

Introduction to Wulver: Job Submission and Resource Requests

Fundamentals of the SLURM workload manager on NJIT's Wulver cluster

Abhishek Mukherjee

Research Computing Facilitator, High Performance Computing, NJIT



Today's Agenda

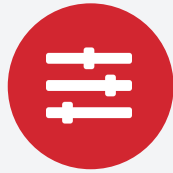
From your first job script to a well-sized request



01

Wulver & SLURM

Hardware recap, what SLURM does, and the login vs. compute node rule



02

Requesting Resources

Partitions, QoS priority levels, job limits, GPUs and MIG slices



03

Service Units

How SUs are charged per partition and how to check your group's balance



04

Submit & Monitor

Batch scripts, interactive sessions, monitoring tools and common mistakes

Wulver at a Glance

100

General CPU nodes

128 cores · 512 GB RAM
each

25

GPU nodes

4× NVIDIA A100 80 GB each

2

Big-memory nodes

2 TB RAM each

1 PB

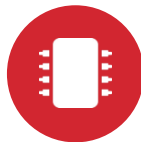
Parallel storage

Arcastream, up to 20 GB/s



100 Gb InfiniBand

High-throughput, low-latency
network



16,256 CPU cores

Across general, GPU and bigmem
nodes



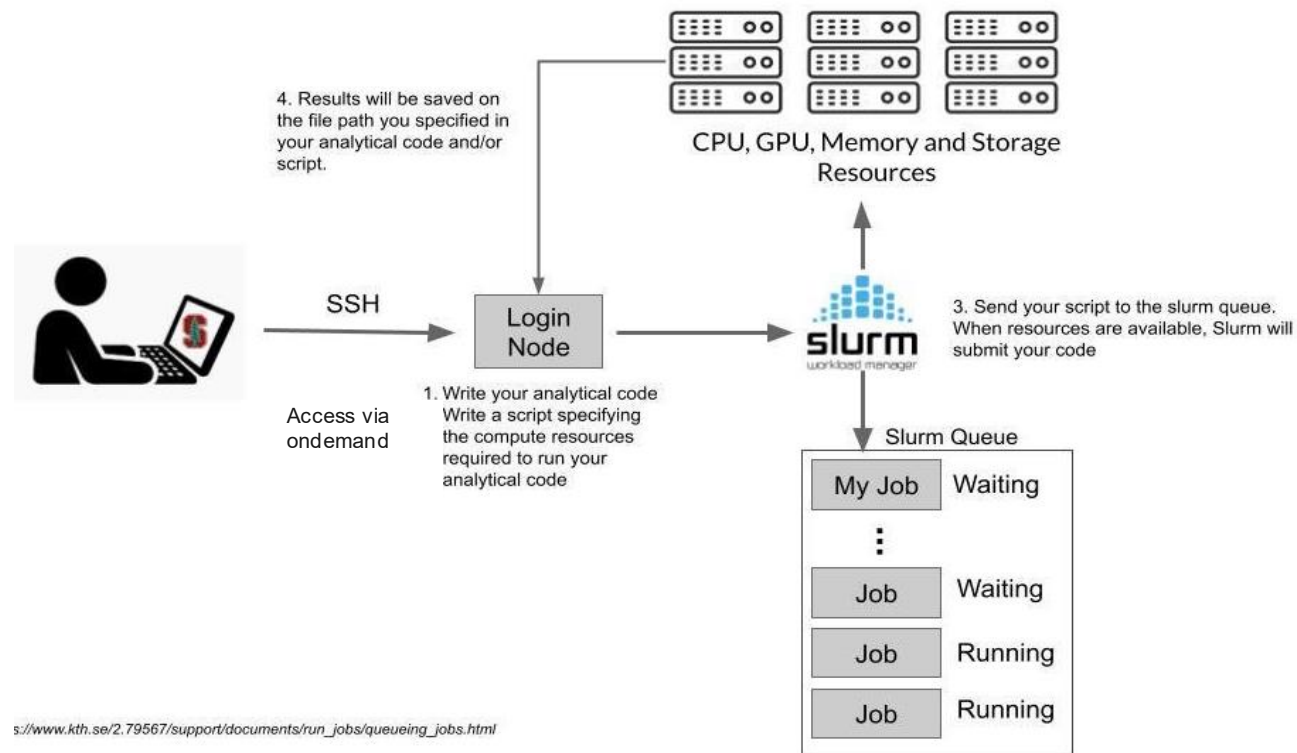
SLURM scheduler

Every job goes through SLURM

What is SLURM?

Simple Linux Utility for Resource Management

The open-source workload manager and job scheduler Wulver uses to allocate CPUs, GPUs, memory and time across the cluster.



[s://www.kth.se/2.79567/support/documents/run_jobs/queueing_jobs.html](https://www.kth.se/2.79567/support/documents/run_jobs/queueing_jobs.html)



Think air traffic control

1

You file a flight plan

Your job script lists what you need

2

The tower finds a runway

SLURM waits for free nodes that fit

3

You take off

The job runs on a compute node

Login Node vs. Compute Nodes

The first rule of using Wulver



Login node

Entry point to Wulver

- ✓ Edit scripts and manage files
- ✓ Load modules, compile small code
- ✓ Submit and monitor jobs
- ✗ **Run simulations or heavy Python**



Compute nodes

Where real work runs

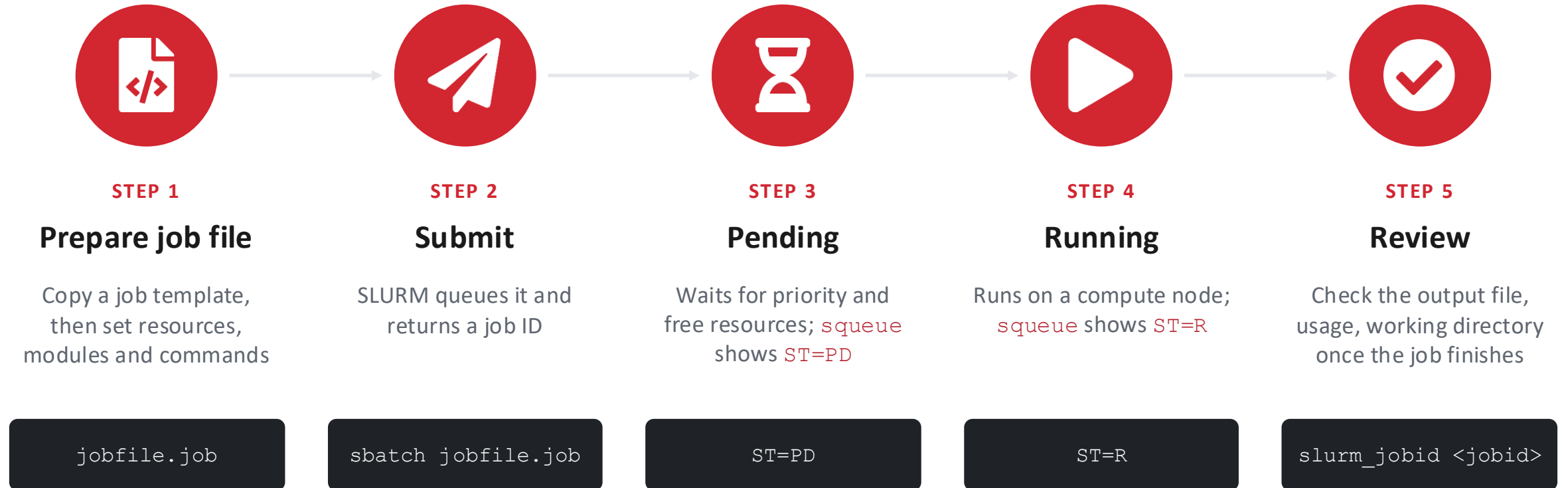
- ✓ All real computation
- ✓ Batch jobs with `sbatch`
- ✓ Live shells with `interactive`
- ✓ CPUs, GPUs and memory reserved for you



Heavy work on the login node slows Wulver down for every logged-in user. Always go through SLURM.

The Life of a SLURM Job

From script to results



PART 1

Requesting Resources

Partitions, QoS, limits, GPUs and Service Units



Partitions: Choose with --partition

Pick the partition that matches your job type

Partition	Best for	Nodes	Cores/node	Memory/node	GPUs	Rules
general	CPU production jobs	100	128	512 GB	—	—
gpu	GPU-accelerated production	25	128	512 GB	4 × A100 80G, (2×10G, 20G, 40G MIG) (4 Nodes)	—
bigmem	Memory-intensive jobs	2	128	2 TB	—	1 job/user; > 500 GB
debug	Short CPU tests	1	4	16 GB	—	--qos=debug; 1 job/user
debug_gpu	Short GPU tests	1	4	16 GB	8×10G, 4×20G, 4×40G MIG	--qos=debug; 1 job/user

```
#SBATCH --partition=general
```

Requesting a GPU on a non-GPU partition fails immediately or is held forever.

Priority Levels: Choose with --qos

QoS decides your priority and whether you are charged SUs



--qos=standard

Charges SUs

Priority: normal

SU charge: yes, from the PI's 300,000 SU/year

Walltime: 72 h · everyone

Extra SUs: \$0.005/SU (min \$250); no carryover



--qos=low

Free

Priority: lowest; preempted by standard or high

SU charge: none

Walltime: 72 h · everyone



--qos=high_<PI_UCID>

Free

Priority: highest

SU charge: none

Walltime: 72 h, extendable on PI request

Access: contributors only; check with `listqos`



--qos=debug

Free

Limits: 1 job, 4 CPUs, 16 GB memory

Partitions: debug or debug_gpu only

Walltime: 8 h · everyone

Job Limits to Plan Around

Know the rules before you submit

72 h

Maximum walltime for
standard, low

8 h on the debug partitions

`--time=3-00:00:00` is the most
you can ask for



Longer requests are rejected

Walltime over 3 days is not allowed; checkpoint and resubmit long runs. **Ask only for the time you need:** padding a request toward 72 h makes the job wait longer to start



When SUs run out

Standard and high QoS are blocked, but low QoS still works



Preemption

Low-priority jobs can be stopped by standard or high jobs



Maintenance holds

Maintenance runs on the 2nd Tuesday of every month and logins are disabled. Jobs whose walltime overlaps the window are held until it ends; reduce `--time` so the job finishes before it starts

Service Units (SUs): Wulver's Currency

Every job spends SUs based on what it requests and how long it runs



1 SU = 1 CPU-hour or 4 GB of RAM for one hour

300,000

SUs per PI per year

Free on request, used through standard QoS. Need more? PIs can request extra SUs through an [allocation proposal](#), reviewed 4 times a year

1 balance

Shared by the PI's group

Allocated per PI account, not per student. Heavy use drains it for everyone

Yearly

Resets each fiscal year

Groups that run out can buy more SUs or invest for high-priority access

GPUs and MIG: Right-Size Your GPU

Multi-Instance GPU splits one A100 into isolated slices

Full A100

`--gres=gpu:1`

GPU memory

80 GB

16 SU/hr

40g slice

`--gres=gpu:a100_40g:1`

40 GB

8 SU/hr

20g slice

`--gres=gpu:a100_20g:1`

20 GB

4 SU/hr

10g slice

`--gres=gpu:a100_10g:1`

10 GB

2 SU/hr



Why MIG?

- Dedicated memory, cores, cache and bandwidth
- Fits jobs that don't need a full 80 GB
- Costs a fraction of the SUs
- Leaves the rest of the GPU for others
- Available on `gpu` and `debug_gpu`

GPU charge = 16 × (GPU memory ÷ 80 GB) SU/hr, added to the CPU/memory charge

How SUs Are Charged by Partition

You pay for whichever is larger: cores or memory

Partition	SU charged per hour	Default memory
general	$\text{MAX}(\text{CPUs}, \text{RAM} \div 4 \text{ GB})$	4 GB per core
gpu	$16 \times (\text{GPU mem} \div 80 \text{ GB}) + \text{MAX}(\text{CPUs}, \text{RAM} \div 4 \text{ GB})$	4 GB per core
bigmem	$\text{MAX}(1.5 \times \text{CPUs}, 1.5 \times \text{RAM} \div 16 \text{ GB})$	16 GB per core
debug, debug_gpu	No charge (requires <code>--qos=debug</code>)	—



Leave out `--mem` and you pay per core

Asking for more memory than the default raises the charge even if you use the same number of cores.

SU Examples: CPU and GPU Jobs

Same formula, very different costs per hour

GENERAL

4 SU/hr

```
--ntasks=4
```

$$\text{MAX}(4, 16 \div 4) = 4$$

Default 4 GB per core

GENERAL

16 SU/hr

```
--ntasks=4  
--mem=64G
```

$$\text{MAX}(4, 64 \div 4) = 16$$

Memory drives the cost:
4x

GPU

20 SU/hr

```
--ntasks=4  
--gres=gpu:1
```

$$4 + 16 = 20$$

A full A100 adds 16 SU/hr

BIGMEM

48 SU/hr

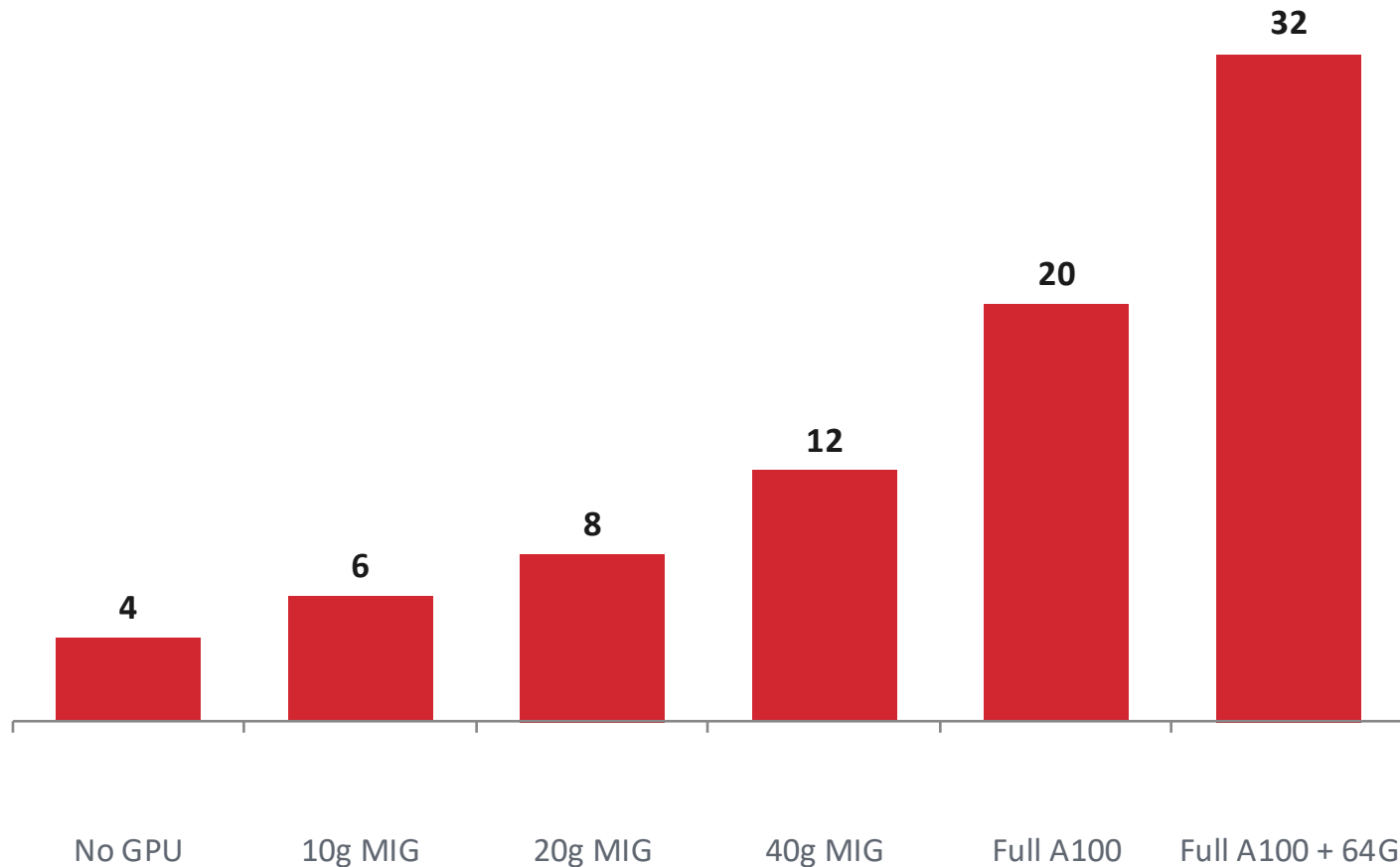
```
--ntasks=16  
--mem=512G
```

$$\text{MAX}(24, 48) = 48$$

Memory drives the cost:
 $1.5 \times 512 \div 16$

SU Examples: GPU Jobs

SU per hour for a 4-CPU job on the gpu partition



$$4 \text{ CPUs} + 16 \times (\text{GPU GB} \div 80)$$



Slices are cheap

A 10g slice is 6 SU/hr, versus 20 for a full A100



Memory adds on top

--mem=64G pushes a full-GPU job to 32 SU/hr



Scale with care

Request extra GPUs only when your code scales

Check Your Balance with quota_info

Group SUs, your own usage, and storage in one command

```
$ quota_info $LOGNAME
Usage for account: xy1234
  SLURM Service Units (CPU Hours): 277557 (300000 Quota)
    User ab1234 Usage: 1703 CPU Hours (of 277557 CPU Hours)
  PROJECT Storage: 867 GB (of 2048 GB quota)
    User ab1234 Usage: 11 GB (No quota)
  SCRATCH Storage: 791 GB (of 10240 GB quota)
    User ab1234 Usage: 50 GB (No quota)
  HOME Storage ab1234 Usage: 0 GB (of 50 GB quota)
```



Account = PI's UCID

xy1234 is what goes in `--account` and `interactive -a`



Group SU balance

277,557 of 300,000 SUs already used by the group



Your share

1,703 of those CPU hours are yours



Storage quotas

HOME 50 GB, PROJECT 2 TB and SCRATCH 10 TB per group

PART 2

Submitting Jobs

Batch scripts, interactive sessions and monitoring



Essential #SBATCH Directives

The flags you will use in almost every job

Directive	What it controls	Example
<code>--job-name / --output / --error</code>	Job name and log files (%x = name, %j = ID)	<code>--output=%x.%j.out</code>
<code>--input</code>	File read as standard input	<code>--input=data.txt</code>
<code>--partition</code>	Which node type runs the job	<code>--partition=general</code>
<code>--qos</code>	Priority level and SU charging	<code>--qos=standard</code>
<code>--account</code>	Whose SUs are charged (PI's UCID)	<code>--account=<PI_UCID></code>
<code>--nodes / --ntasks-per-node</code>	Nodes, and tasks on each node	<code>--nodes=1</code>
<code>--ntasks / --cpus-per-task</code>	Processes, and threads per process	<code>--ntasks=1</code>
<code>--mem / --mem-per-cpu</code>	Memory (default 4000M per core)	<code>--mem-per-cpu=4000M</code>
<code>--time</code>	Walltime limit, D-HH:MM:SS (max 3 days)	<code>--time=59:00</code>
<code>--gres</code>	GPUs or MIG slices	<code>--gres=gpu:1</code>
<code>--mail-type / --mail-user</code>	Email on BEGIN, END, FAIL or REQUEUE	<code>--mail-type=END, FAIL</code>

Anatomy of a Batch Job Script

submit.sh: a text file of instructions for SLURM

```
#!/bin/bash -l
#SBATCH --job-name=job_name
#SBATCH --output=%x.%j.out
#SBATCH --error=%x.%j.err
#SBATCH --partition=general
#SBATCH --qos=standard
#SBATCH --account=<PI_UCID>
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=59:00
#SBATCH --mem-per-cpu=4000M

./myexe <input/output options>
```

1

Shebang

`#!/bin/bash -l` starts a login shell so modules work

2

Job identity

Name plus `.out/.err` logs; `%x` = name, `%j` = job ID

3

Where and who pays

`--partition`, `--qos` and `--account` (the PI's UCID)

4

How much, how long

`--nodes`, `--ntasks`, `--mem-per-cpu`, `--time`
(D-HH:MM:SS)

5

What to run

Your program; use `srun` for parallel runs

CPU Job Patterns

Don't request multiple cores unless your code runs in parallel



Single core

Serial code, which is most scripts

```
#SBATCH --ntasks=1
./myexe
```



MPI across cores

Code built with MPI

```
#SBATCH --ntasks-per-node=8
srun ./myexe
```



Threads (OpenMP)

Multithreaded code

```
#SBATCH --ntasks-per-node=8
OMP_NUM_THREADS=$SLURM_NTASKS ./myexe
```



Hybrid MPI + threads

For example GROMACS

```
#SBATCH --ntasks=64
#SBATCH --cpus-per-task=2
srun gmx_mpi mdrun -ntomp $SLURM_CPUS_PER_TASK
```

Submitting GPU Jobs

gpu_submit.sh on the gpu partition

```
#!/bin/bash -l
#SBATCH --job-name=gpu_job
#SBATCH --output=%x.%j.out
#SBATCH --error=%x.%j.err
#SBATCH --partition=gpu
#SBATCH --qos=standard
#SBATCH --account=<PI_UCID>
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --gres=gpu:a100_10g:1
#SBATCH --time=59:00
#SBATCH --mem-per-cpu=4000M

module load CUDA
nvidia-smi -l 10 --format=csv \
  --query-gpu=timestamp,memory.used \
  > gpu_usage.log &
./myexe <input/output options>
```



Partition and **--gres** together

`--partition=gpu` plus `--gres=gpu:1` for one full A100



Start with a MIG slice

`gpu:a100_10g:1`; change 10g to 20g or 40g as needed



Keep CPU counts low

Most GPU jobs need few CPUs; extra cores add SUs



Multiple GPUs only if proven

Use `--gres=gpu:2` or more only when your code scales

Debug and Bigmem Jobs

Special-purpose partitions with special rules



Debug: test first

- Free, with no SU charge
- 1 job at a time: up to 4 CPUs and 16 GB
- Maximum walltime 8 hours
- Catches errors before a long job fails
- GPU tests go on `debug_gpu` with `gpu:a100_10g`

```
#SBATCH --partition=debug
#SBATCH --qos=debug
#SBATCH --time=7:59:00
# GPU: --partition=debug_gpu
#      --gres=gpu:a100_10g:1
```



Bigmem: large RAM

- 2 nodes with 2 TB RAM each
- 1 job per user; request more than 500 GB
- Default 16 GB per core
- SU charge uses a 1.5× factor
- Minimum memory required 500G

```
#SBATCH --partition=bigmem
#SBATCH --ntasks-per-node=64
#SBATCH --mem=512G
```

Interactive Jobs

A live shell on a compute node, through the interactive command

```
$ interactive -a <PI_UCID> -q standard -j cpu
Job Type: cpu
Starting an interactive session with the general
partition and 1 core for 01:00:00 of walltime
srun: job 584280 queued and waiting for resources
srun: job 584280 has been allocated resources

# GPU session
$ interactive -a <PI_UCID> -q standard -j gpu
# Free debug session
$ interactive -a <PI_UCID> -q debug -j cpu -p debug
```

WHEN TO USE IT

-  **Testing code or modules**
-  **Debugging runtime issues**
-  **Compiling or installing Python packages via `conda`**
-  **Exploring the environment**

A wrapper around `salloc`. Defaults: 1 core and 1 hour (plus 1 GPU for `-j gpu`).

interactive Command Options

Run interactive -h for the full help text

Flag	Required	What it sets	Default
-a ACCOUNT	Yes	PI account (check with <code>quota_info</code>)	—
-q QOS	Yes	Priority: standard, low, debug or <code>high_\$PI</code>	—
-j JOB_TYPE	Yes	cpu or gpu	—
-n NTASKS	No	Number of CPU tasks	1
-t WALLTIME	No	Walltime in hours	1
-g GPU	No	Number of GPUs (gpu jobs only)	1
-p PARTITION	No	SLURM partition	general (cpu), gpu (gpu)



GPU job with serial code? Skip `-n`: the default of 1 CPU per GPU avoids extra SU charges.

Core SLURM Commands

Submit, watch and control your jobs

sbatch

Submit a batch script

```
sbatch submit.sh
```

squeue

Your jobs and their state (PD, R)

```
squeue -u $LOGNAME
```

scancel

Cancel one job, or all of yours

```
scancel <job_id>  
scancel --me
```

scontrol

Detailed query on an active job

```
scontrol show job <job_id>
```

sinfo

Partition and node status

```
sinfo -N
```

srun

Launch tasks inside a job

```
srun ./myexe
```

Wulver Monitoring Tools

Extra helpers built for NJIT users



checkload

CPU load on every node; `sinfo` with more detail

Use it when: Checking whether a job overloads its node



checkq

`squeue` plus account, QoS, priority and time limit

Use it when: A job sits in PD and you want to know why



slurm_jobid <jobid>

Full job record, including total SUs consumed

Use it when: You want the exact SU cost of a job



squeue --start

Pending jobs in order of expected start time

Use it when: Planning ahead; start times are estimates



listqos

All QoS levels, or the members of one

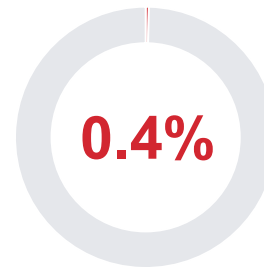
Use it when: Checking which QoS you can submit to

Reading Job Efficiency: Are You Over-Requesting?



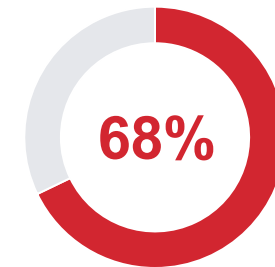
```
$ reportseff 575079
JobID   State      Elapsed   TimeEff  CPUEff  MemEff
575079  COMPLETED  00:00:28   0.4%   67.9%   0.1%

# TimeEff: elapsed time ÷ requested walltime
# CPUEff:  CPU time used ÷ (cores × elapsed)
# MemEff:  peak memory ÷ requested memory
```



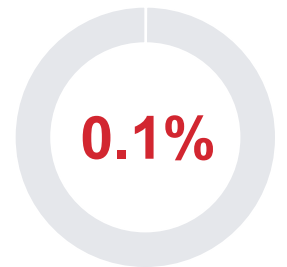
Time efficiency

28 s elapsed



CPU efficiency

19 s of 28 s



Memory efficiency

4.21 MB of 3.91 GB

Used 0.4% of the requested walltime and 0.1% of the memory: next time, request far less of both.

Common Mistakes and Fixes

More resources do not mean faster jobs

MISTAKE



`--ntasks=4` on serial Python

Runs the script 4 times, not 4× faster



128 tasks × 2 threads on one node

256 threads fight over 128 cores and slow down



Too little memory requested

Error: cgroup out-of-memory handler



Wrong partition or walltime

GPU on a non-GPU partition, or `--time` over 3 days

FIX



Request 1 task; use mpi4py or Parsl for real parallelism



`--ntasks-per-node=64 --cpus-per-task=2`
matches the hardware



Raise `--mem`, or add CPUs with `--mem-per-cpu`; then check the setup



Use `--partition=gpu` with `--gres`; keep `--time` at 72 h or less

Common Mistakes and Fixes (cont.)

Read the queue reason before you resubmit

MISTAKE



Job vanishes within 30 seconds

`sbatch` accepts it, but no output file appears



Invalid account or partition

The account/partition combination is rejected



ReqNodeNotAvail, Reserved for maintenance

PD: walltime runs into a maintenance window



MaxCpuPerAccount or MaxMemPerAccount

PD: your PI's high QoS is at its CPU or memory cap



AssocGrpBillingMinutes

PD: your PI group has used all its standard SUs

FIX



Run `slurm_jobid <jobid>; set --output`
and `--error` to see why



Check `--account`; confirm yours with `quota_info`
`$LOGNAME`



Shorten `--time` so the job ends before maintenance
starts



Check limits with `listqos high_$PI` and usage with
`squeue -q high_$PI`



Move to low QoS: `scontrol update`
`JobId=<job_id> QOS=low`

Resources to get your answers

Docs, how-tos, support and system updates



LEARN

Documentation

[HPC Documentation](#)

[Access to Wulver](#)

[Wulver Software](#)

[HPC Workshops](#)

`hpc.njit.edu`



HOW-TO

How-Tos & Software

[Conda environments](#)

[MIG guide](#)

[Request software](#)

[Open OnDemand](#)

`ondemand.njit.edu`



SUPPORT

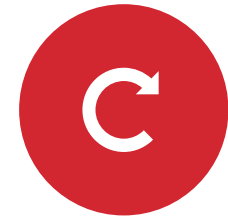
Get Help

[HPC Contact](#)

[Email a ticket](#)

[Open Office Hours](#)

`hpc@njit.edu`



UPDATES

System Updates

Message of the Day at login

[NJIT HPC News](#)

`hpc.njit.edu/news`

Best Practices and Next Steps

Request what you need, measure, then adjust



Right-size every request

Be precise with CPUs, memory, GPUs and `--time`



Test on debug first

Free, quick runs catch errors before long jobs



Measure, then tune

Run `reportseff slurm_jobid` after each new job and adjust



Test interactively

Use `interactive` to debug, compile and install packages



NEXT SESSION · OCTOBER 21, 2:30–3:30 PM · ONLINE

Introduction to Wulver: Job Efficiency and Resource Utilization

Docs: hpc.njit.edu · Questions: submit a support ticket · Prefer a GUI? Try Open OnDemand