Introduction to Slurm

Kenny Peck
HPC Systems Administrator
NASA Center For Climate Simulation
Goddard Space Flight Center
Table of Contents (T.O.C.)

- Glossary
- Slurm core functions
- Slurm functions on your job’s node(s)
- Discover cluster resources
- Key Slurm commands
- Job-submission directives/options
- Simple job with sbatch
- Multi-node parallel MPI job
- List queued jobs with squeue
- Simple parallel job (no MPI)
- Customize squeue output
- Submit chained jobs with --depend
- Submit replicated jobs with --array
- Interactive session
- Slurm-defined environment variables
- MPI from interactive session
- Python/Dask example (single-node only)
### Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>Any of nodes, CPUs, GPUs, memory, time</td>
</tr>
<tr>
<td>Batch job</td>
<td>A list of commands in a script file</td>
</tr>
<tr>
<td>Interactive session</td>
<td>A shell running (on the head node) in an allocated set of resources dedicated to you</td>
</tr>
<tr>
<td>Success</td>
<td>A job completes and terminates well (exit code zero; canceled jobs are not considered successful)</td>
</tr>
<tr>
<td>Failure</td>
<td>Anything that lacks success (exits non-zero)</td>
</tr>
</tbody>
</table>
Slurm core functions (managed by centralized controller daemons)

- Resource manager
  - Manages sets of associated nodes, called *partitions*
  - Allocates resources (e.g., CPUs, memory, time) within each partition
- Scheduler
  - Manages and prioritizes queue of jobs from all users across all partitions
  - Dispatches jobs as requested resources become available
- Database manager
  - Authorizes/authenticates user to submit a job to a particular account, partition and quality of service (QoS)
  - Monitors and records resource usage per job (and job step)
Slurm functions on your job’s nodes

Control job environment and monitor job script
• Launch-time setup of user environment as specified at job submission
• Execution of job script as submitting user
• Monitor resource utilization (kill job if it exceeds requested resource limit)
• Orderly termination of job when script completes, or when time limit is exceeded
Discover cluster resources: definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Contains one or more sockets</td>
</tr>
<tr>
<td>Socket</td>
<td>Contains one processor</td>
</tr>
<tr>
<td>Processor</td>
<td>Contains one or more cores</td>
</tr>
<tr>
<td>Core</td>
<td>CPU, performs FLOPs</td>
</tr>
</tbody>
</table>
Discover cluster resources: Haswell nodes

2 Sockets/Node, 14 Cores/Processor; 4GB/core, 128GB total, shared by all of your job's computational tasks (e.g., MPI threads) across all CPUs

Examples:

- 28 tasks
  - 1 task per CPU
  - 4GB / task

- 4 tasks
  - 24 CPUs unused
  - 32GB / task

- 1 task
  - 27 CPUs unused
  - 128GB / task

etc.

1x Haswell Node

Socket 1

1 2 3 4 5 6 7
8 9 10 11 12 13 14

Socket 2

15 16 17 18 19 20 21
22 23 24 25 26 27 28

128GB RAM

www.nasa.gov

Introduction to Slurm – Brown Bag

Back to T.O.C.
Discover cluster resources: Skylake nodes

2 Sockets/Node, 20 Cores/Processor; 4.75GB/core, 196GB total, shared by all of your job's computational tasks (e.g., MPI threads) across all CPUs

- 1x Skylake Node
- 196GB RAM
- 1 task per CPU
- 196GB / task
- 48GB / task
- 4.75GB / task
- 39 CPUs unused
- 36 CPUs unused
- 4 tasks
- 40 tasks
- etc.

www.nasa.gov
Resource request for parallel tasks

Request for resources needs to be aware of CPUs in the context of required memory

40 tasks, requires 256GB total memory
   2 Haswell nodes
   128GB per node, shared among all tasks that are local to each node

40 tasks, 4.5GB per CPU requires 180GB
   2 Haswell nodes
   Or
   1 Skylake

We'll show you Slurm directives in about 5 slides.
Coming up next

• Brief introduction to some Slurm commands
• Typical use patterns at NCCS
• Examples
## Key Slurm commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit a batch job</td>
<td>sbatch</td>
<td>Batch mode; submits a job to the queue for execution when resources become available</td>
</tr>
<tr>
<td>Submit interactive job</td>
<td>alloc</td>
<td>Allocates resources and attaches a shell for interactive use</td>
</tr>
<tr>
<td></td>
<td>xalloc</td>
<td>NCCS <code>salloc variant</code> that sets up X11 forwarding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(see “Running jobs” → “Submit an interactive job with X11 forwarding”)</td>
</tr>
<tr>
<td>Run command on subset of job resources</td>
<td>srun</td>
<td>Executes a single command or script (creates an allocation if needed; for best results, only use inside sbatch script or alloc session)</td>
</tr>
<tr>
<td>Observe a job</td>
<td>sattach</td>
<td>Connect to stdin, stdout, stderr for a running job</td>
</tr>
</tbody>
</table>
## Key Slurm commands – cont’d

<table>
<thead>
<tr>
<th>Action</th>
<th>Command</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel/delete a submitted job</td>
<td><code>scancel</code></td>
<td></td>
</tr>
<tr>
<td>Check job state</td>
<td><code>squeue</code></td>
<td>Pending (queued), running, completing</td>
</tr>
<tr>
<td>Details about job</td>
<td><code>scontrol show job jobid</code></td>
<td>Detailed information about a job. Quite verbose; use rarely.</td>
</tr>
<tr>
<td>List partitions or nodes</td>
<td><code>sinfo</code></td>
<td>View information about partition and node states.</td>
</tr>
</tbody>
</table>
Slurm command basics

- Slurm includes man pages available for all commands
  
  ```
  $ export MANPATH=$MANPATH:/usr/slurm/share/man
  % setenv MANPATH=$MANPATH:/usr/slurm/share/man
  ```

- **--help**  print brief description
- **--usage** print list of options

- Most commands have options in two formats:
  
  **Single-letter**  -p datamove
  **Long-name**  `--partition=datamove`

- **Time format:**
  
  ```
  days-hours:minutes:seconds
  minutes
  ```

- **Non-zero exit code indicates failure**

Back to T.O.C.
Slurm command basics – cont’d

• The Slurm stdout (or stderr) file will be appended, not overwritten (if it exists).
• Slurm processes that are launched with srun are not run under a shell, so none of the following are executed: ~/.profile ~/.bashrc ~/.login ~/.cshrc, etc.
• NCCS note: by default, SBU accounting is by node count, rather than by CPU. Said differently, you are charged for each full node your job allocates, regardless of whether you specify fewer than the maximum number of CPUs/node.
<table>
<thead>
<tr>
<th>Action</th>
<th>Directive/command-line option</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name the job</td>
<td><code>#SBATCH -J jobname</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --job-name=jobname</code></td>
<td></td>
</tr>
<tr>
<td>Charge to account</td>
<td><code>#SBATCH -A s1234</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --account=s1234</code></td>
<td></td>
</tr>
<tr>
<td>Partition</td>
<td><code>#SBATCH -p datamove</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --partition=datamove</code></td>
<td></td>
</tr>
<tr>
<td>Request 2 nodes</td>
<td><code>#SBATCH -N 2</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --nodes=2</code></td>
<td></td>
</tr>
<tr>
<td>Request one hour</td>
<td><code>#SBATCH -t 60</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --time=1:00:00</code></td>
<td></td>
</tr>
<tr>
<td>Send error output to file</td>
<td><code>#SBATCH -e path/%x-%j.err</code></td>
<td>Slurm autofills %x with jobname and %j with job ID</td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --error=path/%x-%j.err</code></td>
<td></td>
</tr>
<tr>
<td>Send standard output to file</td>
<td><code>#SBATCH -o path/%x-%j.out</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>#SBATCH --output=path/%x-%j.out</code></td>
<td></td>
</tr>
</tbody>
</table>
### Job-submission directives/options - cont’d

<table>
<thead>
<tr>
<th>Action</th>
<th>Directive/command-line option</th>
</tr>
</thead>
</table>
| Memory required per CPU (per task, assuming 1 task/CPU; defaults to MB units) | #SBATCH --mem-per-cpu=1024  
#SBATCH --mem-per-cpu=1G |
| Memory required per node                                               | #SBATCH --mem=3.2G                                                      |
| Total number of tasks in job                                           | #SBATCH -n 560  
#SBATCH --ntasks=560                                                     |
| Number of tasks per node                                               | #SBATCH --ntasks-per-node=24                                            |
| Set job working directory to path other than submission directory (not required) | #SBATCH -D /full/path                                                  |
| E-mail recipient                                                       | #SBATCH --mail-user=My.Name@nasa.gov                                   |
| E-mail when job [completes|fails|starts]                           | #SBATCH --mail-type=[END|FAIL|BEGIN]                                   |

**Back to T.O.C.**
Submit a simple job with sbatch

- **Hello-world job**
  - task: print hostname
  - resources: 1 node, 1 task, 100 MB per CPU, 5 minutes

- **Options to specify requested resources**
  - command line options
  - special comments in the job script
  - command line supersedes in-file
Command-line options

hello.sh

```
#!/bin/bash
hostname
exit $?
```

$ sbatch -N1 -n1 --mem-per-cpu=100 -t 00:05:00 hello.sh

1 node
1 task
at least 100 MB per CPU
5 minutes of wall time
the script

Submitted batch job 184917

megabytes
Command-line options (using long names)

```bash
$ sbatch --nodes=1 --ntasks=1 --time=00:05:00 hello.sh
```

1 node
1 task
5 minutes of wall time
the script
In-line directives in job script

hello.sh

#!/bin/bash
#SBATCH --nodes=1 --ntasks=1 --memory-per-cpu=100M --time=00:05:00
#SBATCH --job-name=hostname
hostname
exit $?

$ sbatch hello.sh
Submitted batch job 184917

Give it a meaningful name
$ scontrol show job 184917

JobId=184917  Name=hostname  UserId=user1(123456789)
  GroupId=s1234(41234)
  ...
  JobState= PENDING  Reason=Priority Dependency=(null)
  ...
  ...
  Partition=compute  AllocNode:Sid=discover16:13607
  ...
  NumNodes=1-1  NumCPUs=1  CPUs/Task=1  Req:C:T=*:*:*
  ...
  MinCPUsNode=1  MinMemoryCPU=100M  MinTmpDiskNode=0
  ...
  Command=/gpfs/m/dhome/user1/slurm/hello.sh
  WorkDir=/gpfs/m/dhome/user1/slurm
  Comment=stdout=/gpfs/m/dhome/user1/slurm/slurm-184917.out
Prepare a multi-node parallel MPI job

① Program
② Compilation
③ Script
④ Submission
⑤ Reading output

① hellompi.f90

```fortran
program hellompi

  implicit none
  include 'mpif.h'
  integer rank, ierr, irc

  call mpi_init(ierr)
  call mpi_comm_rank(MPI_COMM_WORLD, rank, ierr)
  print *, "Hello from rank", rank
  call mpi_finalize(irc)

end program hellompi
```

One of the simplest possible MPI Fortran programs.

② compilation

```
$ module load comp/intel/19.1.3.304 mpi/impi/19.1.3.304
$ mpif90 hellompi.f90 -o hellompi.exe
```

www.nasa.gov
Prepare job script and submit

③ script in hellompi.sh

```
#!/bin/bash
#SBATCH --job-name=test_mpi --time=00:01:00
#SBATCH --nodes=2 --ntasks-per-node=10
#SBATCH --mail-user=user1@nccs.nasa.gov --mail-type=ALL
#SBATCH --output=run.stdout
#SBATCH --error=run.stderr

./usr/share/modules/init/bash
module purge
module load comp/intel/19.1.3.304 mpi/impi/19.1.3.304

cd /home/user1/slurm/pi
mpirun ./hellompi.exe
exit
```

loads module() into shell

needed for mpirun

④ submission to default partition

```
$ sbatch hellompi.sh
```

Back to T.O.C.
Output – standard output

Hello from rank 11
Hello from rank 2
Hello from rank 14
Hello from rank 8
Hello from rank 13
Hello from rank 7
Hello from rank 17
Hello from rank 4
Hello from rank 18
Hello from rank 3
Hello from rank 6
Hello from rank 9
Hello from rank 1
Hello from rank 5
Hello from rank 0
Hello from rank 10
Hello from rank 19
Hello from rank 16
Hello from rank 15
Hello from rank 12
...

stdout from run.stdout

Job Resource Usage Summary for 41640995

CPU Time Used : 00:04:40
Memory Used : 1376K
Virtual Memory Used : 188340K
Walltime Used : 00:00:05
Estimated SBUs : 0

Memory Requested : 125300Mn (n=per node; c=per core)
CPUs Requested / Allocated : 20 / 56
Walltime Requested : 01:00:00

Execution Queue : compute
Head Node : borgm038
Charged to : k3000
Job Stopped : Mon Nov 30 18:46:30 EST 2020
Output – standard error

stderr from run.stderr

+ cd /home/user1/slurm/pi
+ mpirun ./helompi.exe
+ exit

as a result of 'set –x' in script
See all my queued jobs

```
$ squeue -u user1
```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>41640995</td>
<td>compute</td>
<td>test_mpi</td>
<td>user1</td>
<td>PD</td>
<td>0:00</td>
<td>2</td>
<td>(Resources)</td>
</tr>
</tbody>
</table>

Job is queued and is pending in the default "compute" partition waiting for resources to become available.

- Please don’t specify the compute partition explicitly when you submit a job.
- Please don’t run `squeue` inside a short-interval loop (e.g., the watch command) in your terminal window or monitoring script.
View job details

```bash
$ scontrol show job 41640995
JobId=41640995 JobName=test_mpi
   UserId=user1(123456789) GroupId=s1234(41234) MCS_label=N/A
   Priority=10500 Nice=0 Account=s1234 QOS=inter
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:05 TimeLimit=01:00:00 TimeMin=N/A
   RunTime=00:00:05 TimeLimit=01:00:00 TimeMin=N/A
   NodeList=borgm[038-039]
   BatchHost=borgm038
   NumNodes=2 NumCPUs=56 NumTasks=2 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=56,mem=250600M,node=2,billing=56
   WorkDir=/gpfsm/dhome/user1
   StdErr=/gpfsm/dhome/user1/run.stderr
   StdIn=/dev/null
   StdOut=/gpfsm/dhome/user1/run.stdout
   ...
```
Next...

- Simple parallel jobs without MPI
- View information about queued jobs
  - working with squeue
  - getting only the information you need with custom formats
Simple parallel job (no MPI)

Note: NCCS generally recommends you always use sbatch or salloc to submit your job, then use srun inside your job script to control the allocation of subsets of resources to specific commands.

```
$ srun --label --ntasks=24 --nodes=2 --qos=debug hostname
```

--label causes task numbers to be prepended to output lines:

```
srun.slurm: job 208384 queued and waiting for resources
srun.slurm: job 208384 has been allocated resources
08:borgb107
17:borgb113
18:borgb113
20:borgb113
21:borgb113
23:borgb113
13:borgb113
14:borgb113
00:borgb107
02:borgb107
06:borgb107
10:borgb107
11:borgb107
16:borgb113
22:borgb113
04:borgb107
15:borgb113
09:borgb107
19:borgb113
12:borgb113
03:borgb107
05:borgb107
01:borgb107
07:borgb107
```
Customize squeue output

$ squeue -u user1

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>18497</td>
<td>compute</td>
<td>hostname</td>
<td>user1</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
</tbody>
</table>

Additional info from -l/--long option; estimated start time (if any) from --start. See man squeue for many, many options.

Customize output with -o/--format:

$ squeue -u user1 -format="%.15i %.15j %.2t %.10M %.10l %.20S"

<table>
<thead>
<tr>
<th>JOBID</th>
<th>NAME</th>
<th>ST</th>
<th>TIME</th>
<th>TIME_LIMIT</th>
<th>START_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>41148588</td>
<td>hostname</td>
<td>R</td>
<td>5:52:59</td>
<td>12:00:00</td>
<td>2020-11-06T06:45:35</td>
</tr>
</tbody>
</table>

Customize output with SQUEUE_FORMAT:

$ SQUEUE_FORMAT="%.15i %.15j %.2t %.10M %.10l %.20S"

$ squeue -u user1

<table>
<thead>
<tr>
<th>JOBID</th>
<th>NAME</th>
<th>ST</th>
<th>TIME</th>
<th>TIME_LIMIT</th>
<th>START_TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>41148588</td>
<td>hostname</td>
<td>R</td>
<td>5:52:59</td>
<td>12:00:00</td>
<td>2020-11-06T06:45:35</td>
</tr>
</tbody>
</table>
Jobs with dependencies

Submit job1:

```bash
$ sbatch job1.sh
Submitted batch job jobid1
```

Run job2 only after job jobid1 finishes successfully:

```bash
$ sbatch job2.sh --dependency=afterok:jobid1
```

Now, suppose job1 might run so fast that neither I nor my script has enough time to capture the job ID, so by the time job2 is submitted job1 is all gone....
Jobs with dependencies: hold and release

Submit job1, but hold it:

```bash
$ sbatch --hold job1.sh
Submitted batch job jobid1
```

Submit job2:

```bash
$ sbatch job2.sh --dependency=afterok:jobid1
```

Release job1 so it can run:

```bash
$ scontrol release jobid1
```
Job arrays: minimally varying jobs

Submit a set of jobs whose only variance is an index value for input and output filenames or directories. Cap the number of simultaneously running jobs at 10 (with %10 notation):

```bash
$ cat job_array_1.sh
#!/bin/bash
#SBATCH --N1 -t 1:00:00 --job-name=my_array
#SBATCH --array=0-99%10
#SBATCH --directory=/home/user1/proj_data
cd ${SLURM_ARRAY_TASK_INDEX}
$HOME/bin/my_executable < input > output
```

Submit a set of jobs whose only variance is an index value for input and output filenames or directories. Cap the number of simultaneously running jobs at 10 (with %10 notation):

```
$ sbatch job_array_1.sh
Submitted batch job 41129733
```

Inspect job array right after it starts; one task running, the rest still awaiting allocation:

```
$ squeue -j 41129733
   JOBID         NAME   ST NODELIST(REASON)
41129733_[1-99%10] my_array PD (Priority)
41129733_0     my_array R borgn087
```

Array indeces can also vary by stride; see example (under Running Jobs->Submit replicated jobs) here: https://www.nccs.nasa.gov/nccs-users/instructional/using-slurm
Interactive session

- Allocates resources
- Provides access to them
- Performs ssh to head compute node
- Does not start any application
- Environment is copied from the calling shell

Allocate two nodes using the "inter" quality of service (QoS):

```bash
$ salloc -A k3000 --qos=inter --nodes=2
salloc: Pending job allocation 41640673
salloc: job 41640673 queued and waiting for resources
salloc: job 41640673 has been allocated resources
salloc: Granted job allocation 41640673
borgj111:~ $ 
```
View job configuration with `scontrol`

```
borgj111:~ $ scontrol show job
JobId=41640673 JobName=sh
    UserId=user1(123456789) GroupId=s1234(41234) MCS_label=N/A
    Priority=1050 Nice=0 Account=s1234 QOS=inter
    JobState=RUNNING Reason=None Dependency=(null)
    RunTime=00:00:44 TimeLimit=01:00:00 TimeMin=N/A
    AccrueTime=2020-11-30T18:01:55
    StartTime=2020-11-30T18:02:09 EndTime=2020-11-30T19:02:12 Deadline=N/A
    PreemptEligibleTime=2020-11-30T18:02:09 PreemptTime=None
    SuspendTime=None SecsPreSuspend=0 LastSchedEval=2020-11-30T18:02:09
    Partition=compute AllocNode:Sid=discover19:249464
    ReqNodeList=(null) ExcNodeList=(null)
    NodeList=borgj[111,113]
    BatchHost=borgj111
    NumNodes=2 NumCPUs=56 NumTasks=2 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
    TRES=cpu=56,mem=250600M,node=2,billing=56
    ...
```

Default limit: 1 hour may change (so specify)
Slurm-defined environment variables

$ env | grep -e SLURM
SLURM_NODELIST=borgj[111,113]
SLURM_JOB_NAME=sh
SLURMD_NODENAME=borgj111
...
SLURM_JOB_QOS=inter
...
SLURM_NNODES=2
SLURM_STEP_NUM_NODES=1
SLURM_JOB_ID=41640673
...
SLURM_TASKS_PER_NODE=28(x2)
...
SLURM_JOB_USER=user1
SLURM_STEPID=0
...
SLURMPTY_WIN_COL=208
SLURM_UMASK=0077
...
SLURM_CPUS_ON_NODE=28
SLURM_PROCID=0
...
SLURM_JOB_CPUS_PER_NODE=28(x2)
...
SLURM_JOB_NUM_NODES=2
...
MPI program from the interactive session

```bash
borg $ mpirun ./hellomi.exe

Hello from rank 0
Hello from rank 22
Hello from rank 8
Hello from rank 30
Hello from rank 6
...
Hello from rank 14
Hello from rank 55
```

`mpirun` by default will use all available cores

```
SLURM_NNODES=2
SLURM_CPUS_ON_NODE=28
```
### MPI program from the interactive session

**borgj111 $ mpirun -perhost 1 ./hellompi.exe**

| Hello from rank | 0 |
| Hello from rank | 1 |

1 per host = 2

**borgj111 $ mpirun -perhost 3 ./hellompi.exe**

| Hello from rank | 0 |
| Hello from rank | 4 |
| Hello from rank | 5 |
| Hello from rank | 1 |
| Hello from rank | 2 |
| Hello from rank | 3 |

3 per host = 6
Prepare a multi-core Python/Dask job

① dask_matrix.py

```python
#!/usr/bin/env python
import dask
import dask.array as da
import datetime as dt

# Create a large Dask array
da_array = da.random.random((200000, 200000), chunks=(1000, 1000))

print("Dask array Shape: ", da_array.shape)
print("Dask array Size: ", da_array.size)

# Create a task graph for computing the mean and std
mean_arr = da_array.mean()
std_arr = da_array.std()

... 

delta = end_time - beg_time
elapsedTime = ((1000000 * delta.seconds + delta.microseconds) / 1000000.0)
print("Mean = {} - STD = {} --> Time for combined calcs: {} s".format(mean_val, std_val, elapsedTime))
```
**Prepare job script and submit**

② **job script in python_dask_job.csh**

```bash
#!/bin/csh -f
#SBATCH --time=00:30:00
#SBATCH --job-name=dask_array
#SBATCH --ntasks=28
#SBATCH --constraint=hasw
#SBATCH -A s1234
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END

umask 022
source /usr/share/modules/init/csh
module purge
module load python/GEOSpyD/Ana2019.10_py3.7

python example_dask_matrix.py
```

loads module() into shell

③ **submission to default partition**

```bash
$ sbatch python_dask_job.csh
```
Thank You!

As always, feel free to contact the NCCS User Services Group with questions or problems.

301-286-9120

support@nccs.nasa.gov