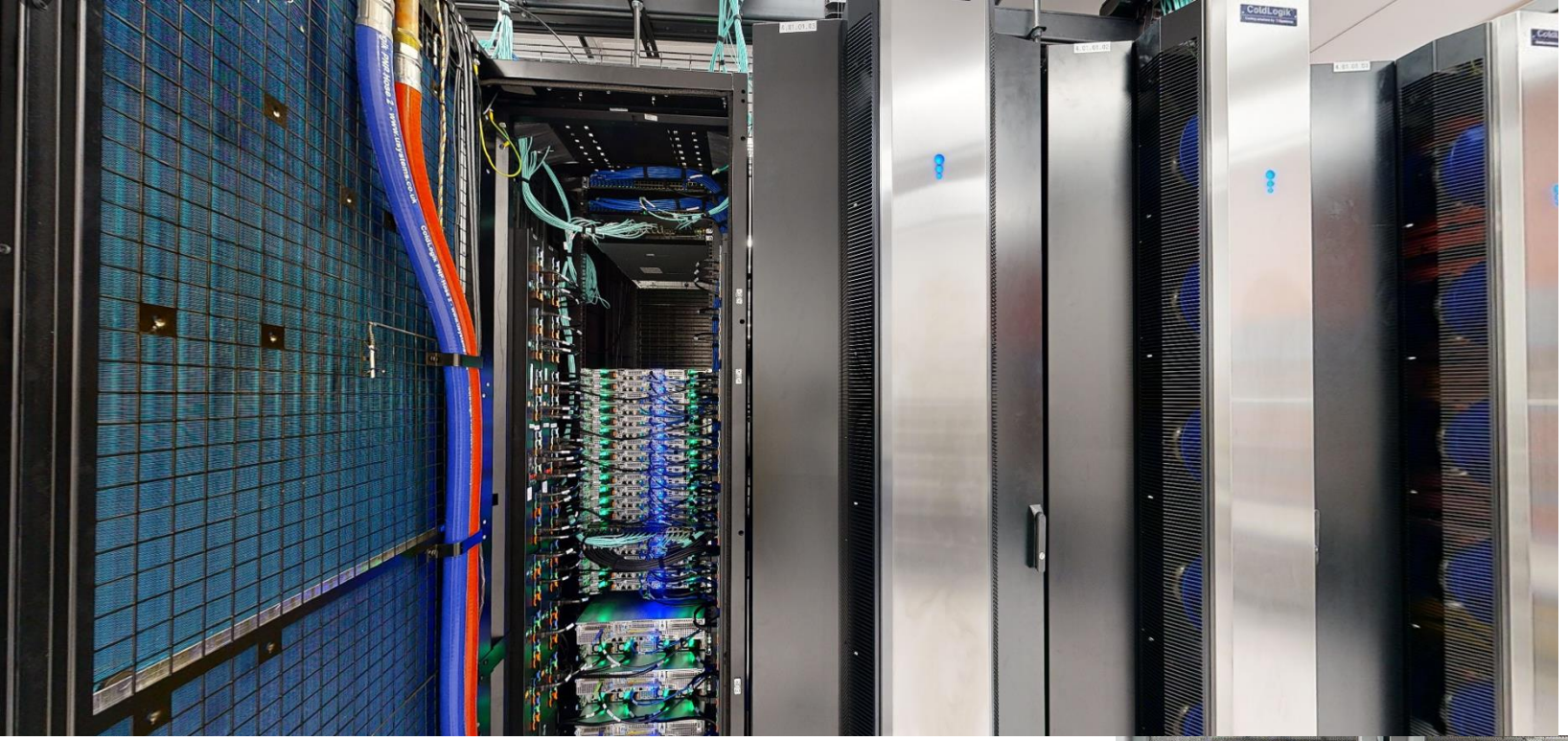


NJIT
New Jersey Institute of Technology

High Performance Computing

Intro to Wulver: Focus on Job Efficiency

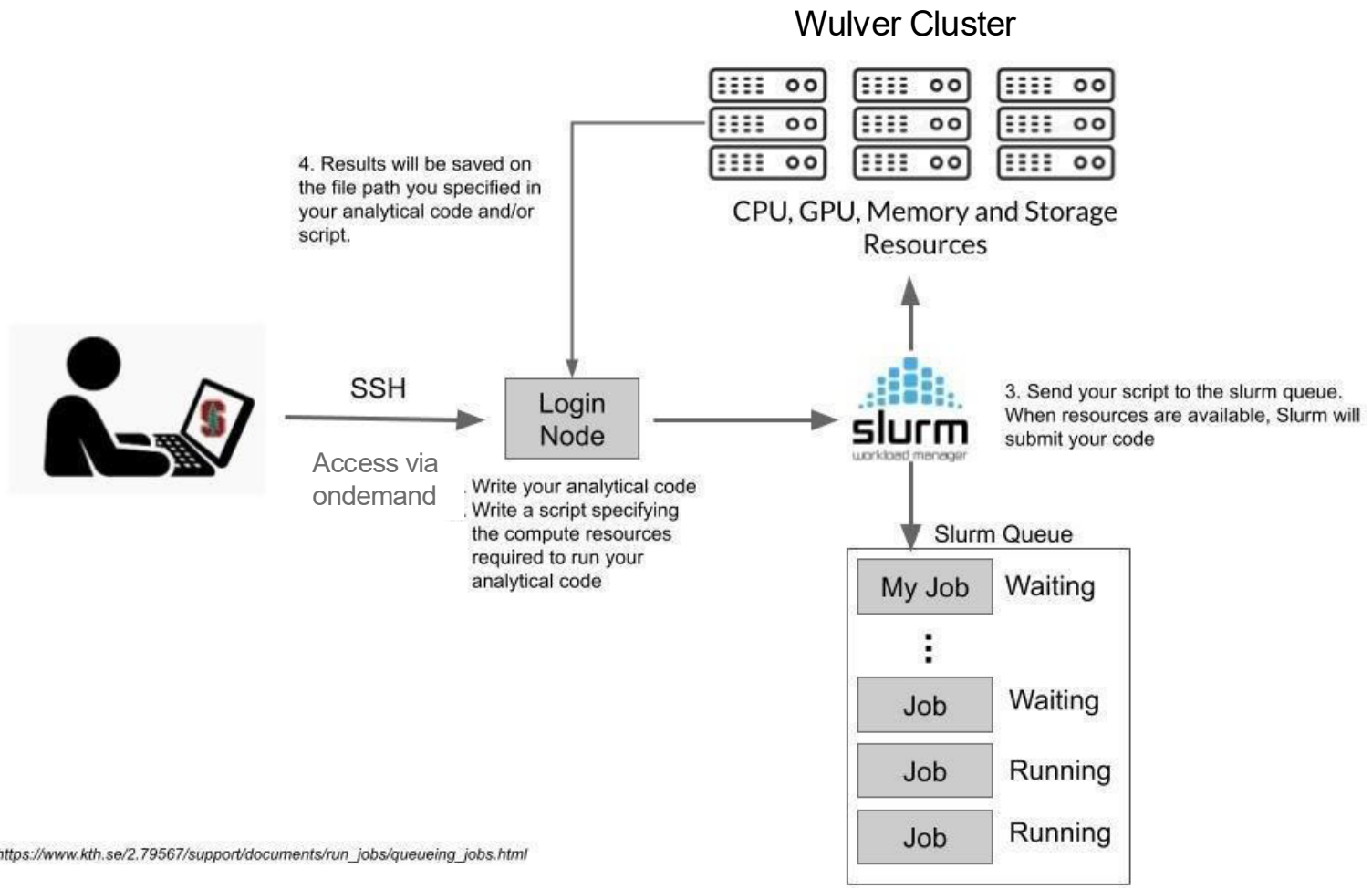
Oct 08, 2025

Outline

- Batch Processing (Summary from previous Slurm webinar)
- Sbatch : Some Examples
- salloc command
- Manage Slurm Jobs
- Job Dependencies
- Job Arrays
- Slurm Switch
- Checkpointing
- Common Problems or Misconceptions
- Reminder and Contact Us

Batch Processing

Why do supercomputers use queuing?



Source: https://www.kth.se/2.79567/support/documents/run_jobs/queueing_jobs.html

sbatch Examples

sbatch Example: Memory

Submit a batch job

```
> sbatch --mem=10G my_work.bash Submitted batch job 44003
```

Options used

- - mem	Amount of reqeusted memory (K|M|G|T)

This Job will allocate 10G of memory

sbatch Example - Node Specification

```
> sbatch --nodes=4 --nodelist=n00[01-02] --exclude=n00[08-09] my.bash
> sbatch --nodes=1 --exclusive my.bash
```

Options used

-x, --exclude	Specific nodes to exclude from the job's allocation (e.g. suspected to be bad)
-w, --nodelist	Specific nodes that must be included in the job's allocation. Additional nodes may be included in the allocation as needed

Default behavior for sharing resources is configurable by partition (see OverSubscribe)

--exclusive	Allocate the job an entire node

For `--exclusive` option , SU charges will be calculated based on 128 cores

sbatch Example - Multiple apps in single script

Submit a batch job

```
> sbatch my_work2.bash
Submitted batch job 44005
```

```
> cat my_work2.bash
```

The ***srun*** commands utilize the resources in the allocation request.

The ***srun*** commands execute desired tasks **WITHIN** the set of requested resource.

They cannot use more/other than what was requested.

```
#!/bin/bash
#SBATCH --ntasks=128
#SBATCH --mem-per-cpu=4G
#SBATCH --time=60
...

srun --ntasks=100 app1 &
srun --ntasks=20  app2 &
srun --ntasks=8   app3 &
wait
bash my_cleanup_script.sh
...
```

Options used

```
--ntasks
--mem-per-cpu
--time
```

Number of tasks

Memory required per CPU

Wall time limit (minutes in our example)

Source: www.schedmd.com

sbatch Example : Requeuing job

```
#!/bin/bash -l
#SBATCH --job-name=dam-break
#SBATCH --output=%x.%j.out
#SBATCH --error=%x.%j.err
#SBATCH --partition=general
#SBATCH --nodes=1
#SBATCH --open-mode=append
#SBATCH --ntasks-per-node=32
#SBATCH --qos=standard
#SBATCH --mem-per-cpu=4G
#SBATCH --account=PI_ucid
#SBATCH --time=3-00:00:00
#SBATCH --requeue
#SBATCH --mail-type=ALL
#SBATCH --mail-user=ab1234@njit.edu

# Load the modules
module load foss/2024a OpenFOAM
source $FOAM_BASH

# Run the job using
requeue_job mpirun interFoam -parallel
```

Append the output to an exiting output file once requeued

Sample job script in /apps/testjobs/requeue

salloc Command

salloc Command

- Used to get a resource allocation and use it **interactively** from your computer terminal
- Typically spawns a shell with various Slurm environment variables set
- Depending upon Slurm configuration (LaunchParameters=use_interactive_step)
  - The shell can be on the login node OR
  - The shell can be an allocated compute node
- Job steps can be launched from the *salloc* shell
- Use `sacct -j [jobid]`

salloc/srun - Multiple (Serial) Job Steps

```
> salloc - n128 bash
salloc: Granted job allocation 17
salloc: Waiting for resource configuration
salloc: Nodes node[00-29] are ready for job

> srun - n100 app1 &

> srun - n20 app2 &

> srun - n8 app3 &

> wait
> exit
```

Job Dependencies

Job Dependencies

Submit sequence of three batch jobs

```
> sbatch --ntasks=1  --parsable  pre_process.bash
45001
> sbatch --ntasks=128 --parsable --dependency=afterok:45001  do_work.bash
45002
> sbatch --ntasks=1  --parsable  --dependency=afterok:45002  post_process.bash
45003
```

Options used

`--ntasks`	Number of tasks and by default the number of cores
`--dependency`	Job dependency

https://hpc.njit.edu/Running_jobs/dependency-jobs/

Job Dependency Options

after:job_id[:job_id...]
This job can begin execution after the specified jobs have begun execution.

afterany:job_id[:job_id...]
This job can begin execution after the specified jobs have terminated (regardless of state).

afterburstbuffer:job_id[:jobid...]
This job can begin execution after the specified jobs have terminated and any associated burst buffer stage out operations have completed.

afternotok:job_id[:job_id...]
This job can begin execution after the specified jobs have terminated in some failed state (non-zero exit code, node failure, timed out, etc).

afterok:job_id[:job_id...]
This job can begin execution after the specified jobs have successfully executed (ran to completion with an exit code of zero).

aftercorr:job_id
A task of this job array can begin execution after the corresponding task ID in the specified job has completed successfully.

singleton
This job can begin execution after any previously launched jobs sharing the same job name and user have terminated

Job Arrays

Job Arrays

- Submit and manage collection of similar jobs easily
- To submit 50 element job array:

```
$ sbatch --array=1-50 -N1 -i my_in_%a -o my_out_%a my.bash
```

- Only supported for batch jobs
- Submit time < 1 second (big bene)
- “%a” in file name mapped to array task ID (1 – 50)
- Default standard output: slurm-<job_id>_<task_id>.out
  - slurm_123_1.out, slurm_123_2.out, etc.

Job Array Environment Variables

- SLURM_JOB_ID – Unique ID for each element
- SLURM_ARRAY_JOB_ID – ID shared by each element
- SLURM_ARRAY_TASK_ID – Task ID
- SLURM_ARRAY_TASK_MIN – Lowest task ID in this array
- SLURM_ARRAY_TASK_MAX – Highest task ID in this array
- SLURM_ARRAY_TASK_COUNT – Count of task IDs in this array

Job Array Example

```
#!/bin/bash
#SBATCH -J myprogram
#SBATCH --partition=general
#SBATCH --qos=standard
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --array=1-30
#SBATCH --output=myprogram%A_%a.out
#SBATCH --error=myprogram%A_%a.err # %A" is replaced by the job ID and "%a"
with the array index
#SBATCH --time=7:59:59

./myprogram input$SLURM_ARRAY_TASK_ID.dat sleep 10
```

/apps/testjobs/job_array

Job Array Use with Dependencies

- Job dependencies support job arrays
  - By individual tasks IDs
  - By an entire job array

```
# Wait for specific job array tasks
$ sbatch --depend=after:123_4 my.job
$ sbatch --depend=afterok:123_4,123_8 my.job2

# Wait for matching element of another job array
$ sbatch --depend=aftercorr:123 my.job3

# Wait for entire job array to complete successfully
$ sbatch --depend=afterok:123 my.job
```

Job Array Use Case

I have an application, **app**, that needs to be run against every line of my dataset. Every line changes how app runs slightly, but I need to compare the runs against each other.

Older, slower way of homogenous batch submission:

```
#!/bin/bash

DATASET=dataset.txt
scriptnum=0

while read LINE; do
    echo "app $LINE" > ${scriptnum}.sh
    sbatch ${scriptnum}.sh
    scriptnum=$(( scriptnum + 1 ))
done < $DATASET
```

Job Array Use Case

Not only is this needlessly complex, it is also slow, as sbatch has to verify each job as it is submitted. This can be done easily with array jobs, as long as you know the number of lines in the dataset. This number can be obtained like so: wc -l dataset.txt in this case lets call it 100.

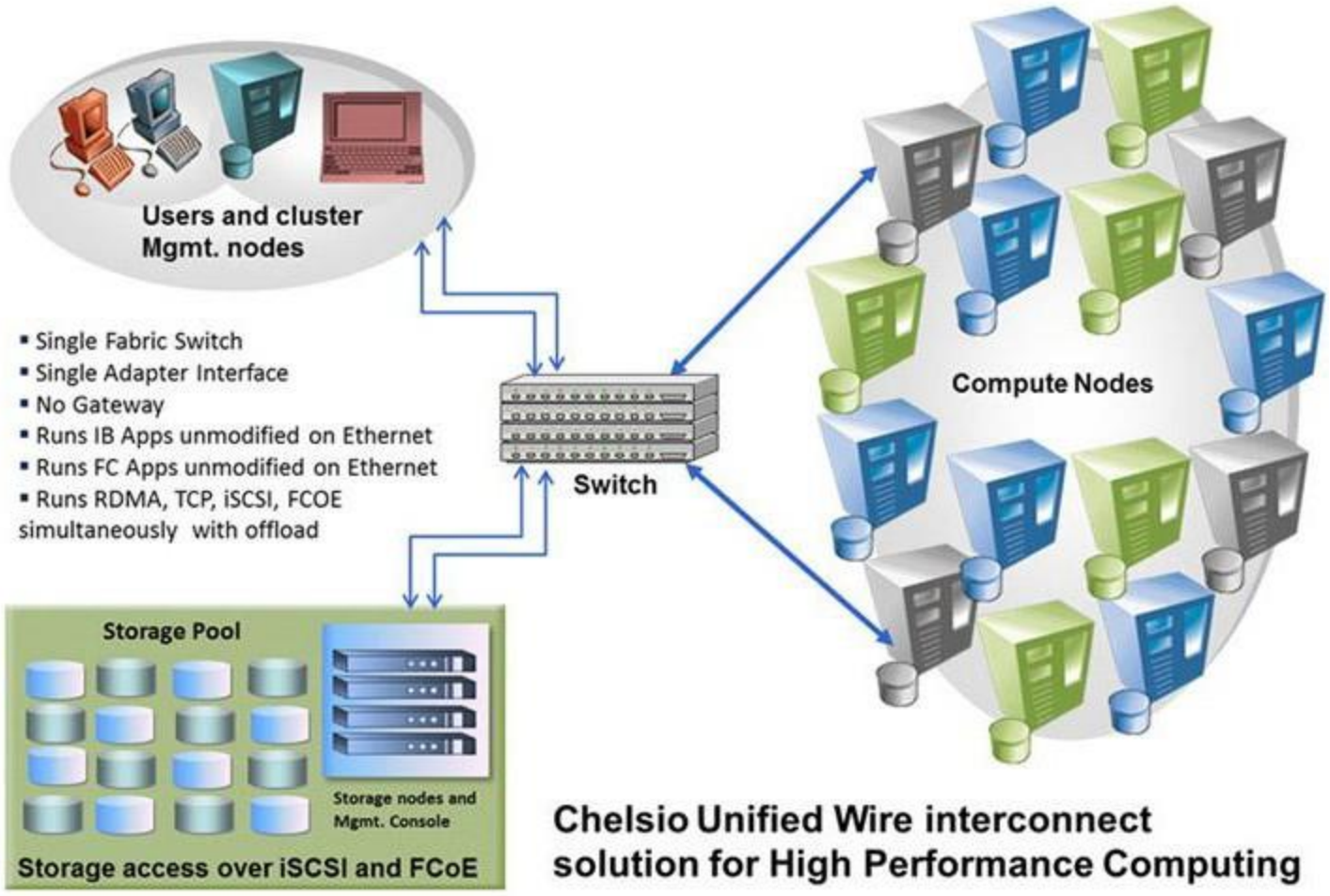
Better way:

```
#!/bin/bash
#SBATCH --array=1-100
srun  app  `sed  -n "${SLURM_ARRAY_TASK_ID}"` dataset.txt
```

Slurm Switch

NJIT

Slurm Switch



Source: https://www.chelsio.com/high_performance_cluster_computing/

Slurm Switch: Job example

```
#!/bin/bash -l
#SBATCH --job-name=dmtcp-test
#SBATCH --output=%x.%j.out # %x.%j expands to slurm JobName.JobID
#SBATCH --error=%x.%j.err # error output
#SBATCH --partition=general #SBATCH --ntasks=64
#SBATCH --qos=standard
#SBATCH --time=30:00    # D-HH:MM:SS
#SBATCH --switch=1@8
```

The #SBATCH --switch=1@8 directive specifies a constraint on network switch usage. This option requests that the job run on nodes connected to at most 1 switch, and it sets a limit of 8 nodes for that switch.

Checkpointing

Checkpointing

- Checkpointing is a process of saving the current state of a running job at certain intervals, so that it can be resumed from that point if it is interrupted or fails unexpectedly.
- Useful for long running process and low priority jobs.
- Most software tools already have their own checkpointing capabilities.

DMTCP: 3rd Party Checkpointing Tool

```
#!/bin/bash -l
#SBATCH --job-name=dmtcp-test
#SBATCH --output=%x.%j.out # %x.%j expands to slurm JobName.JobID
#SBATCH --error=%x.%j.err # error output
#SBATCH --partition=general
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --qos=standard
#SBATCH --time=30:00   # D-HH:MM:SS
###############################################################################
module purge
module load wulver
module load DMTCP
###############################################################################
source start_coordinator.sh
###############################################################################
start_coordinator -i 60 # checkpointing occurs every 60s
###############################################################################
# 2. Launch application
# 2.1. If you use mpiexec/mpirun to launch an application, use the following
#      command line:
#        $ dmtcp_launch --rm mpiexec <mpi-options> ./<app-binary> <app-options>
# 2.2. If you use PMI1 to launch an application, use the following command line:
#        $ srun dmtcp_launch --rm ./<app-binary> <app-options>
# Note: PMI2 is not supported yet.
###############################################################################
dmtcp_launch -j ./test.py
```

Submit the job using
`sbatch dmtcp.submit.sh`

DMTCP: 3rd Party Checkpointing Tool

```
#!/bin/bash -l
#SBATCH --job-name=dmtcp-restart
#SBATCH --output=%x.%j.out # %x.%j expands to slurm JobName.JobID
#SBATCH --error=%x.%j.err # error output
#SBATCH --partition=general
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --qos=standard
#SBATCH --time=30:00   # D-HH:MM:SS
##########################################################################
module purge
module load wulver
module load DMTCP
##########################################################################
source start_coordinator.sh
##########################################################################
start_coordinator -i 60 # checkpointing occurs every 60s
##########################################################################
./dmtcp_restart_script.sh
```

Submit the job using
`sbatch dmtcp.restart.sh`

/apps/testjobs/DMTCP

Checkpointing in PyTorch

Saving a Checkpoint in PyTorch

```
torch.save({
        'epoch': epoch,
        'model_state_dict': model.state_dict(), 'optimizer_state_dict':
        optimizer.state_dict(), 'loss': avg_loss,
}, checkpoint_path)
```

Loading a Checkpoint in PyTorch

```
if os.path.exists(checkpoint_path): checkpoint =
        torch.load(checkpoint_path)
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict']) start_epoch =
        checkpoint['epoch'] + 1
```

Common Problems or Misconceptions

If I allocate more CPUs to my job, my software will use them

Example

```
#!/bin/bash -l
#SBATCH --job-name=python
#SBATCH --output=%x.%j.out # %x.%j expands to slurm JobName.JobID
#SBATCH --error=%x.%j.err # error output
#SBATCH --partition=general
#SBATCH --ntasks=4
#SBATCH --qos=standard
#SBATCH --time=30:00   # D-HH:MM:SS
###############################################################################

Module load foss/2025a Python
srun python test.py
```

- This job will run the python script 4 times if the parallel feature is not enabled in the python script. Use

`python test.py` or `srun -n1 python test.py`

- Use `mpi4py` or `Parsl` for parallel programming in Python

https://mpi4py.readthedocs.io/en/stable/tutorial.html

https://parsl.readthedocs.io/en/stable/quickstart.html

My jobs fail when using threads

Example of Wrong Job Script

```
#!/bin/bash -l
#SBATCH -J gmx-test
#SBATCH -o %x.%j.out
#SBATCH -e %x.%j.err
#SBATCH --partition=gpu
#SBATCH --qos=standard
#SBATCH --time 72:00:00    # Max 3 days
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --gres=gpu:4
###############################################################################

module purge
module load wulver
module load foss/2025a GROMACS/2025.2-CUDA-12.8.0

gmx grompp -f run.mdp -c npt2.gro -r npt2.gro -p topol.top -o run.tpr
srun gmx_mpi mdrun -deffnm run -cpi run.cpt -v -ntomp 2 -pin on -tunepme -dlb yes -noappend
```

- This job will will fail because you did not specify `--cpus-per-task`
- This job will use 128X2=256 CPUs therefore you need to use `--nodes=2`

Solution

```
#!/bin/bash -l
#SBATCH -J gmx-test
#SBATCH -o %x.%j.out
#SBATCH -e %x.%j.err
#SBATCH --partition=gpu
#SBATCH --qos=standard
#SBATCH --time 8:00:00          # Max 3 days
#SBATCH --nodes=1

#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=2
#SBATCH --gres=gpu:4
###############################################################################

OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
module purge && module restore
module load foss/2025a GROMACS/2025.2-CUDA-12.8.0

gmx grompp -f run.mdp -c npt2.gro -r npt2.gro -p topol.top -o run.tpr
srun gmx_mpi mdrun -deffnm run -cpi run.cpt -v -ntomp $SLURM_CPUS_PER_TASK -pin on -tunepme -dlb
yes -noappend
```

This job will launch using 64 cores with 2 threads per core.

Solution

```
#!/bin/bash -l
#SBATCH -J gmx-test
#SBATCH -o %x.%j.out
#SBATCH -e %x.%j.err
#SBATCH --partition=gpu
#SBATCH --qos=standard
#SBATCH --time 8:00:00        # Max 3 days
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=2
#SBATCH --gres=gpu:4
##############################################################################

module purge module load wulver
module load foss/2025a GROMACS/2025.2-CUDA-12.8.0

gmx grompp -f run.mdp -c npt2.gro -r npt2.gro -p topol.top -o run.tpr
srun gmx_mpi mdrun -deffnm run -cpi run.cpt -v -ntomp $SLURM_CPUS_PER_TASK
 -pin on -tunepme -dlb yes -noappend
```

This job will launch using 128 cores with 2 threads per core.

My jobs fail due to out of memory error

Out of Memory Issue

- The Out-of-Memory (OOM) Killer is a Linux kernel process that activates when a compute node is critically low on physical memory.
- **Exit Code:** The job failed with an exit code of 137. (This means it was terminated by SIGKILL, signal 9).

```
slurmstepd: error: Job 123456 exceeded memory limit (16500 MB > 16384 MB), being killed
slurmstepd: error: Out of Memory
```

- seff <Your-Job-ID>
- **What to look for:** A "Memory Efficiency" percentage over 100% and a clear statement that the job was killed by OOM

Solution

- Increase the memory in `--mem`.
- Use a memory profiler to monitor memory usage during code execution
- Use **perf, Valgrind, Memray, memory_profiler**

Reminder

Wulver Monthly Maintenance

- Wulver will be temporarily out of service for maintenance once a month, specifically on the 2nd Tuesday, to perform updates, repairs, and upgrades.
- During the maintenance period, the logins will be disabled
- Jobs that do not end before the maintenance window begins will be held until the maintenance is complete

Open Office Hours

- Date: Every Wednesday and Friday
- Time: 2:00–4:00 p.m.
- Location: GITC 2404
- Meet with our student consultants and ask any questions you have about using HPC resources.
- There's no need to create a ticket in advance.

Resources to get your questions answered

Getting Started: Access to Wulver

List of Software: Wulver Software

HOW TOs: Conda Documentation

Installing Python packages via Conda

Running Jobs: Jobs

Access to OnDemand Open OnDemand

MIG Information MIG

Contact: Please visit HPC Contact

Open a ticket: email to hpc@njit.edu

Consult with Research Computing Facilitator: Facilitator Calendar Appointment

System updates

- Read Message of the Day on login
- Visit NJIT HPC News

NJIT

hpc@njit.edu

hpc.njit.edu