Interacting with the queuing system

Allocating resources: batch or interactive

- @ NSC
  - `$ sbatch [resource req.] job-script.sh`
    - Once submitted, no user interaction
    - Only performs job-script.sh instructions
    - Suitable for production simulations
  - `$ interactive [resource req.]`
    - Direct access to node(s)
    - Suitable for troubleshooting and testing

Some common SLURM options

Short	Long	Explanation
-t	--time	Time limit for the job
-n	--ntasks	How many parallel processes your job will start
-c	--cpus-per-task	How many processors are needed for a single task
-A	--account	What account should this job be run under
-J	--job-name	Name for the job allocation
-N	--nodes	How many nodes to request
-C	--constraint	Required node features
-e	--error	File in which to store job error messages
-o	--output	File in which to store job output messages
	--reservation	Allocate resources for the job from the named reservation
	--mail-type	Notify user by email when certain event types occur.

Three ways of specifying resource req. (SLURM options): in order of increasing priority

1. Specified in a jobscript

```
#!/bin/bash
#SBATCH -t 01:00:00           # Time limit for the job
#SBATCH -n 1                  # How many parallel processes your job will start
#SBATCH -A my-project-code    # What account should this job be run under
```

2. Set in the environment

```
$ export SBATCH_TIMELIMIT=01:00:00
$ export SLURM_NTASKS=1
$ export SLURM_JOB_ACCOUNT=my-project-code
```

3. Specified as command line options

```
$ sbatch -t 01:00:00 -n 1 -A my-project-code job-script.sh
```

Allocating resources

- Know the parallelism model that your code uses
  - shared memory (OpenMP)
  - distributed memory (MPI)
  - hybrid
  - …

Example: Shared memory (OpenMP) using 16 threads

```
#!/bin/bash

#SBATCH -t 01:00:00              # Time limit for the job
#SBATCH -n 1                     # How many parallel tasks your job will start
#SBATCH -c 16                    # How many processors are needed for a single task
#SBATCH -A my-project-code       # What account should this job be run under
#SBATCH -J my-openmp-job         # Name for the job allocation

# Set the number of OpenMP threads based on the SLURM cpus per task variable
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Execute my program
./openmp-program
```

Example: Distributed memory (MPI) using 128 tasks

```
#!/bin/bash

#SBATCH -t 01:00:00            # Time limit for the job
#SBATCH -n 128                 # How many parallel tasks your job will start
#SBATCH -A my-project-code     # What account should this job be run under
#SBATCH -J my-mpi-job          # Name for the job allocation

# Execute my program using the NSC mpi launcher (mpprun)
mpprun mpi-program
```

Example: Distributed memory (MPI) using 128 tasks using installed module

```
#!/bin/bash

#SBATCH -t 01:00:00           # Time limit for the job
#SBATCH -n 128                # How many parallel tasks your job will start
#SBATCH -A my-project-code    # What account should this job be run under
#SBATCH -J my-mpi-job         # Name for the job allocation

# Load module (module purge first)
module purge
module load my-favorite-software/1.2.3-nsc1-intel-2018a-eb

# Execute my program using the NSC mpi launcher (mpprun)
mpprun my-favorite-mpi-program
```

Example: Hybrid (MPI + OpenMP) model (32 tasks, 4 threads per task)

```
#!/bin/bash

#SBATCH -t 01:00:00              # Time limit for the job
#SBATCH -n 32                    # How many parallel tasks your job will start
#SBATCH -c 4                     # How many processors are needed for each task
#SBATCH -A my-project-code       # What account should this job be run under
#SBATCH -J my-hybrid-job         # Name for the job allocation

# Set the number of OpenMP threads based on the SLURM cpus per task variable
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Execute my program using the NSC mpi launcher (mpprun)
mpprun hybrid-program
```

Many serial programs

- There are several ways to start many independent serial instances within one jobscript, e.g.
  - srun --multi-prog (simple example)
  - gnu parallel

Requesting resources: hybrid systems

- there are many SLURM features for the control of allocations containing GPUs.

  `(--gpus, --gpus-per-task, --gpu-bind, --gpus-per-node, --mem-per-gpu, ..)`

- this means there are many ways in which you can give conflicting allocation directives (ie. screw up)

Requesting resources: hybrid systems

- Allocating GPUs correctly depends on:
  - The system
  - The center operating the system
  - Intended purpose
  - …
- Read the documentation: “*If it isn’t documented, it doesn’t exist*”

Requesting resources: hybrid systems

- @NSC
  - Tetralith
    - GPU user guide
    - Running demanding accelerated OpenGL applications
  - Sigma: GPU user guide

Monitoring your job

Runtime monitoring

- Queue status
  - For systems running SLURM
    - `$ squeue -u [user id]`

Runtime monitoring

- Find and use the tools provided by your center
  - For systems running SLURM
    - `$ sstat -j [job id]`
  - @ NSC:
    - `$ jobload [jobid]`
    - More information on monitoring

Runtime monitoring

1. Login to (one) of your running compute nodes
   - @NSC: `$ jobsh [node label]`
2. Examine the status of your running job
   - top, htop
   - perf top
   - hwloc-ps
   - collectl
   - …

Runtime monitoring: Target

1. All the allocated resources (cores) are being utilized (@ close to 100%)
2. A relatively low amount of time is being spent in communication
3. Memory use is not close to the node limit

Post query and logs

- For systems that use SLURM, seff is your friend

seff example:

```
[struthers@tetralith1 testrun]$ seff 12730826
Job ID: 12730826
Cluster: tetralith
User/Group: struthers/struthers
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 32
CPU Utilized: 01:23:47
CPU Efficiency: 81.40% of 01:42:56 core-walltime
Job Wall-clock time: 00:03:13
Memory Utilized: 10.96 GB
Memory Efficiency: 12.07% of 90.75 GB
[struthers@tetralith1 testrun]$
```

Memory management

Memory management: Out Of Memory (OOM)

- A common cause for a job to fail is exhausting the memory on one or more nodes
- How to check:
  a. Determine memory use while the job is running (e.g. run `top` on one of the compute nodes)
  b. Use the `seff` command when the job has ended

Memory management

- Things to try (if you are OOM'ed)
    - Use nodes with more memory
    - For MPI applications
        - Use less cores per compute node
        - Use more MPI tasks

Example: MPI model (Tetralith)

Original (4 Nodes, 128 tasks) – FAIL with OOM

```
#!/bin/bash

#SBATCH -t 01:00:00              # Time limit for the job
#SBATCH -n 128                   # How many parallel tasks your job will start
#SBATCH -A my-project-code       # What account should this job be run under
#SBATCH -J my-mpi-job            # Name for the job allocation

# Execute my program using the NSC mpi launcher (mpprun)
mpprun mpi-program
```

1. Modified to increase effective memory per task (4 Nodes, 64 tasks)

```
#!/bin/bash

#SBATCH -t 01:00:00              # Time limit for the job
#SBATCH -n 64                    # How many parallel tasks your job will start
#SBATCH --ntasks-per-node=16     # Default=32
#SBATCH -A my-project-code       # What account should this job be run under
#SBATCH -J my-mpi-job            # Name for the job allocation

# Execute my program using the NSC mpi launcher (mpprun)
mpprun mpi-program
```

2. Modified to increase effective memory per task (8 Nodes, 128 tasks)

```
#!/bin/bash

#SBATCH -t 01:00:00            # Time limit for the job
#SBATCH -n 128                 # How many parallel tasks your job will start
#SBATCH --ntasks-per-node=16  # Default=32
#SBATCH -A my-project-code     # What account should this job be run under
#SBATCH -J my-mpi-job          # Name for the job allocation

# Execute my program using the NSC mpi launcher (mpprun)
mpprun mpi-program
```