NJIT
New Jersey Institute of Technology

High Performance Computing

NJIT High Performance Computing (HPC) Workshop: Job Arrays and Advanced Submission Techniques

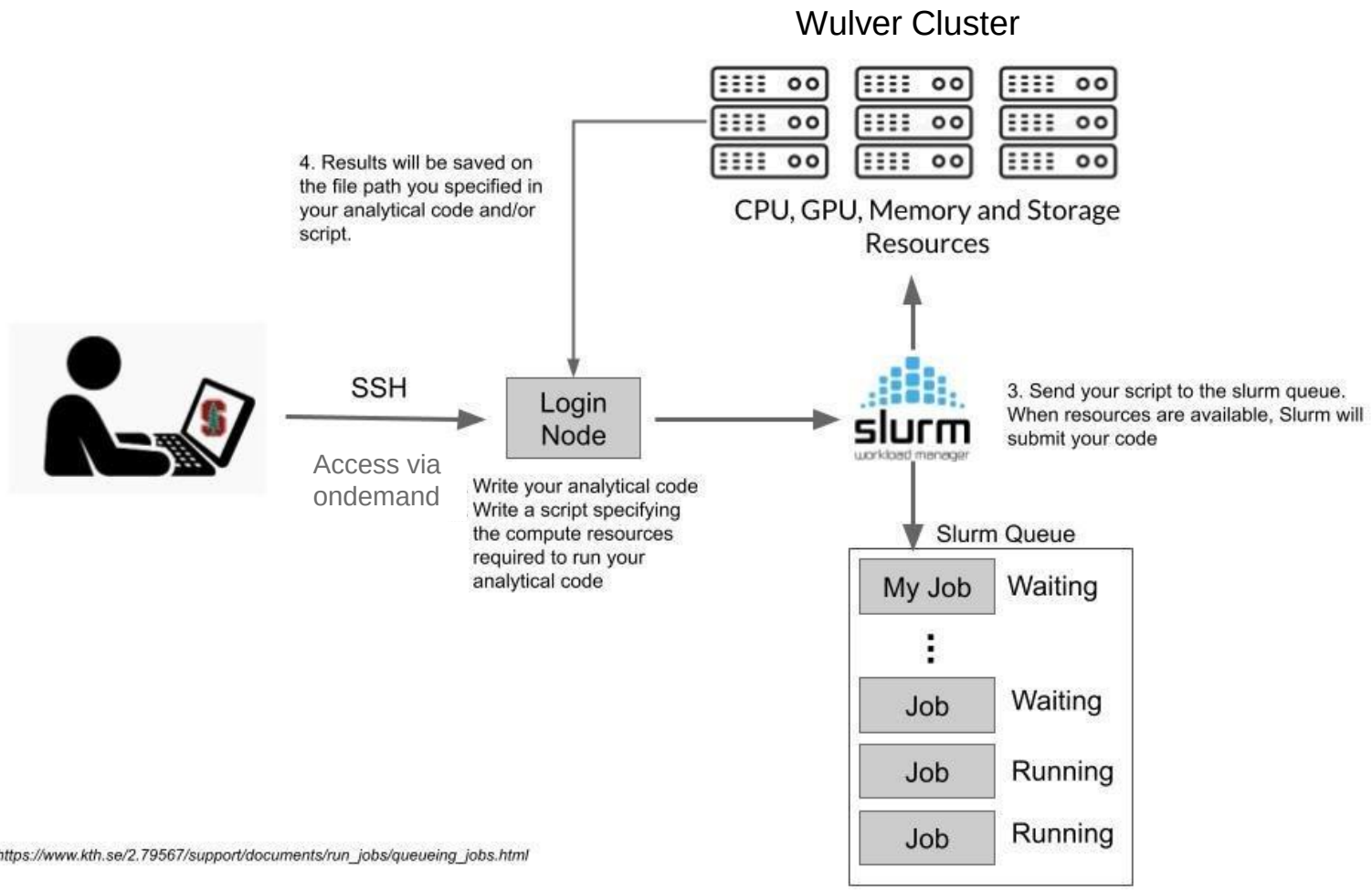
Nov 20, 2024

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

HPC Partitions

- Example of SU charges: (20 cores with 4 GPUs for 8 hours)
- SU = 20 x 8 x 3 = 480

Partition	Nodes	Cores /Node	CPU	GPU	Memory	SU charge
`--partition=general`	100	128	2.5G GHz AMD EPYC 7763 (2)	NA	512 GB	1 SU per hour per cpu
`--partition=debug`	1	4	2.5G GHz AMD EPYC 7763 (2)	NA	512 GB	No charges, must be used with `--qos=debug`
`--partition=gpu`	25	128	2.0 GHz AMD EPYC 7713 (2)	NVIDIA A100 GPUs (4)	512 GB	3 SU per hour per cpu
`--partition=bigmem`	2	128	2.5G GHz AMD EPYC 7763 (2)	NA	2 TB	1.5 SU per hour per cpu

https://hpc.njit.edu/Software/slurm/slurm/

QoS

- Standard Priority (`--qos=standard`)
  - Faculty PIs are allocated 300,000 Service Units (SU) per year on request at no cost
  - Additional SUs may be purchased at a cost of $0.005/SU.
  - The minimum purchase is 50,000 SU ($250)
  - Wall time maximum - 72 hours
- Low Priority (`--qos=low`)
  - Not charged against SU allocation
  - Wall time maximum - 72 hours
  - Jobs can be preempted by those with higher and standard priority jobs when they are in the queue
- High Priority (`--qos=high_$PI`)
  - Only available to owners/investors
  - Not charged against SU allocation
  - Wall time maximum - 72 hours – can be increased based on PI's request
  - Only available to contributors
  - Use `listqos` command to check high priority access

sbatch Examples

sbatch Example: Memory

Submit a batch job

```
> sbatch --mem=1000 my_work.bash Submitted batch job 44003
```

Options used

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

This Job will allocate 1000M of memory

SU Charges for Memory Allocation

```
> srun --partition=gpu --nodes=1 --ntasks-per-node=8 --mem=160G --
qos=standard --gres=gpu:1 --time=2:00:00 --pty bash
```

What will be SU Charge for this job?

SU Charges for Memory Allocation

```
> srun --partition=gpu --nodes=1 --ntasks-per-node=8 --mem=160G --
qos=standard --gres=gpu:1 --time=2:00:00 --pty bash
```

- Number of CPUs - 8
- Memory Requested - 160G
- Based on 4G/core, number of equivalent cores - 160/4 = 40

SU Charges for Memory Allocation

```
> srun --partition=gpu --nodes=1 --ntasks-per-node=8 --mem=160G --
qos=standard --gres=gpu:1 --time=2:00:00 --pty bash
```

- Number of CPUs - 8
- Memory Requested - 160G
- Based on 4G/core, number of equivalent cores - 160/4 = 40
- SU Charges - 40 X 2 X 3 = 240

We will use whichever is the maximum

- Use `slurm_jobid` command to check SU usage

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

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 `saact -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 --exclusive (if want simultaneous) -n100 app1 &
>srun --exclusive (if want simultaneous) -n20 app2 &
>srun --exclusive (if want simultaneous) -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://slurm.schedmd.com/sbatch.html#OPT_dependency

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=71:59:59

./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

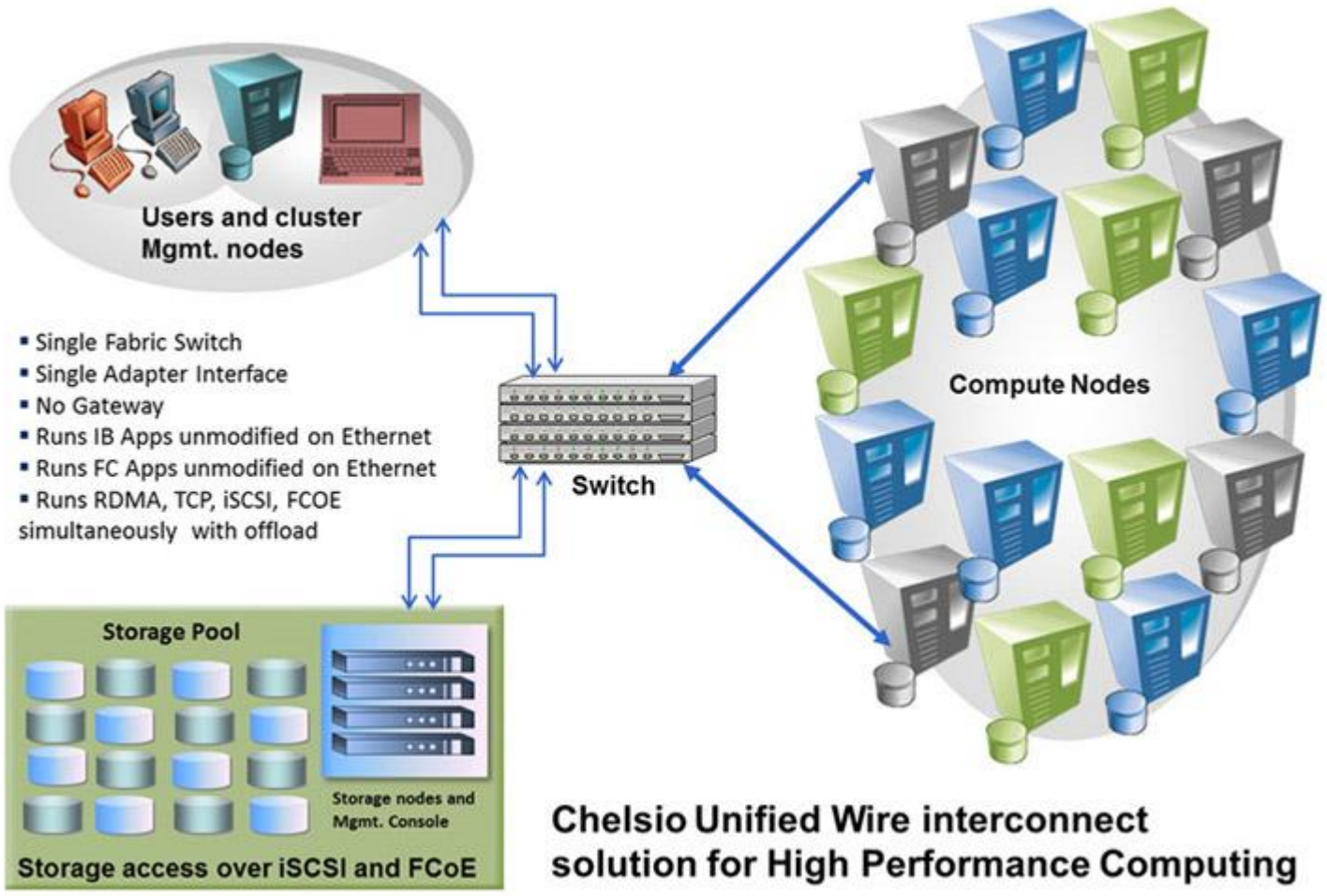
Not only is this needlessly complex, it is also slow, as sbatch has to verify each job as it is submitted. This can be done easily with array jobs, as long as you know the number of lines in the dataset. This number can be obtained like so: wc -l dataset.txt in this case lets call it 100.

Better way:

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

Slurm Switch

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

NJIT

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

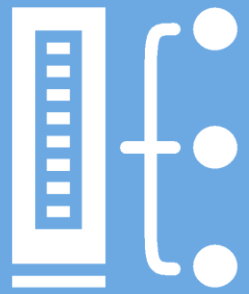
Checkpointing in PyTorch

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/2023b 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 run slower on HPC

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/2021b GROMACS/2021.5-CUDA-11.4.1

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 use 128X2=256 CPUs which would overload the node as the node has 128 cores

Solution

- **Check the CPU load. Use `checkload` command**
- **Immediately cancel the job and modify the 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=64
#SBATCH --cpus-per-task=2
#SBATCH --gres=gpu:4
################################################################################

module purge
module load wulver
module load foss/2021b GROMACS/2021.5-CUDA-11.4.1

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 launch using 64 cores with 2 threads per core.

Solution

- **Check the CPU load. Use `checkload` command**
- **Immediately cancel the job and modify the 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 --ntasks-per-node=128
#SBATCH --cpus-per-task=2
#SBATCH --gres=gpu:4
################################################################################

module purge
module load wulver
module load foss/2021b GROMACS/2021.5-CUDA-11.4.1

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 launch using 128 cores with 2 threads per core.

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

Request Software: HPC Software Installation

Access to OnDemand Open OnDemand

Contact: Please visit HPC Contact

Open a ticket: email to hpc@njit.edu

Consult with Research Computing Facilitator: HPC User Assistance

System updates

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

NJIT

hpc@njit.edu

hpc.njit.edu