# HPC Job Submission & High Throughput Computing

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Be Boulder.

# HPC Job Submission & High Throughput Computing


- Daniel Trahan
- *Email: Daniel.Trahan@Colorado.edu*

- *RC Homepage: https://www.colorado.edu/rc*

Sign in! http://tinyurl.com/curc-names

- Slides available for download at: https://github.com/ResearchComputing/Fundamentals_HPC_Spring_2019

*Adapted from slides by Andrew Monaghan, Aaron Holt, John Blaas, and Shelley Knuth: 1, 2, 3, 4.*

# Outline

- General Info
- Examples of submitting jobs to the supercomputer!
  - Simple Batch jobs
  - Advanced Batch jobs: running programs, mpi
  - Interactive jobs
  - Load balancing

# Hardware: Summit Supercomputer

- 475 compute nodes (Intel Xeon Haswell)
- 24 cores per node
- 11,400 total cores
- Omni-Path network
- 1.2 PB scratch storage
- GPFS File system

- 67% CU, 23% CSU, 10% RMACC

# Additional Types of Summit Compute Nodes

- 10 Graphics Processing Unit (GPU) Nodes
  - NVIDIA Tesla K80 (2/node)
- 5 High Memory Nodes
  - 2 TB of memory/node, 48 cores/node
- Phi Nodes
  - 20 nodes
  - Intel Xeon Phi

# RC Access: Logging in

- For this tutorial, we will be using accounts on RC resources
- In a terminal or Git Bash window, type the following:

```
ssh -X user####@tlogin1.rc.colorado.edu
```

- Type your password when you receive a prompt that says:

```
password:
```

***Note that `user####` is a temporary account that I have assigned to you. If you don't have one, let me know.*

# Working on RC Resources

- When you first log in, you will be on a login node. Your prompt will look like this (e.g.):

```
[user####@tlogin1 ~]$
```

- The login nodes are lightweight virtual machines primarily intended to serve as 'gateways' to RC resources. If you plan to work on Summit (most will), your first step should always be to move to a Summit 'scompile node':

```
ssh scompile
hostname
```

- Now download the material for this workshop:

```
git clone https://github.com/ResearchComputing/Fundamentals_HPC_Spring_2019
```

# Useful Slurm Commands: sbatch

- **sbatch**: submit a batch job to slurm
- You can use a variety of flag in a batch script or on the command line
- Useful to put in script so have for future use
- Example:

```
sbatch test.sh
```

- OR

```
sbatch --partition=shas test.sh
```

http://slurm.schedmd.com/sbatch.html

# SBATCH Options

Specified at command line or in job script as…

`#SBATCH <options>` …where options include:

- Allocation: `--account=<account_name>`
- Partition: `--partition=<partition_name>`
- Sending emails: `--mail-type=<type>`
- Email address: `--mail-user=<user>`
- Number of nodes: `--nodes=<nodes>`
- Number of Tasks `--ntasks=<number-of-tasks>`
- Quality of service: `--qos=<qos>`
- Reservation: `--reservation=<name>`
- Wall time: `--time=<wall time>`
- Job Name: `--job-name=<jobname>`

*More on slurm commands: https://slurm.schedmd.com/quickstart.html*

*FYI: You do NOT actually type <> above – this designates something specific you as a user must enter about your job*

# Available Partitions (--partition)

| Partition (sub-partition) | Description | # of nodes | cores/node | GPUs/node |
|---|---|---|---|---|
| shas<br>(shas-testing)<br>(shas-interactive) | General Compute (Haswell) | ~450 | 24 | 0 |
| sgpu<br>(sgpu-testing) | GPU-enabled nodes | 11 | 24 | effectively 4 |
| smem | High-memory nodes | 5 | 48 | 0 |
| sknl<br>(sknl-testing) | Phi (Knights Landing) nodes | 20 | 68 | 0 |

# Quality of Service (--qos)

| QoS | Description | Max wall time | Max jobs/user | Max nodes/user |
|---|---|---|---|---|
| normal | Default QoS | Derived from partition | n/a | 256 |
| testing | For quick turnaround when testing | 30 M | 1 | 24 cores* |
| interactive | For interactive jobs (command or GUI) | 4 H | 1 | 1 core |
| long | For jobs needing longer wall times | 7 D | n/a | 20 |

* testing QoS allows for 24 cores total allocated across any number or nodes.

# Practice Job Submission Examples

# Submit Your First Job!

- Submit a Slurm job with the following instructions:
  1. The job should run the Linux “hostname” command
  2. The job will be submitted from a bash script named submit_hostname.sh
  3. The job will run on 1 core of 1 node
  4. We will request a 1 minute wall time
  5. Run from the testing QOS
  6. Run on the shas partition
  7. Use the ‘tutorial2’ reservation
     *(Note: This is only for this workshop!)*

# submit_hostname.sh

```
#!/bin/bash
#SBATCH --nodes=1                                  # Number of requested nodes
#SBATCH --ntasks=1                                 # Number of requested cores
#SBATCH --time=0:01:00                             # Max wall time
#SBATCH --qos=testing                              # Specify QOS
#SBATCH --partition=shas-testing                   # Specify Summit Haswell nodes
#SBATCH --output=hostname_%j.out                   # Rename standard output file
#SBATCH --reservation=job_submission_2019          # Reservation (workshop only)

# Written by:                 Shelley Knuth, 15 July 2016
# Updated by:                 Andy Monaghan, 8 March 2018
# Purpose:                    Demonstrate how to run a batch job on RC resources

# purge all existing modules
module purge

hostname
```

# Running the job script

Submit the job:

```
sbatch submit_hostname.sh
```

Check the status of the job:

```
squeue / $ squeue -u <user> /
squeue -q <qos>
```

…or

```
sacct / $ sacct --format=<options>
```

…or

```
scontrol show job <job number>
```

Look at the job output:

```
more sbatch hostname_NNNNNN.out
```

*(*note that NNNNNN is your job number)*

[More on slurm commands: https://slurm.schedmd.com/quickstart.html](https://slurm.schedmd.com/quickstart.html)

# Your turn

- Create a Slurm job script and submit it as a job, with the following instructions:

1. Name it 'submit_sleep.sh'
2. The job should run first the whoami command, then the Linux "sleep" command for 30 seconds, then the hostname command
   - Syntax for these Linux commands is:

```
echo "I am" `whoami`
echo "Running on host" `hostname`
echo "Starting Sleep"
sleep 30
echo "Ending Sleep. Exiting Job!"
```

# Submit_sleep.sh

```
#!/bin/bash
#SBATCH --nodes=1                                  # Number of requested nodes
#SBATCH --ntasks=1                                 # Number of requested tasks
#SBATCH --time=0:01:00                             # Max wall time
#SBATCH --qos=testing                              # Specify QOS
#SBATCH --partition=shas-testing                   # Specify Summit Haswell nodes
#SBATCH --output=sleep_%j.out                      # Rename standard output file
#SBATCH --job-name=sleep                           # Job submission name
#SBATCH --reservation=job_submission_2019          # Reservation (workshop only)

# purge all existing modules
module purge

echo "I am" `whoami`
echo "Running on host" `hostname`
echo "Starting Sleep"
sleep 30
echo "Ending Sleep. Exiting Job!"
```

# Running an external program

- Let's run a Matlab program
- We will run the batch script submit_matlab.sh
- This script calls and runs matlab_tic.m

# Running the job script

- Submit the job:

```
sbatch submit_matlab.sh
```

- Check output

# submit_matlab.sh

```
#!/bin/bash
#SBATCH --nodes=1                          # Number of requested nodes
#SBATCH --ntasks=1                         # Number of requested tasks
#SBATCH --time=0:02:00                     # Max walltime
#SBATCH --qos=testing                      # Specify debug QOS
#SBATCH --partition=shas-testing           # Specify Summit haswell nodes
#SBATCH --output=matlab_%j.out             # Output file name
#SBATCH --reservation=job_submission_2019  # Reservation name

# purge all modules
module purge

# Load Matlab module
module load matlab

# Run matlab without a GUI
cd progs
matlab -nodisplay –nodesktop –r ”matlab_tic;”
```

# Your turn

- Submit a slurm job with the following instructions:

- Create an R program called R_program.R that creates a vector called "planets" and then list the planets in the vector. Syntax:
  planets -> planets <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune", "Pluto")

- Print off the vector. Syntax:
  planets

- Create a bash script called submit_R.sh that runs the R script (hint: Copy and modify the matlab job script you just ran)
- In the script, you'll run R with the following syntax:
  Rscript R_program.R

- Don't forget to load the R module before the Rscript command!

# Solution: R_program.R

```
# Simple R code example

# Create vector
planets <- c("Mercury", "Venus", "Earth",  "Saturn", "Uranus", "Mars",
"Jupiter", "Neptune", "Pluto")


# Print off vector
planets
```

# Solution: submit_R.sh

```
#!/bin/bash
#SBATCH --nodes=1                          # Number of requested nodes
#SBATCH --ntasks=1                         # Number of requested tasks
#SBATCH --time=0:02:00                     # Max walltime
#SBATCH --qos=testing                      # Specify debug QOS
#SBATCH --partition=shas-testing           # Specify Summit haswell nodes
#SBATCH --output=R_code_%j.out             # Output file name
#SBATCH --reservation=job_submission_2019  # Reservation name

# purge all existing modules
module purge

# Load the R module
module load R

# Run R script
Rscript progs/R_program.R
```

# Running an external program as an mpi job

- For cases where you have a code that is parallelized, meaning it can run across multiple cores.
- Number of tasks always > 1. E.g.,

```
--ntasks=4
```

- Will always need to load a compiler and mpi. E.g.,

```
module load intel impi
```

- Executable preceded with mpirun, srun, or mpiexec. E.g.,

```
mpirun -np 4 python yourscript.py
```

# submit_python_mpi.sh

```
#!/bin/bash
#SBATCH --nodes=1                                  # Number of requested nodes
#SBATCH --ntasks=4                                 # Number of requested nodes
#SBATCH --time=0:01:00                             # Max wall time
#SBATCH --qos=normal                               # Specify QOS
#SBATCH --partition=shas                           # Specify Summit haswell nodes
#SBATCH --output=hostname_%j.out                   # Rename standard output file
#SBATCH --reservation=job_submission_2019          # Reservation (workshop only)

# Purge all existing modules
module purge

# Load the modules you need
module load python/3.5.1 intel impi

# Run Python Script
cd progs
mpirun -np $SLURM_NTASKS python hello1.py
```

# Interactive jobs

- Sometimes we want our job to run in the background
- Sometimes we want to work on program in real time
    - Great for testing, debugging
- For example, let's run an interactive R job…

# Running an interactive job

- To work with R interactively, we request time from Summit
- When the resources become available the job starts
- Commands to run:

```
sinteractive --time=00:10:00 --reservation=job_submission_2019
```

- Once we receive a prompt, then:

```
module load R
Rscript R_program.R
```

- Once we finish we must exit! (job will time out eventually)

# CURC Load Balancer

- Useful when you have 100s to 100,000s of “tiny” serial jobs (<5 min)
- Instead of running them all as separate jobs, wrap into one job
- This will save you time and SUs, as it reduces start-up overhead
- To use (within batch job script): module load loadbalance
- Let’s take a look at the example in the documentation: https://curc.readthedocs.io/en/latest/software/loadbalancer.html

# Thank you!

- Please fill out the survey: http://tinyurl.com/curc-survey18
- Sign in! http://tinyurl.com/curc-names

- Contact information: rc-help@Colorado.edu

- Slides and Examples from this course: https://github.com/rctraining/HPC_Short_Course_Fall_2018
- Slurm Commands: https://slurm.schedmd.com/quickstart.html
- Load Balancer Tool: https://curc.readthedocs.io/en/latest/software/loadbalancer.html