Hewlett Packard Enterprise

Running Applications

Comprehensive General LUMI Course

April 23–26, 2024

Running on Compute Nodes

- The HPE Cray EX supercomputer uses a workload manager to manage and control access to compute nodes for user applications/jobs
- Users submit work to the scheduler (typically as a batch job) having specified the resources (compute, time, memory) needed for the work
- The scheduler chooses which resources to allocate to the job and runs the job
- The scheduler tracks resource usage and will kill jobs that exceed resource requirements
- The supported workload managers are
  - Altair PBS Professional
  - Slurm (used on LUMI)
- The CPE integrates with these workload managers so that parallel applications may be launched appropriately
- Installations are often configured by sites to meet their requirements

Viewing Available Partitions

Use the `sinfo` command

- A = Allocated
- I = Idle
- O = Other state (failed, drained…)
- T = Total

```
uan02:~> sinfo -s
PARTITION AVAIL  TIMELIMIT   NODES(A/I/O/T) NODELIST
standard     up 2-00:00:00 240/675/105/1020 nid[001002-001729,…
small        up 3-00:00:00    48/423/32/503 nid[002024-002527]
debug        up       30:00          0/7/1/8 nid[002528-002535]
lumid        up 1-00:00:00          0/0/8/8 nid[000016-000023]
largemem     up 1-00:00:00          0/0/8/8 nid[000101-000108]
eap          up 1-00:00:00       7/11/14/32 nid[005000-005003,…
pilot     inact 2-00:00:00 1183/1054/219/24 nid[005000-005119,…
standard-g   up 2-00:00:00 1139/944/125/220 nid[005032-007239]
small-g      up 3-00:00:00    69/149/62/280 nid[007240-007519]
dev-g        up    6:00:00       0/15/33/48 nid[007520-007531,…
q_fiqci      up      15:00          0/1/0/1 nid001000
q_nordiq     up      15:00          0/1/0/1 nid001001
```

Viewing available resources

```
uan02:~> sinfo -o "%10R  %10c  %10m  %25f  %20G  %20F"
PARTITION   CPUS        MEMORY      AVAIL_FEATURES            GRES                 NODES(A/I/O/T)
standard    256         229376      AMD_EPYC_7763             (null)               516/301/203/1020
small       256         229376+     AMD_EPYC_7763             (null)               27/414/62/503
debug       256         229376      AMD_EPYC_7763             (null)               0/7/1/8
lumid       256         2031616     AMD_EPYC_7742             gpu:a40:8,nvme:32K   0/0/8/8
largemem    256         4063232     AMD_EPYC_7742             (null)               0/0/8/8
standard-g  128         491520      AMD_EPYC_7A53,x1202       gpu:mi250:8          1139/944/125/2208
small-g     128         491520      AMD_EPYC_7A53,x1404       gpu:mi250:8          69/149/62/280
dev-g       128         491520      AMD_EPYC_7A53,x1405       gpu:mi250:8          0/15/33/48
q_fiqci     256         229376      AMD_EPYC_7763             (null)               0/1/0/1
q_nordiq    256         229376      AMD_EPYC_7763             (null)               0/0/1/1
```

- The 8 GCDs contained in the 4 MI250X will show as 8 separate GPUs according to Slurm, `HIP_VISIBLE_DEVICES`, `ROCR_VISIBLE_DEVICES`, and the ROCr runtime, so from this point forward in the quick-start guide, we will simply refer to the GCDs as GPUs

Showing Node Information

```
> sinfo -p small -N -l | head -6
Thu Feb 09 20:56:01 2023
NODELIST   NODES PARTITION       STATE CPUS    S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
nid002028      1     small   allocated 256    2:64:2 229376        0    200 AMD_EPYC none
nid002029      1     small   allocated 256    2:64:2 229376        0    200 AMD_EPYC none
nid002030      1     small   allocated 256    2:64:2 229376        0    200 AMD_EPYC none
nid002031      1     small   allocated 256    2:64:2 229376        0    200 AMD_EPYC none

> sinfo -p standard-g -N -l | head -6
Thu Feb 09 20:56:48 2023
NODELIST   NODES  PARTITION       STATE CPUS    S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
nid005032      1 standard-g        idle 128    1:64:2 491520        0      1 AMD_EPYC none
nid005033      1 standard-g        idle 128    1:64:2 491520        0      1 AMD_EPYC none
nid005034      1 standard-g        idle 128    1:64:2 491520        0      1 AMD_EPYC none
nid005035      1 standard-g        idle 128    1:64:2 491520        0      1 AMD_EPYC none
```

Requesting Resources in Slurm

- Users interact with Slurm by executing commands on the User Access Nodes (UANs)
- Slurm provides multiple mechanisms for users to access compute node resources

- Interactive use
  srun command
- Interactive use within a predefined allocation
  salloc to allocate resources
  srun command(s) to start an application
- Batch usage
  sbatch submits a batch script
  srun command(s) within the batch script

- Resource requests are made by
  - Structured comments in a batch job
  - Environment variables
  - Command-line arguments to sbatch, salloc or srun

Lifecycle of a Batch Script

User Access Node
`sbatch job.slurm`

Example Batch Job Script – `job.slurm`

```
#!/bin/bash
#SBATCH -p <your_partition>
#SBATCH -A <your_project>
#SBATCH --time=00:02:00
#SBATCH --nodes=2
#SBATCH --exclusive

cd <some_working_directory>
srun -n 256 ./simulation.exe
```

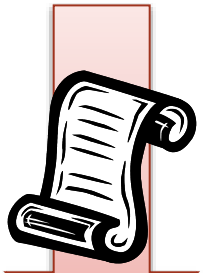
Scheduler Resources

Serial

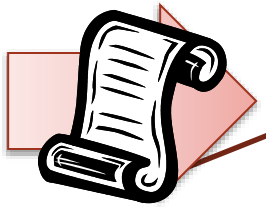
Parallel

Check your projects via the command `groups`, e.g.

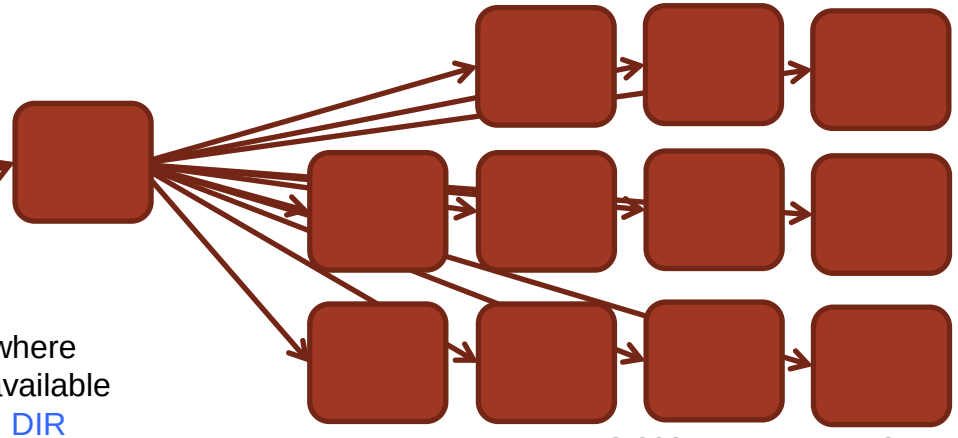
```
> groups
lazzaroa
project_462000003
project_462000031
```



SLURM



Compute Nodes



The script will start by default in the directory where `sbatch` has been executed. This directory is available in the environment variable SLURM_SUBMIT_DIR

Useful Resource-Related Options (srun/sbatch/salloc)

Description	Option
Total Number of tasks	-n,--ntasks
Number of tasks per compute node	--ntasks-per-node
Number of threads per task	-c,--cpus-per-task
Number of nodes	-N,--nodes
Walltime	-t,--time
Request *N* GPUs	--gres=gpu:*N*

Some Examples

Single node, Single task
Run a job on one task on one node with full memory.

```
…
#SBATCH -N 1

srun -n 1 ./<exe>
```

Single node
Run a pure MPI job with 128 Ranks on one node. The user can request a value for `-n` smaller than 128 but not larger.

```
…
#SBATCH -N 1

srun -n 128 ./<exe>
#srun -n 64 ./<exe>
#srun -n 32 ./<exe>
```

Multi node fully packed
Run a pure MPI job on 4 nodes with 128 MPI ranks on each node. The nodes are fully packed.

```
…
#SBATCH -N 4

srun -n 512 ./<exe>
```

Further Job Examples

Multi node partially filled
Run a pure MPI job on 4 nodes with less than 128 tasks per node.

```
#!<your_shell>
…
#SBATCH -N 4

srun --ntasks-per-node=32 -n 128 ./<exe>
#
srun --ntasks-per-node=16 -n 64 ./<exe>
```

Hybrid MPI/OpenMP
Run a hybrid application on 4 nodes with 32 tasks per node and 4 OpenMP threads per task using the `--cpus-per-task (-c)` parameter.

```
#!<your_shell>
…
#SBATCH -N 4

export OMP_NUM_THREADS=4
srun -n 128 -c 4 ./<exe>
```

SMT Threads

- Recall that some processors like the AMD Rome/Milan support Simultaneous Multiple Threading (SMT)
- These are enabled by default (check by running numactl –H on a compute node)
- To control scheduling tasks to these hardware threads you can use

  `--hint=multithread`

  `--hint=nomultithread` (default)
- With 128 cores per node and without SMT threads this will run on two nodes

  ```
  srun -n 256 ./exe
  ```
- With SMT threads this will run on one node

  ```
  srun --hint=multithread -n 256 ./exe
  ```

Example: Access and check available GPUs via ROCM-SMI

- Via batch job script

Example Batch Job Script – `job.slurm`

```
#!/bin/bash
#SBATCH -p <partition>
#SBATCH -A <your_project>
#SBATCH --time=00:02:00
#SBATCH --nodes=1
#SBATCH --gres=gpu:8
#SBATCH --exclusive

srun -n 1 rocm-smi
```

Check your projects via the command `groups`, e.g.

```
> groups
lazzaroa project_462000003 project_462000031
```

Request 8 GPUs

  - Submit via `sbatch job.slurm`

- Via interactive job

```
> srun -p <partition> -A <your_project> --time=00:02:00 --nodes=1 --gres=gpu:8
--exclusive -n 1 rocm-smi
```

ROCM-SMI Output

```
====================== ROCm System Management Interface ======================
================================ Concise Info ================================
GPU  Temp   AvgPwr  SCLK    MCLK     Fan  Perf  PwrCap  VRAM%  GPU%
0    40.0c  92.0W   800Mhz  1600Mhz  0%   auto  560.0W    0%   0%
1    44.0c  N/A     800Mhz  1600Mhz  0%   auto  0.0W      0%   0%
2    43.0c  84.0W   800Mhz  1600Mhz  0%   auto  560.0W    0%   0%
3    41.0c  N/A     800Mhz  1600Mhz  0%   auto  0.0W      0%   0%
4    44.0c  82.0W   800Mhz  1600Mhz  0%   auto  560.0W    0%   0%
5    40.0c  N/A     800Mhz  1600Mhz  0%   auto  0.0W      0%   0%
6    40.0c  83.0W   800Mhz  1600Mhz  0%   auto  560.0W    0%   0%
7    41.0c  N/A     800Mhz  1600Mhz  0%   auto  0.0W      0%   0%
==============================================================================
============================ End of ROCm SMI Log ============================
```

- Check `rocm-smi --help`
  - You can run it on the login node

Opening a shell on the compute node

- Sometimes it can be useful to open an interactive shell on a compute node
- Only serial execution, i.e. no MPI

```
uan02:~> srun -p <partition> -A <your_project> --time=00:02:00 --nodes=1 \
              --gres=gpu:8 --exclusive --pty bash -i
srun: job 1416181 queued and waiting for resources
srun: job 1416181 has been allocated resources
nid005201:~>
```

- Good practice to set a time limit (easy to forget you have allocated resources)
- It is possible to connect to a running job, this will be discussed in a later lecture

Showing Job Queue (squeue)

```
uan01> squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
 194902  standard       60     huey PD       0:00     16 (JobHeldUser)
 208875  standard  sII-122    dewey  R   18:38:01      1 nid001001
 209570  standard     AJHC    bruce  R    2:44:30      2 nid[001362-001363]
 208777  standard   Output    cliff  R   20:48:27      1 nid001316
 209744  standard long3_10   cheryl  R      28:31      1 nid001191
 208967  standard aiida-37    jesse  R   17:32:55      8 nid[001164,001167,001180-001185]
 209562  standard  Sc_c128    larry  R    2:49:13     16 nid[001324-001325,001367-001380]
 209773  standard u-cc629l   steven  R      13:44     36 nid[001331-001338,001352-001359,001382-00
 209045  standard NWChem_t     mark  R   15:55:09      2 nid[001116-001117]
 209787  standard    ktcnq      ron  R       8:35     20 nid[001441-001444,001464-001479]
```

Controlling Slurm command output

```
Uan01>  squeue -o "%.8i %.8u %.14j %.3t %19S %.10M %.10L %.5D %.4C"
   JOBID     USER           NAME  ST START_TIME                TIME  TIME_LEFT NODES CPUS
  209489     bill        cellopt   R 2021-04-18T11:01:12    3:51:35    8:08:25    12 3072
  209421      ben         O-Init   R 2021-04-18T10:16:05    4:36:42       23:18     6 1536
  209049   flower    NWChem_test   R 2021-04-17T22:43:23   16:09:24 1-07:50:36     2  512
  209048   flower    NWChem_test   R 2021-04-17T22:43:15   16:09:32 1-07:50:28     2  512
  209047   flower    NWChem_test   R 2021-04-17T22:43:06   16:09:41 1-07:50:19     2  512
  209046   flower    NWChem_test   R 2021-04-17T22:42:39   16:10:08 1-07:49:52     2  512
  209045   flower    NWChem_test   R 2021-04-17T22:42:33   16:10:14 1-07:49:46     2  512
  209044   flower    NWChem_test   R 2021-04-17T22:42:28   16:10:19 1-07:49:41     2  512
  209793    midge           ag_l   R 2021-04-18T14:30:19      22:28   23:37:32    20 5120
  209791    midge          ag_ts   R 2021-04-18T14:29:52      22:55   23:37:05    20 5120
```

Alternatively, you can set this with

```
export SQUEUE_FORMAT="%.8i %.8u %.14j %.3t %19S %.10M %.10L %.5D %.4C”
```

Useful Slurm commands

- sbatch
  submit job
- srun
  run a parallel job
- sinfo ， scontrol show partitions
  Shows information about Slurm nodes and partitions
- squeue, squeue -u <userid>, squeue --me
  View information about jobs
- scontrol show job <jobid>
  Show information about a running job
- scancel <jobid>
- sstat -j <jobid>
  Show status information for running job
- sacct -j <jobid>
  Show information about completed job

Some useful tips on Slurm

- Make sure you understand the system default for making a job exclusive, often sharing a node is not what you want…
  #SBATCH --exclusive
- Environment variables set within a job
  - SLURM_JOB_ID
  - SLURM_JOB_NAME
  - SLURM_PROCID
  - SLURM_JOB_NODELIST / SLURM_NODELIST
  - and many many more…
- Converting between node list formats
  scontrol show hostlist "… …“
  scontrol show hostnames "…[..-..]“ (output one per line but use Linux magic to combine)

```
nid[001128-001383,001768-001774,001776-001897,002409-002535,002664-002666,003945-004455,004584-004967,005224-005274,005276-005449,005451-005607,005736-005863,006120-006247]
                                                    (2048 nodes)
```

Slurm Job output and error streams

- By default, job output will appear in the file
  slurm-<jobid>.out
- You can direct the output to files using the following
  ```
  #SBATCH --output=<filename_pattern>
  #SBATCH --error=<filename_pattern>
  ```
  - Short form (-o,-e), if error is not specified then error is combined with standard output
- The pattern can have embedded elements, for example %x for job name and %j for jobid…
  ```
  #SBATCH –job-name=CP2K_fast
  #SBATCH –o %x_%j.out
  ```
- Note that the output files appear immediately although output may be buffered
- When running interactively
  srun –u
  runs unbuffered, this can be useful if a debug run crashes as you might not see errors
- To prepend the task number (followed by a :) to stdout/stderr
  srun –l (or --label)

A Note of Caution

- The examples given so far are simple but might not give good performance
- Slurm tends to distribute tasks to nodes in unpredictable ways
- It is essential to use the correct options to control the distribution of tasks, we will cover that later