
NREL HPC
Slurm: Advanced Topics
Presented by
Andrew Archer

Quick review: Slide Conventions

- Verbatim command-line interaction:
  - "$" precedes explicit typed input from the user.
  - "↲" represents hitting "enter" or "return" after input to execute it.
  - "…" denotes text output from execution was omitted for brevity.
  - "#" precedes comments, which only provide extra information.

```
$ ssh hpc_user@eagle.nrel.gov↲ (external)
    …
Password+OTPToken: # Your input will be invisible
```

- Command-line executables in prose:
  - "The command `sinfo` is very useful."

Review of systems access: Eagle DNS Configuration

Internal		External (Requires OTP Token)	
Login	DAV	Login	DAV
eagle.hpc.nrel.gov	**eagle-dav.hpc.nrel.gov**	eagle.nrel.gov	eagle-dav.nrel.gov

Direct Hostnames (Internal)	
Login	DAV
el1.hpc.nrel.gov	ed1.hpc.nrel.gov
el2.hpc.nrel.gov	ed2.hpc.nrel.gov
el3.hpc.nrel.gov	ed3.hpc.nrel.gov

Topics we will address

1. Job monitoring and forensics
2. Advanced Slurm functions (Flags)
3. Eagle best practices
4. Parallelizing with Slurm
5. GPU nodes
6. Questions
7. Follow up

Sections

1. Job monitoring and forensics
2. Advanced Slurm functions (Flags)
3. Eagle best practices
4. Parallelizing with Slurm
5. GPU nodes
6. Questions
7. Follow up

Slurm Job Monitoring and Forensics

- **Squeue -** view information about jobs located in the Slurm scheduling queue.

One of the most useful commands to get quick information about the status of your job or jobs running on Eagle.

```
$ sbatch -A <handle> rollcall.slurm
Submitted batch job 630410
$ squeue -u username --start
JOBID  PARTITION      NAME        USER   ST            START_TIME  NODES SCHEDNODES      NODELIST(REASON)
630410     short  node_rol    username   PD   2019-03-15T12:57:49     10 (null)           (BeginTime)
$ squeue -j 630410 --start
JOBID  PARTITION      NAME        USER   ST            START_TIME  NODES SCHEDNODES      NODELIST(REASON)
630410     short  node_rol    username   PD   2019-03-15T12:57:49     10 (null)           (BeginTime)
```

https://www.nrel.gov/hpc/eagle-monitor-control-commands.html

Basic squeue monitoring

- To check the status of a job:
- squeue -u <username>
- squeue -j <jobid>
- ssh to compute node (derived from squeue –j output)
- Run “top” to review processes

```
$ squeue -j 632719
JOBID     PARTITION  NAME      USER      ST       TIME      NODES   NODELIST(REASON)
632719    standard   JobName   username  R     1-02:22:12     2         r7i7n[19-20]
```

Slurm Job Monitoring and Forensics

- **sacct -** Can report resource usage for running or terminated jobs including individual tasks, which can be useful to detect load imbalance between the tasks

```
$sacct
       JobID    JobName  Partition    Account    AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
637283       helloworld     short      csc000         36     FAILED      2:0
637283.batch      batch                 csc000         36     FAILED      2:0
637283.exte+     extern                 csc000         36  COMPLETED      0:0
637288       helloworld     short      csc000         36  COMPLETED      0:0
637288.batch      batch                 csc000         36  COMPLETED      0:0
637288.exte+     extern                 csc000         36  COMPLETED      0:0
```

- **sstat -** Display various status information of a running job/step.

```
$sstat  --format=AveCPU,AvePages,AveRSS,AveVMSize,JobID -j 632717
   AveCPU     AvePages       AveRSS      AveVMSize            JobID
----------         ----------         ----------      ---------------         ------------
  13:39:07              0       262932     1134371043         632717.2$
```

Using sinfo

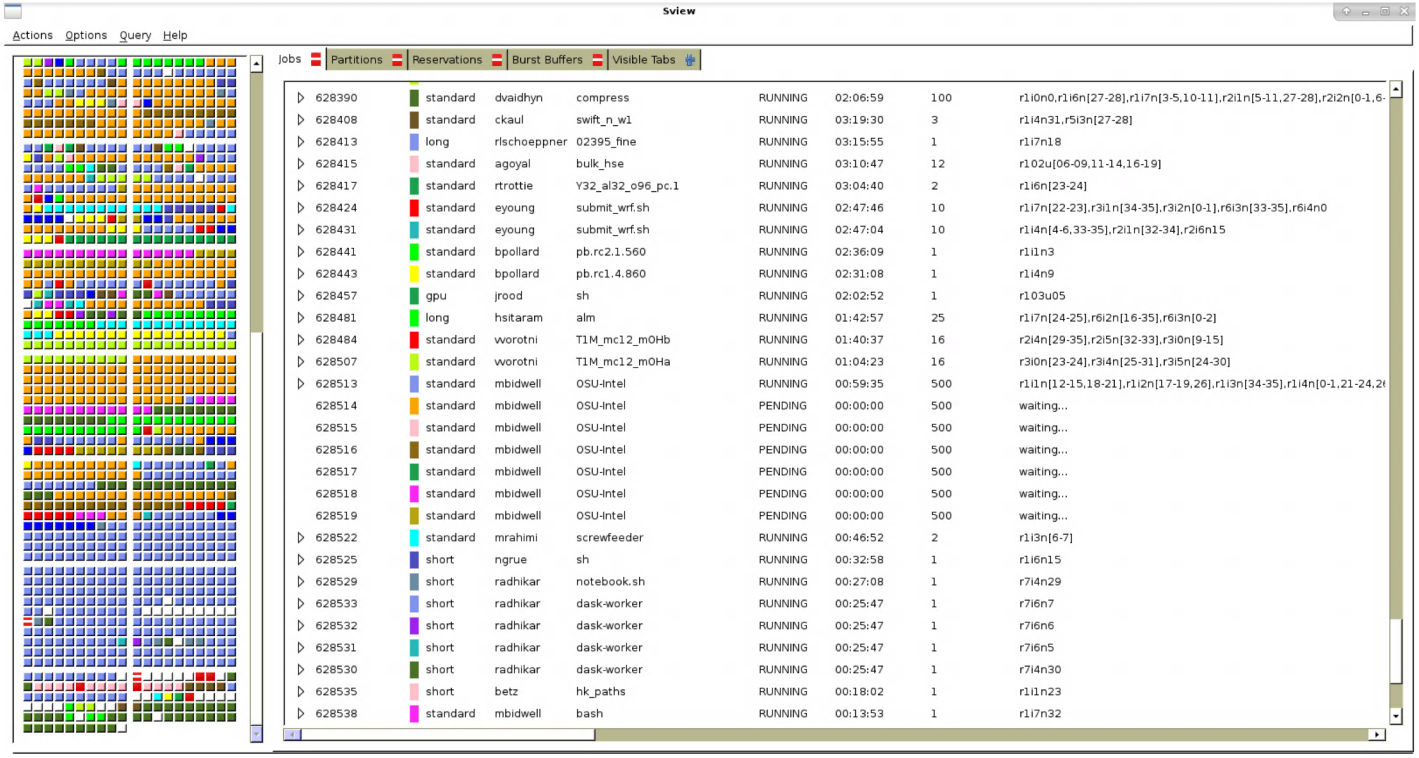
sinfo - view information about Slurm nodes and partitions.

```
$sinfo | grep gpu
gpu           up 2-00:00:00      1 drain* r103u07
gpu           up 2-00:00:00     24  alloc
r103u[01,03],r104u[17,19,21,23,25,27],r105u[01,03,05,07,09,11,13,15,17,19,21,23,25,27,29,31]
gpu           up 2-00:00:00     17   idle r103u[05,09,11,13,15,17,19],r104u[01,03,05,07,09,11,13,15,29,31]

$ sinfo –s <summarize: list only a partition state summary with no node state details>
PARTITION  AVAIL  TIMELIMIT   NODES(A/I/O/T)  NODELIST
short         up    4:00:00  2020/54/13/2087  r1i0n[0-35],r1i1n[0-35],r1i2n[0-35],r1i3n[0-35],r1i4n[0-35],r1i5n[0-
35],r1i6n[0-35],r1i7n[0-35],r2i0n[0-35],r2i1n[0-35],r2i2n[0-35],r2i3n[0-35],r2i4n[0-35],r2i5n[0-35],r2i6n[0-35],r2i7n[0-
35],r3i0n[0-35],r3i1n[0-35,r3i2n[0-35],r3i3n[0-35],r3i4n[0-35],r3i5n[0-35],r3i6n[0-35],r3i7n[0-35],r4i0n[0-35],r4i1n[0-
35],r4i2n[0-35],r4i3n[0-35],r4i4n ...

$ sinfo –T <reservation: Only display information about Slurm reservations>
RESV_NAME                STATE           START_TIME             END_TIME     DURATION  NODELIST
systemtime_april2019  INACTIVE  2019-04-01T06:00:00  2019-04-02T17:00:00   1-11:00:00  r1i0n[0-35],r1i1n[0-35],r1i2n[0-
35],r1i3n[0-35],r1i4n[0-35],r1i5n[0-35],r1i6n[0-35],r1i7n[0-35],r2i0n,r103u[01,03,05,07,09,11,13,15,17,19,21,23,25-
36],r104u[01,03,05,07,09,11,13,15,17,19,21,23,25,27,29,31,33],r105u[01,03,05,07,09,11,13,15,17,19,21,23,25,27,29,31,33] ...
r7i4n28_repair          ACTIVE  2019-03-14T09:00:00  2019-03-28T16:00:00  14-07:00:00  r7i4n[1,10,19,28]
hperepair-r5i3n11       ACTIVE  2019-03-15T10:00:00  2019-03-20T16:00:00   5-06:00:00  r5i3n[2,11,20,29]
workshop_mar2019      INACTIVE  2019-04-20T09:00:00  2019-04-20T15:00:00     06:00:00  r103u[15,17]
```

https://www.nrel.gov/hpc/eagle-monitor-control-commands.html

Sview on Eagle

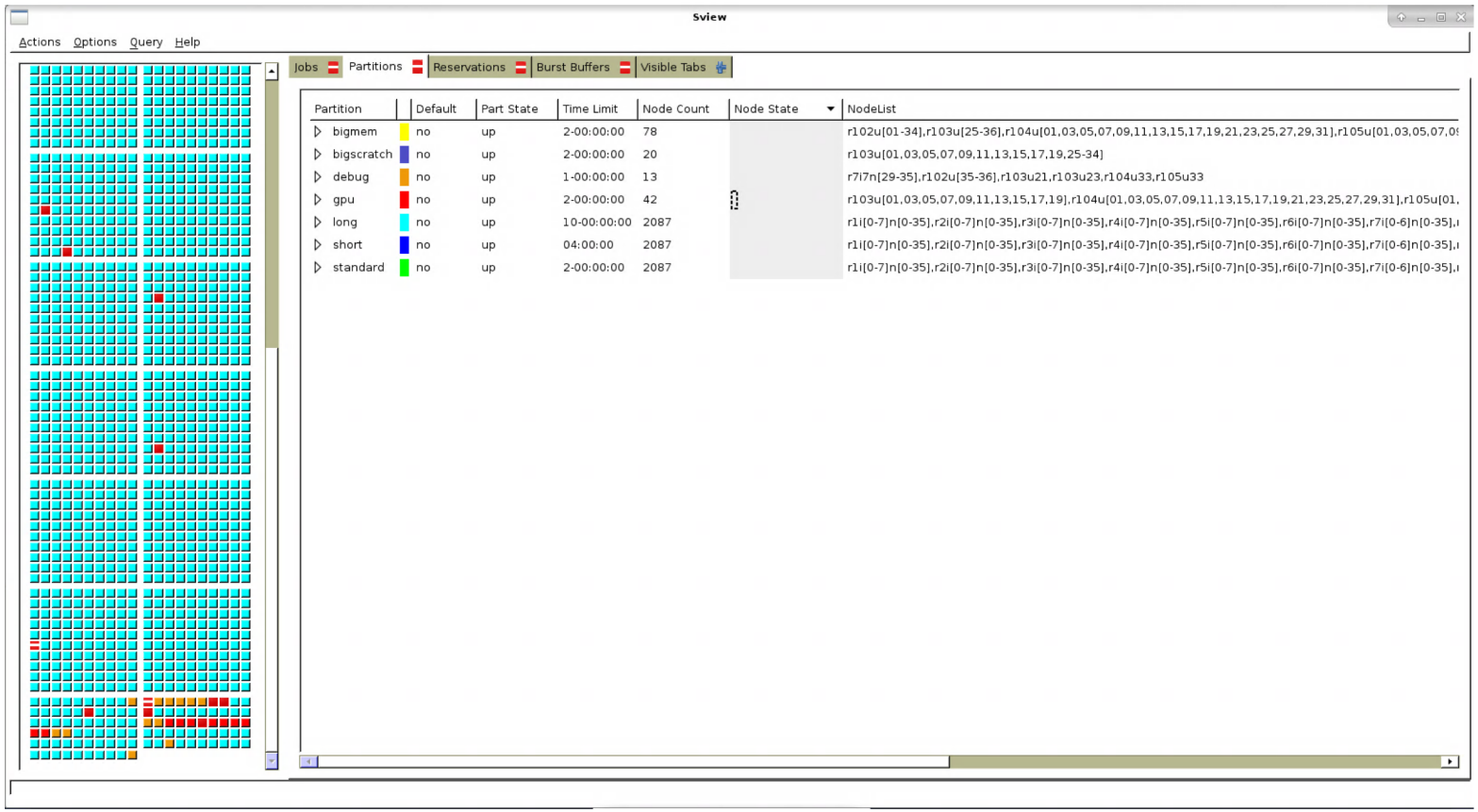
- Sview (with X11 forwarding)



"Sview can be used to view Slurm configuration, job, step, node and partitions state information.

https://www.nrel.gov/hpc/eagle-software-fastx.html

Sview showing Partitions



Sections

1. Job monitoring and forensics
2. Advanced Slurm functions (Flags)
3. Eagle best practices
4. Parallelizing with Slurm
5. GPU nodes
6. Questions
7. Follow up

Slurm functions (flags)

sbatch: Slurm batch script

srun: Slurm run command

NOTE: **sbatch** and **srun** generally have the same options with only a very few exceptions, highlighted later in this presentation.

Some selected Flags:

-n, --ntasks=<number>

sbatch does not launch tasks, it requests an allocation of resources and submits a batch script. This option advises the Slurm controller that job steps run within the allocation will launch a maximum of number tasks and to provide for sufficient resources. The default is one task per node, but note that the --cpus-per-task option will change this default.

--tasks, --ntasks-per-node=<ntasks>

Request that ntasks be invoked on each node. If used with the --ntasks option, the --ntasks option will take precedence and the --ntasks-per-node will be treated as a maximum count of tasks per node. Meant to be used with the --nodes option. This is related to --cpus-per-task=ncpus, but does not require knowledge of the actual number of cpus on each node. In some cases, it is more convenient to be able to request that no more than a specific number of tasks be invoked on each node.

-c, --cpus-per-task=<ncpus>

Request that ncpus be allocated per process. This may be useful if the job is multithreaded and requires more than one CPU per task for optimal performance. The default is one CPU per process.

-q, --qos=high

Request a high quality of service for the job. This will increase the starting priority of your job as well as account allocation usage x2

Slurm functions (flags) cont.

-K, --kill-on-bad-exit[=0|1]

Controls whether or not to terminate a step if any task exits with a non-zero exit code. If this option is not specified, the default action will be based upon the Slurm configuration parameter of KillOnBadExit. If this option is specified, it will take precedence over KillOnBadExit.

-k, --no-kill

Do not automatically terminate a job if one of the nodes it has been allocated fails. This option applies to job and step allocations. The job will assume all responsibilities for fault-tolerance. Tasks launch using this option will not be considered terminated (e.g. -K, --kill-on-bad-exit and -W, --wait options will have no effect upon the job step).

-m, --distribution=*|block|cyclic|arbitrary|plane=<options> [:*|block|cyclic|fcyclic[:*|block| cyclic|fcyclic]][,Pack|NoPack]

Specify alternate distribution methods for remote processes. This option controls the distribution of tasks to the nodes on which resources have been allocated, and the distribution of those resources to tasks for binding (task affinity).

See man pages for details.

--mem-bind=[{quiet,verbose},]type

Bind tasks to memory. This option is typically used to ensure that each task is bound to the memory closest to it's assigned CPU. **The use of any type other than "none" or "local" is not recommended.**

Sbatch only options

There is one main differentiation with `sbatch` operation, Job Arrays. Job Arrays are a great way to monitor and manage many jobs with identical resource requirements.

```
# Submit a job array with index values between 0 and 100
$ sbatch --array=0-100 -N1 array.sh↲

# Submit a job array with index values of 1, 3, 5 and 7
$ sbatch --array=1,3,5,7 -N1 array.sh

# Submit a job array with index values between 1 and 7 with a step size
# of 2 (i.e. 1, 3, 5 and 7)
$ sbatch --array=1-7:2 -N1 array.sh
```

Srun only options

- You can use srun many times from within a job allocation, either interactively or from a sbatch job script. Also, srun may be initiated outside of a job allocation. This will create a job allocation request and potentially wait in queue for requested resources.

- You can use the --exclusive option to run multiple parallel job steps within a single job allocation

```
"--exclusive

...

This option can also be used when initiating more than one job step within an existing resource allocation, where you
want separate processors to be dedicated to each job step. If sufficient processors are not available to initiate the
job step, it will be deferred. This can be thought of as providing resource management for the job within it's
allocation. Note that all CPUs allocated to a job are available to each job step unless the --exclusive option is
used plus task affinity is configured.

Since resource management is provided by processor, the --ntasks option must be specified, but the following options
should NOT be specified --relative, --distribution=arbitrary."
```

Slurm Flags for Forensics

Flags that may be useful for job forensics:

-o, --output=<filename pattern>

Instruct Slurm to connect the batch script's standard output directly to the file name specified in the "filename pattern". By default both standard output and standard error are directed to the same file. A frequently used Slurm token in the filename pattern is `%j`, which will be replaced with the numerical job ID in the resulting text output file produced by the job.

-v, --verbose

Increase the verbosity of sbatch's informational messages. Multiple -v's will further increase sbatch's verbosity. By default only errors will be displayed

Sbatch

Source reference links:

https://www.nrel.gov/hpc/eagle-sample-batch-script.html

https://www.nrel.gov/hpc/eagle-debugging-approaches.html

https://www.nrel.gov/hpc/eagle-software-arm.html

Sections

1. Job monitoring and forensics
2. Advanced Slurm functions (Flags)
3. Eagle best practices
4. Parallelizing with Slurm
5. GPU nodes
6. Questions
7. Follow up

Eagle Best Practices

- Local scratch - /tmp/scratch $LOCAL_SCRATCH big buffer per node, streaming large swaths of data to be captured, replaced. Preventative measure for avoiding overloading the Lustre file system.
- To be aware: There are NREL HPC project allocations (node hours sum) job /resource allocations with in Slurm – within your job. Your job will occupy the entire node requested with in your job and all of it's hardware, so please be cognizant to maximize resource utilization. In effect preventing other users access while on that particular node or nodes. *If you need help with Slurm job optimization please reach out to the HPC Operations team: https://www.nrel.gov/hpc/contact-us.html*

- To access specific hardware, we strongly encourage requesting by feature instead of specifying the corresponding partition:

```
# Request 8 "GPU" nodes for 1 day interactively
$ srun -t1-00 -N8 -A <handle> --gres=gpu:2 --pty $SHELL↲
```

Sections

1. Job monitoring and forensics
2. Advanced Slurm functions (Flags)
3. Eagle best practices
4. Parallelizing with Slurm
5. GPU nodes
6. Questions
7. Follow up

Srun

- **Srun** - This command is used to launch a parallel job step. Typically, srun is invoked from a Slurm job script (sbatch) to launch a MPI job (much in the same way that mpirun or mpiexec are used).

- Alternatively, srun can be run directly from the command line, in which case srun will first create a resource allocation for running the parallel job.

```
$ srun –A <account> -t <time> [...] --pty $SHELL↲
```

- Use srun instead of mpirun, as srun is Slurm's version of mpirun/mpiexec. Srun acts as a wrapper for mpirun and can capture the proper run environment for running mpi applications.

Parallel computing with Slurm

Categories:

- Distributed memory programs that include explicit support for MPI (message passing interfacing processes. These processes execute across multiple CPU cores and/or nodes.

- Multiple instances of the same program execute on multiple data files simultaneously, with each instance running independently from others on its own allocated resources Slurm job arrays offer a simple mechanism for achieving this.

- GPU (graphics processing unit) programs including explicit support for offloading to the device via languages like CUDA. https://www.nrel.gov/hpc/eagle-software-toolchains.html

Srun capabilities

- srun – Run parallel jobs <DEMO> sbatch -A csc000 -t 15 -N 10 node_cpu_rollcall. This will showing that it is trivial to parallelize jobs with Slurm.

- Slurm integrates with the MPI versions installed on eagle using srun. Srun brings mpi wrapper structure with easy slrum/jobstep/job allocation integration.

- srun will automatically use the mpirun from the MPI version that you compiled your code with, So you don't need to code in the specific version dependency to all your scripts.
- MPI Example:

```
#!/bin/bash
#SBATCH --partition=standard          # Name of Partition
#SBATCH --nodes=4                     # Number of nodes
#SBATCH --ntasks=100                  # Request 100 CPU cores
#SBATCH --time=06:00:00               # Job should run for up to 6 hours
#SBATCH --account=<project handle>  # Accounting
module purge
module load mpi/intelmpi/18.0.3.222
srun ./mympi_program
```

Script Used in Demo (node_rollcall.sh)

```
#!/usr/bin/env bash
#SBATCH --nodes=2                    # Change this number to get different outputs
#SBATCH -t 1
#SBATCH --job-name=node_rollcall
#SBATCH -o node_rollcall.%j          # output to node_rollcall.<job ID>

###### USAGE: sbatch –A <project_handle> -N <node amount> node_rollcall.sh

PROCS=$(($SLURM_NNODES * $SLURM_CPUS_ON_NODE))
echo "I am node $SLURMD_NODENAME and I am the master node of this job with ID $SLURM_NODEID"
echo "There are $SLURM_NNODES nodes in this job, and each has $SLURM_CPUS_ON_NODE cores, for a total of $PROCS cores."
printf "Let's get each node in the job to introduce itself:\n\n"

# Send an in-line bash script to each node to run. The single quotes prevent $var evaluation. srun uses all resources by default
srun bash <<< 'printf "\tI am $SLURMD_NODENAME, my ID for this job is $SLURM_NODEID\n"' &
wait

printf "\nLet's get each node to print the ranks of all their cores (concurrently!):\n\n"

srun --ntasks=$PROCS \
        bash <<< 'printf "n${SLURM_NODEID}:c"; awk "{print \$39}" /proc/self/stat' | tr '\n' ' '
echo
```

Sequential Job Steps

```
#!/bin/bash
#SBATCH --account=<allocation>
#SBATCH --time=4:00:00
#SBATCH --job-name=steps
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --output=job_output_filename.%j.out  # %j will be replaced
with the job ID

# By default, srun uses all job allocation resources (8 tasks each)
srun ./myjob.1a
srun ./myjob.1b
srun ./myjob.1c
```

Parallel Job Steps

```
#!/bin/bash
#SBATCH --account=<allocation>
#SBATCH --time=4:00:00
#SBATCH --job-name=steps
#SBATCH --nodes=8
#SBATCH --output=job_output_filename.%j.out

# Be sure to request enough nodes to run all job steps at the same time
srun -N 2 -n 36 -c 2 --cpu-bind=cores ./myjob.1 &
srun -N 4 -n 72 -c 2 --cpu-bind=cores ./myjob.2 &
srun -N 2 -n 28 -c 2 --cpu-bind=cores ./myjob.3 &
wait
```

Sections

1. Job monitoring and forensics
2. Advanced Slurm functions (Flags)
3. Eagle best practices
4. Parallelizing with Slurm
5. GPU nodes
6. Questions
7. Follow up

Review: resources available and how to request on Eagle

Resource	# of Nodes	Request
GPU	44 nodes total 22 nodes per user 2 GPUs per node	--gres=gpu:1 --gres=gpu:2
Big Memory	78 nodes total 40 nodes per user 770 GB max per node	--mem=190000 --mem=500GB
Big Scratch	20 nodes total 10 nodes per user 24 TB max per node	--tmp=20000000 --tmp=20TB

Remote Exclusive GPU usage

Using GPUs

#SBATCH --gres=gpu

After requesting a GPU node (srun/salloc), run nvidia-smi:

```
srun -A csc000 -t 15 --gres=gpu:2 --pty $SHELL
[username@r103u19 ~]$ nvidia-smi
Tue Mar 19 11:52:58 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.72       Driver Version: 410.72       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  Off  | 00000000:37:00.0 Off |                    0 |
| N/A   43C    P0    38W / 250W |      0MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-PCIE...  Off  | 00000000:86:00.0 Off |                    0 |
| N/A   39C    P0    37W / 250W |      0MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

GPU follow up info

- GPU batch example

```
#!/bin/bash
#SBATCH --nodes=2            # Use 2 nodes
#SBATCH --time 00:20:00      # Set a 20 minute time limit
#SBATCH --ntasks 2           # Maximum CPU cores for job
#SBATCH --gres=gpu:2         # GPU request
#SBATCH --mem=18400          # memory usage request

cd /scratch/<userid>/mydir
srun my_graphics_intensive_scripting
```

In this example sbatch requests 2 nodes at 20 minutes, 2 jobs per CPU with the use of 2 GPU's to srun the graphics scripting

Thank You

www.nrel.gov

NREL is a national laboratory of the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy, operated by the Alliance for Sustainable Energy, LLC.

NREL
NATIONAL RENEWABLE ENERGY LABORATORY