

Job Submission Guide

Running jobs on HPC resources requires submitting your jobs to our batch scheduler. The scheduler will then find the resources you requested and will execute your job on those resource when they become available. The following guide will teach you how to write a job submission script and submit it to one of the HPC RedRaider partitions: Nocona and Matador.

Table of Contents

1. Job Submission Script
 - a. Submission Script Layout
 - b. Equivalent Commands for Submission
 - c. Other Command Options for Submission
 - d. Submission Script Commands Explained
2. Nocona Job Submission Tutorial
3. Matador Job Submission Tutorial
4. Monitoring Jobs
 - a. Monitoring a queue or running job – squeue
 - i. Viewing detailed information regarding a currently running job
 - b. Monitoring a failed or completed job – sacct
5. Interactive Jobs

Job Submission Script

Every job that runs requires a submission script that contains parameters for the job scheduler as well as the commands you wish to run on the cluster. The parameters you pass to the job scheduler will depend on the partition you are submitting to. Below you will find a generic submission script as an example to discuss what the various parameters do, followed by example submission scripts for our Nocona and Matador clusters.

Submission Script Layout

1. `#!/bin/bash`
2. `#SBATCH -J <job_name>`
3. `#SBATCH -o %x.o%j`
4. `#SBATCH -e %x.e%j`
5. `#SBATCH -p <partition_name>`
6. `#SBATCH -N <number_of_nodes>`
7. `#SBATCH --ntasks-per-node=<tasks_per_node>`
8. `#SBATCH --mem-per-cpu=<integer>MB`
9. `#SBATCH -t <HH:MM:SS>`
10. `#SBATCH --gpus-per-node=<count> ##(for matador GPU only)`

Command(s) you wish to run on the cluster.

Equivalent Commands for Submission

- `-o %x.o%j` is the same as `--output=%x.o%j`
- `-e %x.e%j` is the same as `--error=%x.e%j`
- `-p <partition_name>` is the same as `--partition=<partition_name>`
- `-N <number_of_nodes>` is the same as `--nodes=<number_of_nodes>`
- `-t <HH:MM:SS>` is the same as `--time=<HH:MM:SS>`

Other Command Options for Submission

- For line (6) and (7), apart from `-N <number_of_nodes>` and `--ntasks-per-node=<tasks_per_node>`, there are a variety of options you can choose to submit your job.

```
--ntasks=<total_tasks> or -n <total_tasks>
--ntasks-per-core=<tasks_per_core>
--cpus-per-task=<cpus_per_task>
--threads-per-core=<threads_per_core>
```

- For line (8), alternative command for your job submission includes:

```
--mem=<integer>G
```

Which would specify the total amount of memory per node.

- For line (10), another command for your job submission in Matador includes:

```
--gpus=<count>
```

Where `<count>` is the total number of gpus. Since there are two GPUs per node, make sure `<count>` does not exceed GPUs limit.

- To set a working directory for your job, you can include this option:

```
-D <directory_path>
or
--chdir <directory_path>
```

Where `<directory_path>` is the path to your working directory.

- For **array** job, you can add in this extra line of command in your submission script to specify your request:

```
--array=<start_num-><end_num->:<increment>
```

or

```
-a <start_num><end_num>:<increment>
```

<start_num> is the beginning task ID of your array.

<end_number> is the ending taskID of your array.

<increment> is the increment between each task.

- To specify **job dependencies**, you can add in an extra line of either one of these commands:

```
--dependency=after:job_id[:job_id-...] (1)
```

```
--dependency=afterany:job_id[:~job_id...] (2)
```

```
--dependency=--afternotok:job_id[:
```

```
-...] (3)
```

```
--dependency=afterok:job_id
```

```
-job_id...] (4)
```

```
--dependency=aftercorr:job_id (5)
```

```
--dependency=singleton (6)
```

Note:

(1) Job begins after the specified jobs have begun execution.

(2) Job begins after the specified jobs have terminated.

(3) Job begins after the specified jobs have terminated in some failed state.

(4) Job begins after the specified jobs have successfully executed.

(5) A task of a job array can begin after the corresponding task ID in the specified job has completed successfully.

(6) Job begins after any previously launched jobs sharing the same job name and user have terminated

- To schedule a **begin time** of when you would like to start your job, you can add in this command line option in your job script:

```
--begin=<YYYY-MM-DD[THH:MM:[SS]]>
```

- To receive notifications about your job, you can add in these options:

```
--mail-user=<username@ttu.edu> (1)
```

```
--mail-type=ALL (2)
```

(1) Instructs the scheduler to send notification emails regarding the job status to `<username@ttu.edu>`.

(2) Specifies notification emails to be sent at the event of when the job begins, ends, fails, or is requeued. If you would like to be notified on a specific event, other alternatives for **ALL** include **BEGIN**, **END**, **FAIL**, and **REQUEUE**.

Instead of **ALL**, you can replace it with other command options to specify at what events related to your job you would like to receive email notifications:

BEGIN: when the job starts to execute

END: when the job completes

FAIL: if and when the job fails

REQUEUE: if and when the job is requeued.

ALL: for all of the above cases.

Submission Script Commands Explained

For the **Submission Script Layout**, the lines starting with **#SBATCH** are the script directives for the job scheduler in SLURM. Each option is described below:

- **-J <job_name>** sets the name for the job. This can be referred to later in the script using the variable `$SLURM_JOB_NAME`.
- **-o %x.o%j** indicates the name of the standard output file in the format of `<job_name>.o<job_ID>`
 - For filename pattern within **#SBATCH** directives, use `%x` and `%j` to denote your job name and job ID.
 - For commands outside of **#SBATCH** directives, `$SLURM_JOB_NAME` and `$SLURM_JOB_ID` can be used to denote your job name and job ID.
- **-e %x.e%j** indicates the name of the standard error file in the format of `<job_name>.e<job_ID>`.
- **-p <partition_name>** instructs the scheduler to submit the job to the partition `<partition_name>`.
- **-N <number_of_nodes>** and **--ntasks-per-node=<tasks_per_node>** instruct the scheduler to allocate the job based on the number of nodes and number of cores specified. Some important information:

- The `<tasks_per_node>` should be the number of tasks per node you request, where a task in SLURM refers to one core. The number of tasks/cores can be modified as long as it does not exceed the limit amount.
- An alternative option to `--ntasks-per-node=<tasks_per_node>` would be `--ntasks=<total_tasks>` or `-n <total_tasks>`, which indicates the number of cores to request in total. For instance, to request 128 cores per node in 2 nodes, which yields a total of 256, one can request `-N 2` and `-n 256`.
- For **serial** jobs (**SM** job), `<number_of_nodes>` should be 1 and `<tasks_per_node>` should be less than the fixed number of cores per node. For instance, `-N 1 --ntasks-per-node=50` is a valid job allocation for an SM job in Nocona (since 50 cores are less than 128 cores).
- **--mem-per-cpu=<integer>MB** instructs the scheduler to reserve `<integer>` megabytes of virtual memory per core. Some important information:
 - Make sure you are not requesting more than 3.9GB per core for Nocona and 9.4GB per core for Matador. *Additional information: there are 512GB per node in Nocona and 384GB per node in Matador but the maximum memory allowed for job submission is 503GB per node in Nocona and 376GB per node in Matador!*
 - When requesting memory involving floating point numbers in Nocona (e.g: 3.3GB, 3.4GB), please request in terms of megabytes (MB) and not gigabytes (GB). Another thing to keep in mind is that RedRaider uses a system of GiB (gibibytes), so 1GiB = 1024MiB (not 1GB = 1000MB) in Nocona and Matador, so 3.9GB is actually around 3994MB, which is why 3994MB should be requested instead for 3.9GB.
 - Your job will not be allowed to use more than the amount of memory you request - so you will need to ensure you know the memory requirements of your programs before you run them.
- **-t <HH:MM:SS>** sets the the maximum amount of time your job will take to execute. This allows the scheduler to better fit your job into the workflow. The default and maximum values for this parameter will depend on the account you select.
- **--gpus-per-node=<count>** allocates the generic resources for a number `<count>` of GPUs per node, used for Matador partition only. There are 2 GPUs in each node, so make sure the count does not exceed 2 per node.

The last line will be any commands you would like to run in your script. You should modify this command to any commands that you need to run for your project.

Nocona Job Submission Tutorial

Submitting a job on Nocona can be done using the following steps:

Step 1. Log on to *login.hpcc.ttu.edu* using your eRaider account and password.

Step 2. Copy the tutorial job script folder for nocona to your \$HOME area

```
#Copy the folder and all of its contents to your $HOME
cp -r /lustre/work/examples/nocona/mpi ~/mpi_tutorial

#Change the current directory to the mpi_tutorial we just
copied.
cd ~/mpi_tutorial
```

Inside the ~/mpi_tutorial folder there should be a file named mpi.sh. This is the submission script file for this particular tutorial. To read this file we can use the command *cat mpi.sh* which will print out the contents of this script.

```
#!/bin/bash
#SBATCH --job-name=MPI_Test_Job
#SBATCH --output=%x.o%j
#SBATCH --error=%x.e%j
#SBATCH --partition nocona
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --time=10:00:00
##SBATCH --mem-per-cpu=3994MB  ##3.9GB, Modify based on needs

module load gcc/10.1.0 openmpi/4.0.4
mpirun ./mpi_hello_world
```

This script sets the name of the project (**--job-name** or **-J**) to "MPI_Test_Job", requests the job be run on the "nocona" partition (**--partition** or **-p**), requests the job be assigned a full 128 cores per node, and requests that the job be allowed to run for a maximum of 10 hours (**--time** or **-t**). When a full node with 128 cores is used, there is no need to request memory because the maximum memory for that node will be allocated by default. However, if you would like to request less memory per core, feel free to uncomment and modify the memory part (**--mem-per-cpu**) based on your needs.

Next the script runs the command "*module load gcc/10.1.0 openmpi/4.0.4*". This will execute before the mpi job is run, allowing the correct compiler and mpi implementation to be loaded prior to executing the mpirun command.

Step 3. We will now submit a job to the Nocona partition using the *sbatch* command. The sbatch command will request the job be scheduled and return the job ID for the job so you can monitor its progress.

```
sbatch mpi.sh
```

Step 4. You can use the “*squeue -u <username>*” command to view information about any jobs you have submitted that have not yet completed. For more information about this or other commands used to monitor your job, please see the [Monitoring Jobs](#) section below. Below is an image showing the output of running the commands listed in **Step 3** and **Step 4**.

```
login-20-26:/Slurm_job/mpi_tutorial$ sbatch mpi.sh
Submitted batch job 1435
login-20-26:/Slurm_job/mpi_tutorial$ squeue -u nocona55
      JOBID PARTITION    NAME     USER ST       TIME  NODES NODELIST(REASON)
      1435    nocona MPI_Test nocona55 R        0:05      1 cpu-23-1
login-20-26:/Slurm_job/mpi_tutorial$
```

Step 5. Once execution has completed, you can read the results by viewing the output file for our mpi job. You can expect the output file to be found in the `~/mpi_tutorial` directory under the name `MPI_Test_Job.o<job_ID>`. In our example the file would be found at `~/mpi_tutorial/MPI_Test_Job.o3406`. While your results will likely vary, the output should look something like this:

```
Hello world from processor cpu-23-1, rank 70 out of 128
processors
Hello world from processor cpu-23-1, rank 102 out of 128
processors
Hello world from processor cpu-23-1, rank 118 out of 128
processors
Hello world from processor cpu-23-1, rank 119 out of 128
processors
Hello world from processor cpu-23-1, rank 126 out of 128
processors
Hello world from processor cpu-23-1, rank 4 out of 128
processors
Hello world from processor cpu-23-1, rank 5 out of 128
processors
Hello world from processor cpu-23-1, rank 6 out of 128
processors
Hello world from processor cpu-23-1, rank 15 out of 128
processors
Hello world from processor cpu-23-1, rank 31 out of 128
processors
Hello world from processor cpu-23-1, rank 53 out of 128
processors
Hello world from processor cpu-23-1, rank 54 out of 128
processors
```

Hello world from processor cpu-23-1, rank 55 out of 128 processors
Hello world from processor cpu-23-1, rank 58 out of 128 processors
Hello world from processor cpu-23-1, rank 76 out of 128 processors
Hello world from processor cpu-23-1, rank 81 out of 128 processors
Hello world from processor cpu-23-1, rank 82 out of 128 processors
Hello world from processor cpu-23-1, rank 85 out of 128 processors
Hello world from processor cpu-23-1, rank 86 out of 128 processors
Hello world from processor cpu-23-1, rank 90 out of 128 processors
Hello world from processor cpu-23-1, rank 97 out of 128 processors
Hello world from processor cpu-23-1, rank 100 out of 128 processors
Hello world from processor cpu-23-1, rank 103 out of 128 processors
Hello world from processor cpu-23-1, rank 106 out of 128 processors
Hello world from processor cpu-23-1, rank 109 out of 128 processors
Hello world from processor cpu-23-1, rank 117 out of 128 processors
Hello world from processor cpu-23-1, rank 120 out of 128 processors
Hello world from processor cpu-23-1, rank 124 out of 128 processors
Hello world from processor cpu-23-1, rank 0 out of 128 processors
Hello world from processor cpu-23-1, rank 1 out of 128 processors
Hello world from processor cpu-23-1, rank 2 out of 128 processors
Hello world from processor cpu-23-1, rank 3 out of 128 processors
Hello world from processor cpu-23-1, rank 7 out of 128 processors
Hello world from processor cpu-23-1, rank 8 out of 128 processors
Hello world from processor cpu-23-1, rank 9 out of 128 processors

Hello world from processor cpu-23-1, rank 10 out of 128 processors
Hello world from processor cpu-23-1, rank 11 out of 128 processors
Hello world from processor cpu-23-1, rank 12 out of 128 processors
Hello world from processor cpu-23-1, rank 13 out of 128 processors
Hello world from processor cpu-23-1, rank 14 out of 128 processors
Hello world from processor cpu-23-1, rank 16 out of 128 processors
Hello world from processor cpu-23-1, rank 17 out of 128 processors
Hello world from processor cpu-23-1, rank 18 out of 128 processors
Hello world from processor cpu-23-1, rank 19 out of 128 processors
Hello world from processor cpu-23-1, rank 20 out of 128 processors
Hello world from processor cpu-23-1, rank 21 out of 128 processors
Hello world from processor cpu-23-1, rank 22 out of 128 processors
Hello world from processor cpu-23-1, rank 23 out of 128 processors
Hello world from processor cpu-23-1, rank 24 out of 128 processors
Hello world from processor cpu-23-1, rank 25 out of 128 processors
Hello world from processor cpu-23-1, rank 26 out of 128 processors
Hello world from processor cpu-23-1, rank 27 out of 128 processors
Hello world from processor cpu-23-1, rank 28 out of 128 processors
Hello world from processor cpu-23-1, rank 29 out of 128 processors
Hello world from processor cpu-23-1, rank 30 out of 128 processors
Hello world from processor cpu-23-1, rank 32 out of 128 processors
Hello world from processor cpu-23-1, rank 33 out of 128 processors
Hello world from processor cpu-23-1, rank 34 out of 128 processors

Hello world from processor cpu-23-1, rank 35 out of 128 processors
Hello world from processor cpu-23-1, rank 36 out of 128 processors
Hello world from processor cpu-23-1, rank 37 out of 128 processors
Hello world from processor cpu-23-1, rank 38 out of 128 processors
Hello world from processor cpu-23-1, rank 39 out of 128 processors
Hello world from processor cpu-23-1, rank 40 out of 128 processors
Hello world from processor cpu-23-1, rank 41 out of 128 processors
Hello world from processor cpu-23-1, rank 42 out of 128 processors
Hello world from processor cpu-23-1, rank 43 out of 128 processors
Hello world from processor cpu-23-1, rank 44 out of 128 processors
Hello world from processor cpu-23-1, rank 45 out of 128 processors
Hello world from processor cpu-23-1, rank 46 out of 128 processors
Hello world from processor cpu-23-1, rank 47 out of 128 processors
Hello world from processor cpu-23-1, rank 48 out of 128 processors
Hello world from processor cpu-23-1, rank 49 out of 128 processors
Hello world from processor cpu-23-1, rank 50 out of 128 processors
Hello world from processor cpu-23-1, rank 51 out of 128 processors
Hello world from processor cpu-23-1, rank 52 out of 128 processors
Hello world from processor cpu-23-1, rank 56 out of 128 processors
Hello world from processor cpu-23-1, rank 57 out of 128 processors
Hello world from processor cpu-23-1, rank 59 out of 128 processors
Hello world from processor cpu-23-1, rank 60 out of 128 processors
Hello world from processor cpu-23-1, rank 61 out of 128 processors

Hello world from processor cpu-23-1, rank 62 out of 128 processors
Hello world from processor cpu-23-1, rank 63 out of 128 processors
Hello world from processor cpu-23-1, rank 64 out of 128 processors
Hello world from processor cpu-23-1, rank 65 out of 128 processors
Hello world from processor cpu-23-1, rank 66 out of 128 processors
Hello world from processor cpu-23-1, rank 67 out of 128 processors
Hello world from processor cpu-23-1, rank 68 out of 128 processors
Hello world from processor cpu-23-1, rank 69 out of 128 processors
Hello world from processor cpu-23-1, rank 71 out of 128 processors
Hello world from processor cpu-23-1, rank 72 out of 128 processors
Hello world from processor cpu-23-1, rank 73 out of 128 processors
Hello world from processor cpu-23-1, rank 74 out of 128 processors
Hello world from processor cpu-23-1, rank 75 out of 128 processors
Hello world from processor cpu-23-1, rank 77 out of 128 processors
Hello world from processor cpu-23-1, rank 78 out of 128 processors
Hello world from processor cpu-23-1, rank 79 out of 128 processors
Hello world from processor cpu-23-1, rank 80 out of 128 processors
Hello world from processor cpu-23-1, rank 83 out of 128 processors
Hello world from processor cpu-23-1, rank 84 out of 128 processors
Hello world from processor cpu-23-1, rank 87 out of 128 processors
Hello world from processor cpu-23-1, rank 88 out of 128 processors
Hello world from processor cpu-23-1, rank 89 out of 128 processors
Hello world from processor cpu-23-1, rank 91 out of 128 processors

Hello world from processor cpu-23-1, rank 92 out of 128 processors
Hello world from processor cpu-23-1, rank 93 out of 128 processors
Hello world from processor cpu-23-1, rank 94 out of 128 processors
Hello world from processor cpu-23-1, rank 95 out of 128 processors
Hello world from processor cpu-23-1, rank 96 out of 128 processors
Hello world from processor cpu-23-1, rank 98 out of 128 processors
Hello world from processor cpu-23-1, rank 99 out of 128 processors
Hello world from processor cpu-23-1, rank 101 out of 128 processors
Hello world from processor cpu-23-1, rank 104 out of 128 processors
Hello world from processor cpu-23-1, rank 105 out of 128 processors
Hello world from processor cpu-23-1, rank 107 out of 128 processors
Hello world from processor cpu-23-1, rank 108 out of 128 processors
Hello world from processor cpu-23-1, rank 110 out of 128 processors
Hello world from processor cpu-23-1, rank 111 out of 128 processors
Hello world from processor cpu-23-1, rank 112 out of 128 processors
Hello world from processor cpu-23-1, rank 113 out of 128 processors
Hello world from processor cpu-23-1, rank 114 out of 128 processors
Hello world from processor cpu-23-1, rank 115 out of 128 processors
Hello world from processor cpu-23-1, rank 116 out of 128 processors
Hello world from processor cpu-23-1, rank 121 out of 128 processors
Hello world from processor cpu-23-1, rank 122 out of 128 processors
Hello world from processor cpu-23-1, rank 123 out of 128 processors
Hello world from processor cpu-23-1, rank 125 out of 128 processors

```
Hello world from processor cpu-23-1, rank 127 out of 128
processors
```

Step 6. Edit the `mpi.sh` file and change the line "`#SBATCH --nodes=1`" to instead say "`#SBATCH --nodes=2`" and then rerun steps 3, 4 and 5 and see what changed. Notice how now the job had to run on multiple compute nodes, causing you to get messages back from multiple machines on the cluster.

Step 7. Congratulations, you have now successfully set up and run an MPI job on Nocona!

Matador Job Submission Tutorial

Submitting a job on Matador can be done using the following steps:

Step 1. Log on to *login.hpc.ttu.edu* using your eRaider account and password.

Step 2. Copy the tutorial job script folder for matador to your \$HOME area

```
#Copy the folder and all of its contents to your $HOME
cp -r /lustre/work/examples/matador/mpi ~/mpi_tutorial
#Change the current directory to the mpi_tutorial we just
copied.
cd ~/mpi_tutorial
```

Inside the `~/mpi_tutorial` folder there should be a file named `mpi.sh`. This is the submission script file for this particular tutorial. To read this file we can use the command `cat mpi.sh` which will print out the contents of this script.

```
#!/bin/bash
#SBATCH --job-name=MPI_Test_Job
#SBATCH --output=%x.o%j
#SBATCH --error=%x.e%j
#SBATCH --partition matador
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=10:00:00
#SBATCH --gpus-per-node=2
##SBATCH --mem-per-cpu=9625MB  ##9.4GB, modify based on needs

module load gcc/9.3.0 openmpi/3.1.6-cuda
mpirun ./mpi_hello_world
```

This script sets the name of the project (**--job-name** or **-J**) to "MPI_Test_Job", requests the job be run on the "matador" partition (**--partition** or **-p**), requests the job be assigned a full 40 cores per node, and requests that the job be allowed to run for a maximum of 10 hours (**--time** or **-t**). When a full node with 40 cores is used, there is no need to request memory because the maximum memory for that node will be allocated by default. However, if you would like to request less memory per core, feel free to uncomment and modify the memory part (**--mem-per-cpu**) based on your needs.

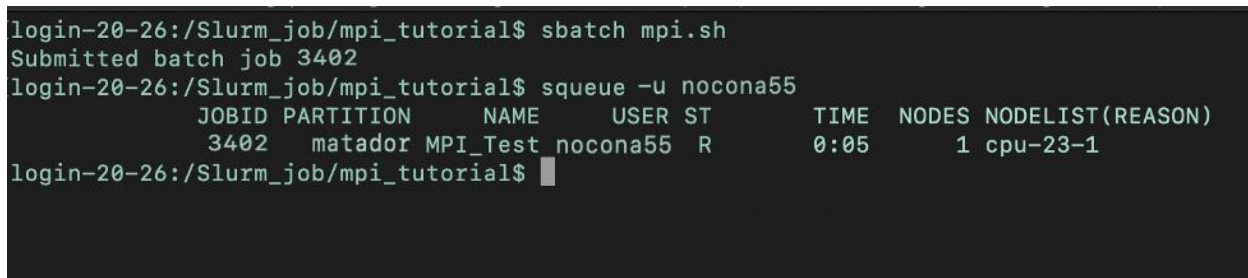
Next the script runs the command "*module load gcc/9.3.0 openmpi/3.1.6-cuda*". This will execute before the mpi job is run, allowing the correct compiler and mpi implementation to be loaded prior to executing the mpirun command.

Next the script runs the "mpirun" command which will actually execute the mpi_hello_world program.

Step 3. We will now submit a job to the Matador partition using the sbatch command. The sbatch command will request the job be scheduled and return the job ID for the job so you can monitor its progress.

```
sbatch mpi.sh
```

Step 4. You can use the "*queue -u <username>*" command to view information about any jobs you have submitted that have not yet completed. For more information about this or other commands used to monitor your job, please see the [Monitoring Jobs](#) section below. Below is an image showing the output of running the commands listed in **Step 3** and **Step 4**.



```
login-20-26:/Slurm_job/mpi_tutorial$ sbatch mpi.sh
Submitted batch job 3402
login-20-26:/Slurm_job/mpi_tutorial$ squeue -u nocona55
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
       3402   matador MPI_Test nocona55  R        0:05      1 cpu-23-1
login-20-26:/Slurm_job/mpi_tutorial$
```

Step 5. Once execution has completed, you can read the results by viewing the output file for our mpi job. You can expect the output file to be found in the ~/mpi_tutorial directory under the name MPI_Test_Job.o<job_ID>. In our example the file would be found at ~/mpi_tutorial/MPI_Test_Job.o3402. While your results will likely vary, the output should look something like this:

```
Hello world from processor gpu-20-1, rank 29 out of 40
processors
Hello world from processor gpu-20-1, rank 26 out of 40
processors
```

Hello world from processor gpu-20-1, rank 39 out of 40 processors
Hello world from processor gpu-20-1, rank 22 out of 40 processors
Hello world from processor gpu-20-1, rank 21 out of 40 processors
Hello world from processor gpu-20-1, rank 6 out of 40 processors
Hello world from processor gpu-20-1, rank 35 out of 40 processors
Hello world from processor gpu-20-1, rank 18 out of 40 processors
Hello world from processor gpu-20-1, rank 25 out of 40 processors
Hello world from processor gpu-20-1, rank 16 out of 40 processors
Hello world from processor gpu-20-1, rank 7 out of 40 processors
Hello world from processor gpu-20-1, rank 0 out of 40 processors
Hello world from processor gpu-20-1, rank 33 out of 40 processors
Hello world from processor gpu-20-1, rank 30 out of 40 processors
Hello world from processor gpu-20-1, rank 5 out of 40 processors
Hello world from processor gpu-20-1, rank 38 out of 40 processors
Hello world from processor gpu-20-1, rank 11 out of 40 processors
Hello world from processor gpu-20-1, rank 4 out of 40 processors
Hello world from processor gpu-20-1, rank 13 out of 40 processors
Hello world from processor gpu-20-1, rank 28 out of 40 processors
Hello world from processor gpu-20-1, rank 15 out of 40 processors
Hello world from processor gpu-20-1, rank 12 out of 40 processors
Hello world from processor gpu-20-1, rank 19 out of 40 processors
Hello world from processor gpu-20-1, rank 36 out of 40 processors
Hello world from processor gpu-20-1, rank 37 out of 40 processors

```
Hello world from processor gpu-20-1, rank 10 out of 40
processors
Hello world from processor gpu-20-1, rank 23 out of 40
processors
Hello world from processor gpu-20-1, rank 14 out of 40
processors
Hello world from processor gpu-20-1, rank 17 out of 40
processors
Hello world from processor gpu-20-1, rank 24 out of 40
processors
Hello world from processor gpu-20-1, rank 27 out of 40
processors
Hello world from processor gpu-20-1, rank 8 out of 40
processors
Hello world from processor gpu-20-1, rank 1 out of 40
processors
Hello world from processor gpu-20-1, rank 20 out of 40
processors
Hello world from processor gpu-20-1, rank 9 out of 40
processors
Hello world from processor gpu-20-1, rank 2 out of 40
processors
Hello world from processor gpu-20-1, rank 31 out of 40
processors
Hello world from processor gpu-20-1, rank 3 out of 40
processors
Hello world from processor gpu-20-1, rank 32 out of 40
processors
Hello world from processor gpu-20-1, rank 34 out of 40
processors
```

Step 6. Edit the `mpi.sh` file and change the line `"#SBATCH --nodes=1"` to instead say `"#SBATCH --nodes=2"` and then rerun steps 3, 4 and 5 and see what changed. Notice how now the job had to run on multiple compute nodes, causing you to now get messages back from multiple machines on the cluster.

Step 7. Congratulations, you have now successfully set up and run an MPI job on Matador!

Monitoring Jobs

Once you have submitted a job you may find that you wish to monitor how it is progressing. There are three basic commands:

- The `sacct` command is generally used to view information about all the jobs that are running or have completed
 - To view information of a specific job, use `sacct -j <job_ID>`.
- The `squeue` command is generally used to show the status of jobs that are waiting on the queue or running
 - To view information of a specific job, use `squeue -j <job_ID>`.
- The `sinfo` command shows the status of partitions.

NOTE: `sacct` is recommended to be used for completed jobs only, while `squeue` is recommended to view jobs that are waiting or running.

Monitoring a queued or running jobs - squeue

The `squeue` command is used to view information about any jobs you have waiting on the queue or are currently running in the SLURM schedule. This command can be run as “`squeue -u <username>`” in order to only show your job status and not other users.

```
[login-20-26:/Slurm_job$ sbatch testnocona2.sh
Submitted batch job 1395
[login-20-26:/Slurm_job$ squeue -u nocona55
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REAS
SON)
1395 nocona Job_Test nocona55 R 0:03 1 cpu-23-1
[login-20-26:/Slurm_job$ █
```

Do not run `squeue` within a `watch` command! While it may not seem like a resource intensive command to run, running the `squeue` command often leads to slower HPC resources for all of our users.

If you had no running jobs, then `squeue` will appear to do nothing. Otherwise you should see a table that contains 10 columns, detailed below:

1. JOBID

The job ID for the job. This is the number you will need to know in order to get detailed information about that job.

2. PARTITION

The partition where you submitted your job to.

3. NAME

The name of the job as set by the -J or --job-name option when the job was started.

4. USER

The username of the person who submitted the job.

5. ST

The current state of the job, commonly denoted as one of the following:

- CA: Job is cancelled
- CG: Job is completing
- R: Job is currently running
- RD: Job is being held
- RQ: Job is reqeud
- PD: Job is pending and awaiting resource allocation
- F: Job failed
- NF: Job failed due to allocated node failure

6. NODES

The number of nodes allocated for the job.

7. NODELIST

The compute node name allocated for your job.

Viewing detailed information regarding a currently running job

To view a detailed summary of a running job on the SLURM schedule, you can use the command *squeue* as follows:

```
squeue -j <job_ID>
```

Do not run squeue within a watch command! While it may not seem like a resource intensive command to run, running the *squeue* command often leads to slower HPCC resources for all of our users.

Monitoring a failed or completed job - sacct

Once a job has completed, you will no longer be able to view any information regarding the job using the *squeue* command. If you wish to view a summary of information regarding a job that has failed or completed, then you will instead need to use the *sacct* command.

Using the command "sacct" you can see useful information regarding all the jobs that have been submitted.

Using the command "sacct -j <job_ID>" you can see useful information regarding a particular job.

If you wish to view some sample *sacct* data then feel free to view the *sacct* information for the jobs we test ran above, or substitute in any of your previously completed job IDs.

Nocona example: `sacct -j 3406`

Matador example: `sacct -j 3402`

Your job status will be presented in a table containing 7 columns:

```
login-20-26:/Slurm_job/mpi_tutorial$ sacct -j 3402
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
3402	MPI_Test_+	matador	default	20	COMPLETED	0:0
3402.batch	batch		default	20	COMPLETED	0:0
3402.extern	extern		default	20	COMPLETED	0:0

1. JobID

The job ID for your job.

2. JobName

The name of the job set by -J or --job-name in your job submission.

3. Partition

The partition you submitted the job to.

4. Account

The type of account/project that your job was in.

5. State

The current state of the job, commonly denoted as one of the following:

- CANCELLED
- COMPLETED
- FAILED
- NODE_FAIL
- OUT_OF_MEMORY

- PENDING
- RUNNING
- SUSPENDED
- TIMEOUT

6. ExitCode

The exit code returned for the job.

Additionally, you can use “*sacct --helpformat*” to see a list of fields for your job status that can be specified with the “*--format*” option.

For example:

Indicated below is the output of all available fields after running the command “*sacct --helpformat*”

```
[root@login-20-26 ~]# sacct --helpformat
Account          AdminComment     AllocCPUS        AllocNodes
AllocTRES        AssocID          AveCPU           AveCPUFreq
AveDiskRead      AveDiskWrite     AvePages         AveRSS
AveVMSize        BlockID         Cluster          Comment
Constraints      ConsumedEnergy   ConsumedEnergyRaw CPUTime
CPUTimeRAW       DBIndex          DerivedExitCode  Elapsed
ElapsedRaw       Eligible         End              ExitCode
Flags            GID              Group            JobID
JobIDRaw         JobName          Layout           MaxDiskRead
MaxDiskReadNode MaxDiskReadTask  MaxDiskWrite     MaxDiskWriteNode
MaxDiskWriteTask MaxPages         MaxPagesNode     MaxPagesTask
MaxRSS           MaxRSSNode       MaxRSSTask       MaxVMSize
MaxVMSizeNode    MaxVMSizeTask    McsLabel         MinCPU
MinCPUNode       MinCPUTask       NCPUS            NNodes
NodeList         NTasks          Priority          Partition
QOS              QOSRAW          Reason           ReqCPUFreq
ReqCPUFreqMin    ReqCPUFreqMax   ReqCPUFreqGov   ReqCPUS
ReqMem           ReqNodes         ReqTRES          Reservation
ReservationId    Reserved         ResvCPU          ResvCPURAW
Start            State            Submit           Suspended
SystemCPU        SystemComment    Timelimit        TimelimitRaw
TotalCPU         TRESUsageInAve  TRESUsageInMax   TRESUsageInMaxNode
TRESUsageInMaxTask TRESUsageInMin TRESUsageInMinNode TRESUsageInMinTask
TRESUsageInTot   TRESUsageOutAve TRESUsageOutMax  TRESUsageOutMaxNode
TRESUsageOutMaxTask TRESUsageOutMin TRESUsageOutMinNode TRESUsageOutMinTask
TRESUsageOutTot  UID              User             UserCPU
WCKey            WCKeyID         WorkDir
```

Indicated below is the output of the fields’ status specified in “*--format*” option.

```
[root@login-20-26 ~]# sacct --format=jobid,elapsed,ncpus,ntasks,state
JobID      Elapsed      NCPUS      NTasks      State
-----
9463       00:00:00     16384      1           PENDING
9464       00:00:00     16384      1           PENDING
10076      00:00:00     1200       1           PENDING
11298      00:00:10     256        1           FAILED
11298.batch 00:00:10     128        1           FAILED
11298.extern 00:00:10     256        2           COMPLETED
11298.0     00:00:04     256        256        FAILED
11299      00:00:02     2          1           COMPLETED
11299.extern 00:00:02     2          2           COMPLETED
11299.0     00:00:01     2          2           COMPLETED
11300      00:00:18     256        1           COMPLETED
11300.batch 00:00:18     128        1           COMPLETED
11300.extern 00:00:18     256        2           COMPLETED
11300.0     00:00:16     256        256        COMPLETED
```

Interactive Jobs

The “*interactive*” command in SLURM allows you to directly access a CPU or GPU node(s). In other words, an interactive job allocates resources on compute nodes and allows you to directly interact with the nodes.

Command:

```
interactive [-A] [-c] [-p] [-J] [-w] [-g] [-h]
```

Optional arguments:

- A: the account name
- c: number of CPU cores to request (default: 1)
- p: partition to run job in (default: nocona)
- J: job name (default: INTERACTIVE)
- w: node name
- g: number of GPU to request
- h: show this usage info