Using the Slurm Scheduler on Composable Resources

HPRC Training
February 7, 2023
Slurm Job Scheduling

- HPC Architecture
- SBATCH Parameters
- Single node jobs
  - single-core
  - multi-core
  - Break
- Multi-node jobs
  - MPI jobs
  - TAMULauncher
  - array jobs
- Monitoring job resource usage
  - at runtime
  - after job completion
  - job debugging
FASTER is a 184-node Intel cluster from Dell with an InfiniBand HDR-100 interconnect. A100 GPUs, A10 GPUs, A30 GPUs, A40 GPUs and T4 GPUs are distributed and composable via Liqid PCIe fabrics. All nodes are based on the Intel Ice Lake processor.
Nodes, Cores and Queues

- **Node**
  - one computer unit of an HPC cluster each containing memory and one or more CPUs. There are generally two classifications of HPC nodes; login and compute.
    - **login node**
      - this is where users first login to stage their job scripts and do file and directory manipulations with text file editors (vi, gedit, emacs, portal) and Unix commands.
    - **compute node**
      - These are often referred to as just nodes since jobs are only scheduled on the compute nodes.
      - some compute nodes contain GPUs.
      - can use one or more compute nodes for a job based on how you configure your job script

- **Core**
  - There are 64 cores (cpus) on the FASTER 256GB memory compute nodes (250GB avail)
  - can use one or all 64 cores in a job script
    - check to see that the software you use in your job script supports multi-core

- **Queue**
  - Ordered list of all scheduled jobs for all users both in the PENDING and RUNNING states
  - Individual queues are also termed Partition (Example partition names: gpu, cpu)
  - cpu partition is auto-assigned but you must specify gpu partition to use GPUs
HPC Diagram

**Job Scheduler Queue**
- **compute node 1**
- **compute node 2**
- **compute node 3**
- **compute node 4**
- **compute node 5**
- **compute node 6**
- **hundreds of more compute nodes**

**Shared File Storage for /home and /scratch**

**Login Nodes**
- **login node 1**
- **login node 2**
- **login node 3**

**Login nodes are for:**
- file manipulation and job script preparation
- software installation and testing
- short tasks (< 60 minutes and max 8 cores)
  - also be aware of amount of memory utilized

**Compute Nodes**
- computational jobs which can use up to 64 cores and/or up to 250GB memory per FASTER compute node.
- all jobs running > 60 minutes
A batch job script is a text file that contains Unix and software commands and batch manager job parameters.
Accessing the HPRC Portal

- HPRC webpage: hprc.tamu.edu, Portal dropdown menu
Accessing FASTER via SSH

• SSH command is required for accessing FASTER:
  - On campus: ssh userNetID@faster.hprc.tamu.edu
  - Off campus:
    - Set up and start VPN (Virtual Private Network): u.tamu.edu/VPnetwork
    - Then: ssh userNetID@faster.hprc.tamu.edu
  - Two-Factor Authentication enabled for CAS, VPN, SSH

• SSH programs for Windows:
  - MobaXTerm (preferred, includes SSH and X11)
  - PuTTY SSH
  - Windows Subsystem for Linux

• https://portal-faster.hprc.tamu.edu
  - Select the "Clusters" tab and then ">_faster Shell Access"

• FASTER has 2 login nodes for TAMU users. Check the bash prompt.
  Login sessions that are idle for **60** minutes will be closed automatically.
  Processes run longer than **60** minutes on login nodes will be killed automatically.
  **Do not use more than 8 cores across all the login nodes!**
  **Do not use the sudo command.**

hprc.tamu.edu/wiki/HPRC:Access
Two-Factor Authentication

- Duo NetID two-factor authentication to enhance security ([it.tamu.edu/duo](it.tamu.edu/duo))
  - All web login (Howdy, portal.hprc.tamu.edu, Globus) through CAS
  - VPN to TAMU campus
  - SSH/SFTP to HPRC clusters

- See instructions in two-factor wiki page ([hprc.tamu.edu/wiki/Two_Factor](hprc.tamu.edu/wiki/Two_Factor))

- SSH clients work with Duo
  - ssh command from Linux, macOS Terminal, Windows cmd
  - MobaXterm for Windows (click on "Session" icon or via local session: hit "enter" 3 times and wait for "Password:" prompt)
  - Putty for Windows

- SFTP clients work with Duo
  - scp/sftp command from Linux, macOS Terminal, Windows cmd
  - WinSCP for Windows
  - Cyberduck for macOS

- Not all software supports SSH+Duo: SFTP in MATLAB

---

**Example: SSH login with Duo**

```bash
$ ssh userNetID@faster.hprc.tamu.edu
************************************************
.... warning message (snipped) ........
************************************************
Password:
Duo two-factor login for userNetID
Enter a passcode or select one of the following options:
1. Duo Push to XXX-XXX-1234
2. Phone call to XXX-XXX-1234
3. SMS passcodes to XXX-XXX-1234
Passcode or option (1-3): 1
Success. Logging you in...
```
Hands-On Activity: 2 Minutes

1. Please try to login to FASTER now.

2. What message do you see when you log on?

3. On which login node did you land?
How Busy are the HPRC Clusters?

Check on the command line of a cluster using the `sinfo` command

### `sinfo` command output

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>AVAIL</th>
<th>TIMELIMIT</th>
<th>JOB_SIZE</th>
<th>NODES (A/I/O/T)</th>
<th>CPUS (A/I/O/T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu*</td>
<td>up</td>
<td>7:00:00</td>
<td>1:32</td>
<td>35/50/27/112</td>
<td>2058/3382/1728/7168</td>
</tr>
<tr>
<td>gpu</td>
<td>up</td>
<td>7:00:00</td>
<td>1:32</td>
<td>14/27/11/52</td>
<td>252/2372/704/3328</td>
</tr>
<tr>
<td>memmerge</td>
<td>up</td>
<td>2:00:00</td>
<td>1:2</td>
<td>0/0/2/2</td>
<td>0/0/128/128</td>
</tr>
<tr>
<td>fpga</td>
<td>up</td>
<td>2:00:00</td>
<td>1:2</td>
<td>0/0/2/2</td>
<td>0/0/128/128</td>
</tr>
<tr>
<td>spr</td>
<td>up</td>
<td>2:00:00</td>
<td>1</td>
<td>0/1/0/1</td>
<td>0/112/0/112</td>
</tr>
<tr>
<td>atsp</td>
<td>up</td>
<td>2:00:00</td>
<td>1</td>
<td>0/8/2/10</td>
<td>0/512/128/640</td>
</tr>
<tr>
<td>staff</td>
<td>down</td>
<td>7:00:00</td>
<td>1-infinite</td>
<td>49/85/44/178</td>
<td>2310/6266/2816/11392</td>
</tr>
</tbody>
</table>

### A/I/O/T

- **A** = Active (in use by running jobs)
- **I** = Idle (available for jobs)
- **O** = Offline (unavailable for jobs)
- **T** = Total

**not all partitions are available to users**
Checking GPU Availability on FASTER

See GPU configuration and current availability

gpuavail

#### CONFIGURATION

<table>
<thead>
<tr>
<th>NODE</th>
<th>TYPE</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpu:a100:4</td>
<td>a100:4</td>
<td>3</td>
</tr>
<tr>
<td>gpu:a100:8</td>
<td>a100:8</td>
<td>1</td>
</tr>
<tr>
<td>gpu:a10:2</td>
<td>a10:2</td>
<td>1</td>
</tr>
<tr>
<td>gpu:a10:4</td>
<td>a10:4</td>
<td>1</td>
</tr>
<tr>
<td>gpu:a30:2</td>
<td>a30:2</td>
<td>2</td>
</tr>
<tr>
<td>gpu:a40:2</td>
<td>a40:2</td>
<td>2</td>
</tr>
<tr>
<td>gpu:a40:4</td>
<td>a40:4</td>
<td>1</td>
</tr>
<tr>
<td>gpu:t4:2</td>
<td>t4:2</td>
<td>1</td>
</tr>
<tr>
<td>gpu:t4:4</td>
<td>t4:4</td>
<td>29</td>
</tr>
<tr>
<td>gpu:t4:8</td>
<td>t4:8</td>
<td>10</td>
</tr>
</tbody>
</table>

#### AVAILABILITY

<table>
<thead>
<tr>
<th>NODE</th>
<th>GPU TYPE</th>
<th>GPU COUNT</th>
<th>GPU AVAIL*</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc023</td>
<td>t4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>fc070</td>
<td>t4</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>fc085</td>
<td>t4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>fc096</td>
<td>a40</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>fc097</td>
<td>a40</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>fc098</td>
<td>a40</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>fc099</td>
<td>a10</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>fc101</td>
<td>t4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>fc102</td>
<td>t4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>fc121</td>
<td>t4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>fc122</td>
<td>a10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>fc123</td>
<td>a30</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>fc124</td>
<td>a30</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

* Some available GPUs may not be immediately available due to other jobs using all or most of the compute node memory
SBATCH Parameters
#!/bin/bash
#SBATCH --job-name=spades         # keep job name short with no spaces
#SBATCH --time=1-00:00:00          # request 1 day; Format: days-hours:minutes:seconds
#SBATCH --nodes=1                 # request 1 node (optional since default=1)
#SBATCH --ntasks-per-node=1       # request 1 task (command) per node
#SBATCH --cpus-per-task=1         # request 1 cpu (core, thread) per task
#SBATCH --mem=3G                  # request 3GB total memory per node
#SBATCH --output=stdout.%x.%j     # save stdout to a file with job name and JobID appended to file name
#SBATCH --error=stderr.%x.%j      # save stderr to a file with job name and JobID appended to file name

# unload any modules to start with a clean environment
module purge
# load software modules
module load GCC/11.2.0   SPAdes/3.15.3
# run commands
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1

- Always include the first line exactly as it is; no trailing spaces or comments
- Slurm job parameters begin with `#SBATCH` and you can add comments afterwards as above
- Name the job script whatever you like but be consistent to make it easier to search for job scripts
  - `my_job_script.job`
  - `my_job_script.sbatch`
  - `run_program_project.sh`
  - `job_program_project.slurm`
Commonly Used Slurm SBATCH Parameters

- **--nodes**
  - number of nodes to use where a node is one computer unit of many in an HPC cluster
  - **--nodes=1**  
    # request 1 node (optional since default=1)
  - used for multi-node jobs
  - **--nodes=10**

- if number of cpus per node is not specified then defaults to 1 cpu
- default is 1 node if --nodes not used & can use with --ntasks-per-node and --cpus-per-task
  - do not use --nodes with --array

- **--ntasks**
  - a task can be considered a command such as blastn, bwa, script.py, etc.
  - **--ntasks=1**  
    # total tasks across all nodes where each task is scheduled a max of 1 cpu

- when using --ntasks > 1 without --nodes=1, the job may be scheduled on multiple compute nodes

- **--ntasks-per-node**
- use together with --cpus-per-task
  - **--ntasks-per-node=1**

- **--cpus-per-task**
  - number of CPUs (cores) for each task (command)
  - **--cpus-per-task=64**

---

one task
64 cpus

magicblast  -num_threads 64
Commonly Used Slurm SBATCH Parameters

- **--time**
  - max runtime for job *(required)*; format: days-hours:minutes:seconds *(days- is optional)*
  - **--time=24:00:00**  # set max runtime 24 hours *(same as --time=1-00:00:00)*
  - **--time=7-00:00:00**  # set max runtime 7 days

- **--mem**
  - total memory for each node *(required)*
  - **--mem=250G**  # request 250GB total memory *(max available for 256gb nodes)*

- **--job-name**
  - set the job name, keep it short and concise without spaces *(optional but highly recommended)*
  - **--job-name=myjob**

- **--output**
  - save all stdout to a specified file *(optional but highly recommended for debugging)*
  - **--output=stdout.%x.%j**  # saves stdout to a file named stdout.jobname.JobID

- **--error**
  - save all stderr to a specified file *(optional but highly recommended for debugging)*
  - **--error=stderr.%x.%j**  # saves stderr to a file named stderr.jobname.JobID
  - use just **--output** to save stdout and stderr to the same output file: **--output=output.%x.%j.log**

- **--partition**
  - specify a partition *(queue)* to use *(optional, use as needed)*
  - partition is automatically assigned to short, medium, long.
    - need to specify **--partition** parameter to use gpu, memverge, fpga
Commonly Used Optional Slurm Parameters

- **--gres**
  - used for requesting 1 or 2 GPUs; use GPU type in lowercase
  - `--gres=gpu:t4:1`  # request 1 T4 GPU; use replace :1 with :2 for two GPUs, etc
  - `--partition=gpu`  # also include this line when requesting GPUs

- **--account**
  - specify which HPRC account to use; see your accounts with
  - `--account=ACCOUNTNUMBER`
  - default account from myproject output is used if not specified

- **--mail-user**
  - send email to user
  - `--mail-user=myemail@tamu.edu`

- **--mail-type**
  - send email per job event: BEGIN, END, FAIL, ALL
  - `--mail-type=ALL`

- **--dependency**
  - schedule a job to start after a previous job successfully completes
  - `--dependency=afterok:JobID`
    - get the JobID of the previous job with `squeue -u $USER`
Submitting Slurm Jobs

- A job script is a text file of Unix commands with `#SBATCH` parameters
- `#SBATCH` parameters provide resource configuration request values
  - time, memory, nodes, cpus, output files, ...
- Jobs can be submitted using a job script or directly on the command line
- Submit the job using `sbatch` command with the job script name
  - your job script provides a record of commands used for an analysis
  - `sbatch my_job_script.job`
- Submit command on the command line by specifying all necessary parameters
  - must rely on your bash history to see `#SBATCH` parameters used which is not reliable
  - `sbatch -t 01:00:00 -n 1 -J myjob --mem 3G -o stdout.%j commands.sh`
- You can start an interactive job on the command line which ends when you exit the terminal
  - do not to use more than the requested memory and CPUs when your job starts.
  - `srun --time=04:00:00 --mem=3G --ntasks=1 --cpus-per-task=1 --pty bash`

slurm.schedmd.com/sbatch.html
FASTER Service Unit Calculations

- For the FASTER 256GB memory nodes (250GB available), you are charged Service Units (SUs) based on one of the following values whichever is greater.
  - 1 SU per CPU per hour or 1 SU per 3GB (3700M) of requested memory per hour

- Each T4 GPU would be an additional 64 SUs for one hour.
- Each A100, A10, A30, or A40 would be an addition 128 SUs for one hour.
- Unused SUs expire at the end of each fiscal year (Aug 31) and must be renewed.

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Total Memory per node (GB)</th>
<th>GPU count, type</th>
<th>Hours</th>
<th>SUs charged</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>0</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>64</td>
<td>250</td>
<td>0</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1 gpu:a30:2</td>
<td>1</td>
<td>129</td>
</tr>
<tr>
<td>64</td>
<td>250</td>
<td>1 gpu:a30:2</td>
<td>1</td>
<td>320</td>
</tr>
</tbody>
</table>

Charged for the second GPU which becomes unavailable for other jobs due to using all 64 cores.
Single-Node Jobs
Single vs Multi-Core Jobs

- When to use single-core jobs
  - The software being used only supports commands utilizing a single-core

- When to use multi-core jobs
  - If the software supports multiple cores (--threads, --cpus, ...) then configure the job script and software command options to utilize all CPUs on a compute node to get the job done faster unless the software specifically recommends a limited number of cores
    - FASTER 256GB memory compute nodes
      - 64 CPUs (cores) per compute node
      - 250GB of available memory per compute node
  - Can group multiple single-core commands into a "multi-core" job using TAMULauncher on one or multiple nodes
Single-Node Single-Core Job Scripts

```bash
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=01:00:00
#SBATCH --nodes=1  # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=3G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/11.2.0 SPAdes/3.15.3
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 1
```

**Example 1**

<table>
<thead>
<tr>
<th>1</th>
<th>Total CPUs requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>CPUs per node</td>
</tr>
<tr>
<td>3</td>
<td>total requested GB memory</td>
</tr>
<tr>
<td>1</td>
<td>SUs to start job</td>
</tr>
</tbody>
</table>

**Example 2**

<table>
<thead>
<tr>
<th>#!/bin/bash</th>
<th>#SBATCH --job-name=spades</th>
</tr>
</thead>
<tbody>
<tr>
<td>#SBATCH --time=01:00:00</td>
<td>#SBATCH --nodes=1  # optional since default=1</td>
</tr>
<tr>
<td>#SBATCH --ntasks-per-node=1</td>
<td>#SBATCH --cpus-per-task=1</td>
</tr>
<tr>
<td>#SBATCH --mem=4G</td>
<td>#SBATCH --output=stdout.%x.%j</td>
</tr>
<tr>
<td>#SBATCH --error=stderr.%x.%j</td>
<td>module purge</td>
</tr>
</tbody>
</table>

module load GCC/11.2.0 SPAdes/3.15.3
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 1

any unused SUs charged for a job are automatically reimbursed a few minutes after the job completes
Slurm Parameter: --ntasks

```bash
#!/bin/bash
#SBATCH --job-name=myjob    # job name
#SBATCH --time=1:00:00      # set the wall clock limit to 1 hour
#SBATCH --ntasks=1          # request 1 task (command) per node
#SBATCH --mem=3G            # request 3GB of memory per node
#SBATCH --output=stdout.%x.%j # create a file for stdout
#SBATCH --error=stderr.%x.%j # create a file for stderr
```

When only --ntasks is used, --cpus-per-task=1 and --nodes could be more than 1 if --ntasks is > 1

- **--ntasks=1**
  - NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1
- **--ntasks=48**  # without --nodes=1 could result in multiple nodes
  - NumNodes=1 NumCPUs=48 NumTasks=48 CPUs/Task=1
  - NumNodes=48 NumCPUs=1 NumTasks=48 CPUs/Task=1
Requesting all CPUs and Available Memory on FASTER Compute Nodes

CPU only node

64 cores, 256 GB nodes (180 nodes available)

```
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
```

GPU node

64 cores, 256 GB memory and 4 x A100 GPUs

```
#SBATCH --gres=gpu:a100:4
#SBATCH --partition=gpu
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
```

use lowercase gpu types: t4, a100, a10, a30, a40
Requesting one GPU on FASTER
Compute Nodes that have four GPUs

- Request \( \frac{1}{4} \) or less of the available CPU and memory resources for a compute node that has 4 installed GPUs when you need 1 GPU so that someone else can use the other GPUs, unless you need more CPUs and memory resources for your job.
- If you request 1 GPU with 64 cores and 250GB memory, the other 3 GPUs on the compute node are unavailable for other jobs and you are charged for all 4 GPUs.
- On FASTER each T4 GPU would be an additional 64 SUs for one hour and each A100, A10, A30, or A40 would be an addition 128 SUs for one hour.

```
#SBATCH --gres=gpu:a100:1
#SBATCH --partition=gpu
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --mem=56G
#SBATCH --time=01:00:00
```

### CONFIGURATION

<table>
<thead>
<tr>
<th>NODE</th>
<th>TYPE</th>
<th>COUNT</th>
</tr>
</thead>
</table>
gpu:a100:4 | 3 |     |

### AVAILABILITY

<table>
<thead>
<tr>
<th>GPU</th>
<th>GPU</th>
<th>GPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE</td>
<td>TYPE</td>
<td>COUNT</td>
<td>AVAIL*</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>fc002</td>
<td>a100</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>fc026</td>
<td>a100</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

16 + 64 = 80 SUs per hour for fc026
How Many SUs?

- How many SUs are charged if your job is scheduled on GPU node fc026?
- On FASTER you are charged per CPU/memory plus GPU count
  - Each T4 GPU is charged an additional 64 SUs for one hour
  - Each A100, A10, A30, or A40 GPU is charged an additional 128 SUs for one hour.

```
#SBATCH --gres=gpu:a100:1
#SBATCH --partition=gpu
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --time=01:00:00

? + ? = ?? SUs per hour for fc026
```
Select GPU type on FASTER Cluster

#!/bin/bash
#SBATCH --job-name=my_gpu_job
#SBATCH --time=1-00:00:00          # request 1 day of time for the job
#SBATCH --ntasks=32               # request half of the total cores
#SBATCH --gres=gpu:a30:1          # request 1 A30 GPU; replace :1 with :2 for two GPUs
#SBATCH --partition=gpu           # use partition=gpu when selecting GPUs
#SBATCH --mem=115G                # request 3.7GB memory per core
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

# unload modules to start with a clean environment; then load required modules
module purge
module load CUDA/11.7.0
#SBATCH --mem=115G

# run your gpu command
my_gpu_command

There are five types of GPUs on FASTER compute nodes. Select the type and quantity with --gres:

- **T4**:
  - `--gres=gpu:t4:N` (N can be 1 to 8)
  - 200 x T4 on FASTER

- **A100**:  
  - `--gres=gpu:a100:N` (N can be 1 to 12)
  - 40 x A100 on FASTER

- **A10**:  
  - `--gres=gpu:a10:N` (N can be 1 to 4)
  - 8 x A10 on FASTER

- **A30**:  
  - `--gres=gpu:a30:N` (N can be 1 to 2)
  - 4 x A30 on FASTER

- **A40**:  
  - `--gres=gpu:a40:N` (N can be 1 to 4)
  - 8 x A40 on FASTER
Single-Node Multi-Core Job Scripts

Example 1

| 32 | Total CPUs requested | 64 |
| 32 | CPUs per node | 64 |
| 250 | total requested GB memory | 250 |
| 1,536 | SUs to start job | 1,536 |

Example 2

It is best to request all cores and all memory if using the entire node.
You can use the environment variable `SLURM_CPUS_PER_TASK` to capture the value in the `#SBATCH --cpus-per-task` parameter so that you only need to adjust the cpus in one place.

```bash
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/11.2.0 SPAdes/3.15.3
spades.py -1 s1_R1 -2 s1_R2 -o outdir_s1 --threads $SLURM_CPUS_PER_TASK
spades.py -1 s2_R1 -2 s2_R2 -o outdir_s2 --threads $SLURM_CPUS_PER_TASK
```
Multi-Node Jobs
Slurm Parameters: --nodes --ntasks-per-node

```bash
#!/bin/bash
#SBATCH --job-name=myjob          # job name
#SBATCH --time=1:00:00            # set the wall clock limit to 1 hour
#SBATCH --nodes=2                 # request 2 nodes
#SBATCH --ntasks-per-node=1      # request 1 task (command) per node
#SBATCH --cpus-per-task=64       # request 64 cores per task
#SBATCH --mem=250G                # request 250GB of memory per node
#SBATCH --output=stdout.%x.%j     # create a file for stdout
#SBATCH --error=stderr.%x.%j      # create a file for stderr
```

It may be easier to scale jobs by using --nodes with --ntasks-per-node instead of with --ntasks. If you use --nodes with --ntasks, you need to calculate total CPUs for all nodes as the --ntasks value

- **--nodes=2 --ntasks-per-node=64**
  - NumNodes=2 NumCPUs=128 NumTasks=128 CPUs/Task=1 mem=250G per node
- **--nodes=2 --ntasks=128**
  - NumNodes=2 NumCPUs=128 NumTasks=128 CPUs/Task=1 mem=250G per node
- **--nodes=1 --ntasks=64**
  - NumNodes=1 NumCPUs=64 NumTasks=64 CPUs/Task=1 mem=250G per node
- **--nodes=2 --ntasks=64**
  - will allocate 32 cores on one node and 32 cores on a second node
- **when --nodes is > 1, make sure the software you are using supports multi-node processing**
PBS Multi-Node Multi-Core Job Script: Example 1

```bash
#!/bin/bash
#SBATCH --job-name=moose
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load MOOSE/e70fe56913
mpirun -np 640 -npernode 64 /path/to/moose-opt -i moose.i
```

<table>
<thead>
<tr>
<th></th>
<th>Total CPUs requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>640</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>CPUs per node</td>
</tr>
<tr>
<td>250</td>
<td>Total requested GB memory per node</td>
</tr>
<tr>
<td>15,360</td>
<td>SUs to start job (nodes * cpus * hours)</td>
</tr>
</tbody>
</table>
MPI Multi-Node Multi-Core Job Script: Example 2

```bash
#!/bin/bash
#SBATCH --job-name=abyss-pe
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/8.3.0 OpenMPI/3.1.4 ABySS/2.1.5
abyss-pe np=640 j=64 lib='lib1' lib1='sample1_R1 sample1_R2'
```

<table>
<thead>
<tr>
<th>640</th>
<th>Total CPUs requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>CPUs per node</td>
</tr>
<tr>
<td>250</td>
<td>Total requested GB memory per node</td>
</tr>
<tr>
<td>15,360</td>
<td>SUs to start job (nodes * cpus * hours)</td>
</tr>
</tbody>
</table>
TAMULauncher

- [hprc.tamu.edu/wiki/SW:tamulauncher](hprc.tamu.edu/wiki/SW:tamulauncher)
- Use when you have hundreds or thousands of commands to run each utilizing a single-core or a few cores
  - `tamulauncher` keeps track of which commands completed successfully
    - to see the list of completed commands
      - `tamulauncher --status commands_file.txt`
    - if time runs out, `tamulauncher` can be restarted and it will know which was the last successfully completed command
  - submit `tamulauncher` as a batch job within your job script
  - can run `tamulauncher` interactively on login node; limited to 8 cores
  - you can check the `--status` on the command line from the working directory
- run a single command of your thousands to make sure the command is correct and to get an estimate of resource usage (CPUs, memory, time)
- request all cores and memory on the compute node(s) and configure your commands to use all available cores
TAMULauncher Multi-Node Single-Core Commands

commands.txt (500 lines for example)

run_spades_tamulauncher.sh

run 64 spades.py commands per node with each command using 1 core.
Requesting all 64 cores on FASTER reserves entire node for your job

run 64 single-core commands per node; useful when each command requires <= 3.7GB memory
create a commands file (named whatever you want) to go with the the job script
load the software module in the job script not the commands file

# !/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=1
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/11.2.0 SPAdes/3.15.3
tamulauncher commands.txt

spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 1
spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 1
spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 1
spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 1
spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 1
spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 1
spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 1
spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 1
spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 1
spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 1
spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 1
spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 1
spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 1
spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 1
spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 1
spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 1
spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 1
spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 1
spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 1
spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 1
spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 1
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1
```bash
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
#SBATCH --cpus-per-task=4
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
module purge
module load GCC/11.2.0 SPAdes/3.15.3
tamulauncher commands.txt
run 16 spades.py commands per node with each command using 4 cores.
Requesting all 64 cores on FASTER reserves entire node for your job
```

**TAMULauncher Multi-Node Multi-Core Commands**

### commands.txt

500 lines for example

<table>
<thead>
<tr>
<th>commands.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 4</td>
</tr>
<tr>
<td>spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 4</td>
</tr>
</tbody>
</table>

- useful when each command requires more than 3.7GB but less than all available memory
- use OMP_NUM_THREADS if needed when running fewer commands than requested cores
  - add on the line before the tamulauncher command
  - `export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK`

run_spades_tamulauncher.sh

```
node=2
export OMP_NUM_THREADS=4
module purge
module load GCC/11.2.0 SPAdes/3.15.3
tamulauncher commands.txt
```

**Useful when each command requires more than 3.7GB but less than all available memory**

**Use OMP_NUM_THREADS if needed when running fewer commands than requested cores**

- add on the line before the tamulauncher command
- `export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK`
Making a TAMULauncher Commands File
Part 1

Input files are two files per sample and named:

- s1_R1.fastq.gz
- s1_R2.fastq.gz

Run this command to create the example files:

```bash
mkdir seqs && touch seqs/s{1..40}_R{1,2}.fastq.gz
```

Run the following commands to get familiar with useful shell commands for creating and manipulating variables:

```bash
file=seqs/s1_R1.fastq.gz
echo $file
basename $file
sample=$(basename $file)
echo $sample
echo ${sample/_R1.fastq.gz}
echo ${sample/R1/R2}
```

# create variable named file
# show contents of variable
# strip off path of variable
# create variable named sample
# show contents of variable
# strip off _R1.fastq.gz
# substitute R1 text with R2
Input files are two files per sample and named: s1_R1.fastq.gz  s1_R2.fastq.gz

- Run the following commands to loop through all R1 files in the reads directory and create the commands.txt
- Use just the R1 files because we only need to capture the sample names once.

```bash
for file in seqs/*_R1.*gz
do
    read1=$file
    read2=${read1/_R1./_R2.}
    sample=$(basename ${read1/_R1.fastq.gz})
    echo spades.py -1 $read1 -2 $read2 -o ${sample}_out --threads 1
done > commands.txt
```

Match as much as possible to avoid matching sample names
**Other Useful Unix Commands**

- \$\{variable#*SubStr\} # will drop beginning of variable value up to first occurrence of 'SubStr'
- \$\{variable##*SubStr\} # will drop beginning of variable value up to last occurrence of 'SubStr'
- \$\{variable%SubStr*\} # will drop part of variable value from last occurrence of 'SubStr' to the end
- \$\{variable%%SubStr*\} # will drop part of variable value from first occurrence of 'SubStr' to the end

These are useful if the part of the filename for each sample that needs to be removed is not the same.

\s1\_S1\_R1.fastq.gz  \s2\_S2\_R1.fastq.gz  \s3\_S3\_R1.fastq.gz

Make a new directory and create a new set of files for this exercise.

```
mkdir seqs2
for i in {1..10}; do touch seqs2/s\${i}\_S\${i}\_R{1,2}.fastq.gz; done
```

Unix Command                      Output
file=seqs2/s1_S1_R1.fastq.gz      seqs2/s1_S1_R1.fastq.gz
echo $file                        seqs2/s1_S1_R1.fastq.gz
echo ${file%_S*}                  seqs2/s1
basename ${file%_S*}              s1
sample=$(basename ${file%_S*})    s1
echo $sample
Slurm Job Array Parameters and Runtime Environment Variables

- Array jobs are good to use when you have multiple samples each of which can utilize an entire compute node running software that supports multiple threads but does not support MPI
  - `--array=0-23` # job array indexes 0-23
  - `--array=1-24` # job array indexes 1-24
  - `--array=1,3,5,7` # job array indexes 1,3,5,7
  - `--array=1-7:2` # job array indexes 1 to 7 with a step size of 2
  - do not use `--nodes` with `--array`

- Use the index value to select which commands to run either from a text file of commands or as part of the input file name or parameter value
  - `$_SLURM_ARRAY_TASK_ID` is the array index value

- Stdout and stderr files can be saved per index value
  - `--output=stdout.%x.%j.%A_%a`
  - `--error=stderr.%x.%j.%A_%a`

- Maximum array size is 1000 and max total pending and running jobs per user is 1000

- Limit the number of simultaneously running tasks
  - can help prevent reaching file and disk quotas due to many intermediate and temporary files
  - `--array=1-40%5` # job array with indexes 1-40; max of 5 running array jobs
  - as one job completes another array index is run on an available node
#!/bin/bash
#SBATCH --job-name=bwa_array
#SBATCH --time=1-00:00:00
#SBATCH --array=1-40%5           # job array of indexes 1-40; max of 5 running array indexes
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j.%A_%a
#SBATCH --error=stderr.%x.%j.%A_%a

module purge
module load GCCcore/11.2.0  BWA/0.7.17

# get a line from commands.txt file into a variable named command
command=$(sed -n ${SLURM_ARRAY_TASK_ID}p commands.txt)

$command

- The sed command will print a specified line number from commands.txt based on the $SLURM_ARRAY_TASK_ID
- The number of lines in your commands.txt file should be the same as the number of array indexes
- Can use %5 to limit the array to a maximum of 5 nodes used simultaneously but you need enough SUs to cover entire number of array indexes in order to submit the job. May be useful to prevent creating too many temporary files.
- There are other ways to use $SLURM_ARRAY_TASK_ID but this example is useful because it has a file of all commands used in each array index
- Doesn't work when commands have redirection operators: | < >
Slurm Job Array Example 2

```bash
#!/bin/bash
#SBATCH --job-name=bwa_array
#SBATCH --time=1-00:00:00
#SBATCH --array=1-40
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j.%A_%a
#SBATCH --error=stderr.%x.%j.%A_%a

module purge
module load GCCcore/11.2.0  BWA/0.7.17

bwa mem -M -t 64 -R '@RG\tID:\tLB:pe\tSM:DR34\tPL:ILLUMINA' genome.fasta \
  sample_${SLURM_ARRAY_TASK_ID}_R1.fastq.gz  sample_${SLURM_ARRAY_TASK_ID}_R2.fastq.gz \\
  | samtools view -h -Sb - | samtools sort -o sample_${SLURM_ARRAY_TASK_ID}.out.bam \
  -m 7G -@ 1 -T $TMPDIR/tmp4sort${SLURM_ARRAY_TASK_ID} -
```

- Can use `$SLURM_ARRAY_TASK_ID` if your commands only differ by a number in the file names
- The `$SLURM_ARRAY_TASK_ID` variable will be assigned the array index from 1 to 40 in this example
- Can use this approach when commands have redirection operators: | < >

- Lines ending with \ indicate the command continues on the next line
- do not put a space after \
Useful Slurm Runtime Environment Variables

- **$TMPDIR**
  - This is a temporary local disk space (3.84TB) created at runtime and is deleted when the job completes.
  - The directory is mounted on the compute node and files created in $TMPDIR do not count against your file and disk quotas.
  - `samtools sort -T $TMPDIR/sort`

- **$SLURM_CPUS_PER_TASK**
  - Returns how many CPU cores were allocated on this node.
  - Can be used in your command to match requested `#SBATCH cpus`.
  - `#SBATCH --cpus-per-task=64`
  - `samtools sort --threads $SLURM_CPUS_PER_TASK`

- **$SLURM_ARRAY_TASK_ID**
  - Can be used to select or run one of many commands when using a job array.

- **$SLURM_JOB_NAME**
  - Populated by the `--job-name` parameter.
  - `#SBATCH --job-name=bwa_array`

- **$SLURM_JOB_NODELIST**
  - Can be used to get the list of nodes assigned at runtime.

- **$SLURM_JOBID**
  - Can be used to capture JobID at runtime.
Useful Unix Environment Variables

- Type `env` to see all Unix environment variables for your login session
- `$USER`
  - This will be automatically populated with your NetID
    - `echo $USER`
- `$SCRATCH`
  - You can use this to change to your `/scratch/user/netid` directory
    - `cd $SCRATCH`
- `$OMP_NUM_THREADS`
  - Used when software uses OpenMP for multithreading; default is 1
    - `export OMP_NUM_THREADS=64`
- `$PWD`
  - Contains the full path of the current working directory
Finding NGS job template scripts using GCATemplates on HPRC clusters

```
mkdir $SCRATCH/slurm_class

cd $SCRATCH/slurm_class
gcatemplates
```

For practice, we will copy a template file

- Enter 1 then continue through the menus to find the template that contains bbmap
  - or use the search to find bbmap
- Final step will save a template job script file to your current working directory
Monitoring Job Resource Usage
Submit a Slurm Job

- Submit your job script
  - `sbatch run_bbmap_38.96_bbnorm_faster.sh`
- See status and JobID of all your submitted jobs
  - `squeue -u $USER`
- Can cancel (kill) a PENDING or RUNNING job using JOBID
  - `scancel JOBID`
Monitor CPU usage for a FASTER Running Job

- See CPU and memory usage of all your running jobs
  - `pestat -u $USER`
  - Stats for `pestat` are updated every 3 minutes

<table>
<thead>
<tr>
<th>Hostname</th>
<th>Partition</th>
<th>Node</th>
<th>Num_CPU</th>
<th>CPUload</th>
<th>Memsize</th>
<th>Freemem</th>
<th>Joblist</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc162</td>
<td>cpu*</td>
<td>mix</td>
<td>64</td>
<td>63.56*</td>
<td>256000</td>
<td>253786</td>
<td>731601</td>
<td>netid</td>
</tr>
<tr>
<td>fc163</td>
<td>cpu*</td>
<td>mix</td>
<td>64</td>
<td>61.78*</td>
<td>256000</td>
<td>253404</td>
<td>731601</td>
<td>netid</td>
</tr>
<tr>
<td>fc164</td>
<td>gpu*</td>
<td>mix</td>
<td>56</td>
<td>25.65*</td>
<td>256000</td>
<td>253492</td>
<td>731601</td>
<td>netid</td>
</tr>
<tr>
<td>fc165</td>
<td>gpu*</td>
<td>mix</td>
<td>28</td>
<td>27.10*</td>
<td>256000</td>
<td>239764</td>
<td>731601</td>
<td>netid</td>
</tr>
<tr>
<td>fc166</td>
<td>gpu*</td>
<td>mix</td>
<td>28</td>
<td>0.01*</td>
<td>256000</td>
<td>238543</td>
<td>731601</td>
<td>netid</td>
</tr>
</tbody>
</table>

Low CPU load utilization highlighted in **Red**
Good CPU load utilization highlighted in **Purple**
Ideal CPU load utilization displayed in **White**
(Freemem should also be noted)
Monitor a Running Job

- See lots of info about your running or recently completed (~10 minutes) job

```bash
scontrol show job JobID
```

- Can add this command at the beginning of your job script to capture job info into the stdout file

```bash
scontrol show job $SLURM_JOB_ID
```

Example:

```bash
#!/bin/bash
#SBATCH --job-name=bbnorm
#SBATCH --time=01:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
```

Job details:

- JobId=119112
- JobName=bbnorm
- UserId=mynetid(99999)
- GroupId=mynetid(99999)
- Priority=22505
- Nice=0
- Account=hprc
- QOS=normal
- JobState=RUNNING
- Reason=None
- Dependency=(null)
- Requeue=1
- Restarts=0
- BatchFlag=1
- Reboot=0
- ExitCode=0:0
- RunTime=00:00:09
- TimeLimit=01:00:00
- TimeMin=N/A
- SubmitTime=2023-02-06T10:03:44
- EligibleTime=2023-02-06T10:03:44
- AccrueTime=2023-02-06T10:03:44
- StartTime=2023-02-06T10:03:45
- EndTime=2023-02-06T11:03:45
- Deadline=N/A
- SuspendTime=None
- SecsPreSuspend=0
- LastSchedEval=2023-02-06T10:03:45
- Scheduler=Main
- Partition=cpu
- AllocNode:Sid=login2:513947
- ReqNodeList=(null)
- ExcNodeList=(null)
- NodeList=fc083
- BatchHost=fc083
- NumNodes=1
- NumCPUs=64
- NumTasks=1
- CPUs/Task=64
- TRES=cpu=64,mem=250G,node=1,billing=64
- Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=* MinCPUsNode=64
- MinMemoryNode=250G
- MinTmpDiskNode=0
- Features=(null)
- DelayBoot=00:00:00
- OverSubscribe=OK
- Contiguous=0
- Licenses=(null)
- Network=(null)

Command=/scratch/user/mynetid/slurm_class/run_bbmap_38.96_bbnorm_faster.sh
WorkDir=/scratch/user/mynetid/slurm_class
StdErr=/scratch/user/mynetid/slurm_class/stderr.bbnorm.119112
StdIn=/dev/null
StdOut=/scratch/user/mynetid/slurm_class/stdout.bbnorm.119112
Power=
See Completed Job Efficiency Stats

seff JobID

will show CPU and Memory efficiency based on selected resources

seff 119112

Job ID: 119112
Cluster: faster
User/Group: mynetid/mynetid
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 64
CPU Utilized: 03:20:22
CPU Efficiency: 57.80% of 05:46:40
core-walltime
Job Wall-clock time: 00:05:25
Memory Utilized: 118.81 GB
Memory Efficiency: 47.52% of 250.00 GB

CPU load was at 100% for 57% of the run time or 37 cores were at 100% for the job time or some other combination

max memory utilized was 47% of requested memory
Monitor GPU and CPU usage for a Job

You can use the jobstats command to monitor resource usage and create graphs.

```
#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --time=2-00:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64
#SBATCH --mem=250G
#SBATCH --gres=gpu:a100:2
#SBATCH --partition=gpu
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load CUDA/11.7.0

# run jobstats in the background (&) to monitor resource usage
jobstats &

my_gpu_command

# run jobstats to create graphs of resource usage for this job
jobstats
```

Notice that GPU: 1 was not used in this job.

Next time configure job script to use just 1 GPU

```bash
--gres=gpu:a100:1
```

[https://hprc.tamu.edu/wiki/SW:jobstats](https://hprc.tamu.edu/wiki/SW:jobstats)
See All Your Jobs for Current Fiscal Year

- **myproject -j all**
  - ProjectAccount
  - JobID
  - JobArrayIndex
  - SubmitTime
  - StartTime
  - EndTime
  - Walltime
  - TotalSlots
  - UsedSUs
  - Total Jobs
  - Total Usage (SUs)

It will take a few days for today’s jobs to show up in the results.
Debugging Job Submission

- The job was not scheduled
  - make sure the job CPU count and memory specification exist
    - sbatch: error: CPU count per node cannot be satisfied
      sbatch: error: Batch job submission failed: Requested node configuration is not available
  - check your HPRC default account to see if you have enough SUs
    - myproject
    - ask your PI to allocate SUs to your account
    - Set or change your default account
- myproject -d Account

```bash
[mynetid@faster2 bbnorm]$ myproject
run 'myproject -h' to see the other options.
```

---

<table>
<thead>
<tr>
<th>Account</th>
<th>FY</th>
<th>Default</th>
<th>Allocation</th>
<th>Used &amp; Pending SUs</th>
<th>Balance</th>
<th>PI</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789101</td>
<td>2023</td>
<td>Y</td>
<td>5000.00</td>
<td>4999.00</td>
<td>1.00</td>
<td>X, Reveille</td>
</tr>
<tr>
<td>123456789102</td>
<td>2023</td>
<td>N</td>
<td>10000000.00</td>
<td>0.00</td>
<td>10000000.00</td>
<td>X, Reveille</td>
</tr>
</tbody>
</table>

---

High Performance Research Computing | hprc.tamu.edu | NSF Award #1925764

53
If your job is in the PENDING state for a long time:
- Check to see if the cluster is busy using `sinfo` or see hprc.tamu.edu.
- Use `gpuavail` to see if the GPU partition is busy.
- Check to see if your job walltime overlaps with a scheduled maintenance.

The scheduled 11 hour FASTER maintenance will start in:

- 3 days 16 hours 41 minutes

Scheduled jobs will not start if they overlap with this maintenance window.

A 7-day job submitted at the time of the above message will remain queued and will not start until after the maintenance is complete.
Debugging Slurm Jobs

- Look for an out of memory error message; could occur in only one index of a job array

```bash
slurmstepd: error: Exceeded job memory limit at some point.
```

  - Increase the amount of SBATCH memory in your job script and resubmit the job

- If you see an 'Out of disk space' or 'No space left on device' error
  - check your file and disk quotas using the `showquota` command
    ```bash
    showquota
    ```
  - reduce the number of files you have generated
    - delete any nonessential or temporary files
    - use `$TMPDIR` in your command if software supports a temporary directory
    - create and download a .tar.gz package of completed projects and delete the original directory to free up disk space
  - request a temporary increase in file and/or disk quota for your project
The HPRC portal allows users to do the following:

- Browse files on the FASTER filesystem
- Access the FASTER Unix command line
  - no SUs charged for using command line
  - SUs charged for jobs launched from command line
  - runs on login node; limit your processes to 8 cores
- Compose job scripts
- Launch interactive GUI apps (SUs charged)
- Monitor and stop running jobs and interactive sessions
Need Help?

First check the FAQ hprc.tamu.edu/wiki/HPRC:CommonProblems
- FASTER User Guide hprc.tamu.edu/wiki/FASTER
- Email your questions to help@hprc.tamu.edu

Help us, help you -- we need more info
- Which Cluster
- Username
- JobID(s) if any
- Location of your jobfile, input/output files
- Application used if any
- Module(s) loaded if any
- Error messages
- Steps you have taken, so we can reproduce the problem

Let us know when the issue has been resolved so we can close the helpdesk ticket.