Spock Tips & Information

Tom Papatheodore
Oak Ridge Leadership Computing Facility

Spock Training – May 20, 2021

ORNL is managed by UT-Battelle, LLC for the US Department of Energy
Where to Find Helpful Environment Variables

**man pages for** srun, salloc, sbatch

- Input and output environment variables can be found at bottom of each page
- E.g., SLURM_JOB_ID, SLURM_LOCALID, SLURM_JOB_NODELIST

**man intro_mpi**

- MPI environment variables
- E.g., MPICH_GPU_SUPPORT_ENABLED, MPICH_GPU_IPC_THRESHOLD

**module show <package_name>**

- Shows how loading a package will change your environment

```
$ module show boost
------------------------------------------------------------
/sw/spock/spack-envs/base/modules/spack/linux-sles15-x86_64/Core/boost/1.73.0.lua:
---------------------------------------------------------------------------
whatis("Name : boost")
whatis("Version : 1.73.0")
whatis("Target : x86_64")
whatis("Short description : Boost provides free peer-reviewed portable C++ source libraries, emphasizing libraries that work well with the C++ Standard Library.")
help(["Boost provides free peer-reviewed portable C++ source libraries, emphasizing libraries that work well with the C++ Standard Library.
Boost libraries are intended to be widely useful, and usable across a broad spectrum of applications. The Boost license encourages both commercial and non-commercial use."])
prepend_path("CMAKE_PREFIX_PATH","/sw/spock/spack-envs/base/opt/linux-sles15-x86_64/gcc-7.5.0/boost-1.73.0-nnaww46oatokeudktwfaxu6dv17zsyhw/")
setenv("BOOST_ROOT","/sw/spock/spack-envs/base/opt/linux-sles15-x86_64/gcc-7.5.0/boost-1.73.0-nnaww46oatokeudktwfaxu6dv17zsyhw")
setenv("OLCF_BOOST_ROOT","/sw/spock/spack-envs/base/opt/linux-sles15-x86_64/gcc-7.5.0/boost-1.73.0-nnaww46oatokeudktwfaxu6dv17zsyhw")
```
## Where to Find Helpful Environment Variables

<table>
<thead>
<tr>
<th>Command</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>man intro_</code></td>
<td><code>intro_asm_intrin</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_pmi</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_blaCs</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_pragmas</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_blas1</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_pxf</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_blas2</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_quad</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_blas3</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_quad_precision</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_cblas</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_scalapack</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_coarray</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_specialibs</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_conversion</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_timing</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_craypat</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_hugepages</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_upc</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_craype-api</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_ieee</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_lapack</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_openacc</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_papi</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_papi</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_papi</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_PAPI</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_perftools</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_openmp</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_libm</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_libsci</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_PAPI</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_perftools</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_openacc</code></td>
</tr>
<tr>
<td></td>
<td><code>intro_pgas</code></td>
</tr>
</tbody>
</table>
Helpful Tools

Users might find the following tools helpful...

**rocm-smi**
- AMD ROCm System Management Interface
- Find useful information about AMD GPUs (e.g., running processes, memory usage, temperature, driver version)
- `rocm-smi --help`

**rocminfo**
- ROCm Application for Reporting System Info
- Find useful system information (CPUs and GPUs)
- `rocminfo --help`

[https://code.ornl.gov/olcf/hello_mpi_omp](https://code.ornl.gov/olcf/hello_mpi_omp)
- MPI + OpenMP Hello World-type program for process and thread placement

[https://code.ornl.gov/olcf/hello_jobstep](https://code.ornl.gov/olcf/hello_jobstep)
- MPI + OpenMP + HIP Hello World-type program for process, thread, and GPU mapping
Debugging Issues on a Compute Node

You can `ssh` to compute nodes that are allocated to your job.

`squeue` (or other ways) will show the list of compute nodes for a job.

`ssh <spock_node_id>`

From there you can start troubleshooting. E.g., using `rocm-smi` to look at memory usage, processes on the GPUs, or `gstack <pid>` for CPU processes.
Custom Slurm Commands

You can customize the output of e.g., `squeue`, `sinfo`, `sacct` to your preference.

```bash
$ squeue -o "%-7A | %-10P | %-10u | %-10a | %-20j | %-11l | %-7M | %-3t | %-8Q | %-30R"

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>USER</th>
<th>ACCOUNT</th>
<th>NAME</th>
<th>TIME_LIMIT</th>
<th>TIME</th>
<th>NODES</th>
<th>ST</th>
<th>PRIORITY</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>261090</td>
<td>batch</td>
<td>tpapathe</td>
<td>stf016</td>
<td>interactive</td>
<td>1:00:00</td>
<td>27:42</td>
<td>1</td>
<td>R</td>
<td>2300</td>
<td>spock13</td>
</tr>
</tbody>
</table>

$ sacct --format=User,JobID,Jobname,partition,state,time,start,elapsed,nnodes,ncpus,nodeidlist,Priority -j 261090

<table>
<thead>
<tr>
<th>User</th>
<th>JobID</th>
<th>JobName</th>
<th>Partition</th>
<th>State</th>
<th>Timelimit</th>
<th>Start</th>
<th>Elapsed</th>
<th>NNodes</th>
<th>NCPUS</th>
<th>NodeList</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpapathe</td>
<td>261090</td>
<td>interacti+</td>
<td>batch</td>
<td>RUNNING</td>
<td>01:00:00</td>
<td>2021-05-19T17:25:27</td>
<td>00:35:02</td>
<td>1</td>
<td>128</td>
<td>spock13</td>
<td>20</td>
</tr>
<tr>
<td>261090.inte+</td>
<td>interacti+</td>
<td>RUNNING</td>
<td>2021-05-19T17:25:27</td>
<td>00:35:02</td>
<td>1</td>
<td>64</td>
<td>spock13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>261090.exte+</td>
<td>extern</td>
<td>RUNNING</td>
<td>2021-05-19T17:25:27</td>
<td>00:35:02</td>
<td>1</td>
<td>128</td>
<td>spock13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>261090.0</td>
<td>p2p</td>
<td>COMPLETED</td>
<td>2021-05-19T17:31:54</td>
<td>00:00:53</td>
<td>1</td>
<td>64</td>
<td>spock13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>261090.1</td>
<td>p2p</td>
<td>COMPLETED</td>
<td>2021-05-19T17:32:56</td>
<td>00:00:52</td>
<td>1</td>
<td>64</td>
<td>spock13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>261090.2</td>
<td>p2p</td>
<td>COMPLETED</td>
<td>2021-05-19T17:33:56</td>
<td>00:00:52</td>
<td>1</td>
<td>64</td>
<td>spock13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$ sinfo --format="%-10P | %-6a | %-7M | %-10T | %-9s | %-30N" -p caar

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>AVAIL</th>
<th>NODES</th>
<th>STATE</th>
<th>JOB_SIZE</th>
<th>NODELIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>caar</td>
<td>up</td>
<td>5</td>
<td>allocated</td>
<td>1-16</td>
<td>spock[01-02,13,25-26]</td>
</tr>
<tr>
<td>caar</td>
<td>up</td>
<td>31</td>
<td>idle</td>
<td>1-16</td>
<td>spock[03-12,14-24,27-36]</td>
</tr>
</tbody>
</table>
Capture Job Information

```bash
$ cat submit.sh
#!/bin/bash
#SBATCH -A STF016
#SBATCH -J test
#SBATCH -o %x-%j.out
#SBATCH -t 00:05:00
#SBATCH -p batch
#SBATCH -N 1

scontrol show job ${SLURM_JOBID}
srun -n1 ./p2p --correct
sacct -j ${SLURM_JOBID} -o jobid%20,Start%20,elapsed%20
```

Add other information as needed.
• e.g., `module -t list`

```bash
$ cat test-261226.out
JobId=261226 JobName=test
UserId=tpapathe(5987) GroupId=tpapathe(8654) MCS_label=N/A
Priority=594 Nice=0 Account=stf016 QoS=normal
JobState=RUNNING Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:00 TimeLimit=00:05:00 TimeMin=N/A
AccrueTime=2021-05-19T20:45:43
StartTime=2021-05-19T20:58:17 EndTime=2021-05-19T21:03:17 Deadline=N/A
SuspendTime=None SecsPreSuspend=0 LastSchedEval=2021-05-19T20:58:17
Partition=batch AllocNode:Sid=login1:24709
ReqNodeList=(null) ExcNodeList=(null)
NodeList=spock25
BatchHost=spock25
NumNodes=1 NumCPUs=128 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:1
TRES=cpu=128,node=1,billing=128
Socks/Node=* NtasksPerN:B:S:C=0:0:*:*
MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
Command=/autofs/nccs-svm1_home1/tpapathe/systems/spock/scripts/slurm_scripts/submit.sh
WorkDir=/autofs/nccs-svm1_home1/tpapathe/systems/spock/scripts/slurm_scripts
StdErr=/autofs/nccs-svm1_home1/tpapathe/systems/spock/scripts/slurm_scripts/test-261226.out
StdOut=/autofs/nccs-svm1_home1/tpapathe/systems/spock/scripts/slurm_scripts/test-261226.out
Power=*
NtasksPerTRES=0

...<APPLICATION OUTPUT>...

<table>
<thead>
<tr>
<th>JobID</th>
<th>Start</th>
<th>Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>261226</td>
<td>2021-05-19T20:58:17</td>
<td>00:00:42</td>
</tr>
<tr>
<td>261226.batch</td>
<td>2021-05-19T20:58:17</td>
<td>00:00:42</td>
</tr>
<tr>
<td>261226.extern</td>
<td>2021-05-19T20:58:17</td>
<td>00:00:42</td>
</tr>
<tr>
<td>261226.0</td>
<td>2021-05-19T20:58:17</td>
<td>00:00:42</td>
</tr>
</tbody>
</table>
```
Show Information About Completed Jobs

Specify a start (-S) and end (-E) time to search for completed jobs.

- The default time window depends on other options (see man sacct)

```bash
$ sacct --user=tpapathe -S 2021-05-18T23:00 -E 2021-05-18T23:45

<table>
<thead>
<tr>
<th>JobID</th>
<th>JobName</th>
<th>Partition</th>
<th>Account</th>
<th>AllocCPUS</th>
<th>State</th>
<th>ExitCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>259409</td>
<td>scontrol</td>
<td>batch</td>
<td>stf016</td>
<td>128</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>259409.exte+</td>
<td>extern</td>
<td></td>
<td>stf016</td>
<td>128</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>259409.0</td>
<td>scontrol</td>
<td></td>
<td>stf016</td>
<td>64</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>259426</td>
<td>test</td>
<td>batch</td>
<td>stf016</td>
<td>128</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>259426.batch</td>
<td>batch</td>
<td></td>
<td>stf016</td>
<td>64</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>259426.exte+</td>
<td>extern</td>
<td></td>
<td>stf016</td>
<td>128</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>259426.0</td>
<td>hostname</td>
<td></td>
<td>stf016</td>
<td>64</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
</tbody>
</table>
```
Cray and HIP Compiler Wrappers

$ CC -craype-verbose test.cpp
clang++ -march=znver2 -dynamic -D__CRAY_X86_ROME -D__CRAYXT_COMPUTE_LINUX_TARGET --gcc-toolchain=/opt/gcc/8.1.0/snos -isystem /opt/cray/pe/cce/11.0.4/cce-clang/x86_64/lib/clang/11.0.0/include -isystem /opt/cray/pe/11.0.4/cce/5.0.0/include/craylibs -Wl,-rpath=/opt/cray/pe/gcc-libs tests.cpp -I/opt/cray/pe/libsci/21.04.1.1/CRAY/9.0/x86_64/include -I/opt/cray/pe/mpich/8.1.4/ofi/cray/9.1/include -I/opt/cray/pe/ml6.0.10/include -I/opt/cray/pe/dsmml/0.1.4/dsmml/include -I/opt/cray/pe/xpmem/2.2.40-2.1.2.7_g3cfs325.shasta/include -L/opt/cray/pe/libsci/21.04.1.1/CRAY/9.0/x86_64/lib -L/opt/cray/pe/ml6.0.10/lib -L/opt/cray/pe/dsmml/0.1.4/dsmml/lib -L/opt/cray/pe/xpmem/2.2.40-2.1.2.7_g3cfs325.shasta/lib -L/opt/cray/pe/mpich/8.1.4/ofi/cray/9.1/lib -L/opt/cray/pe/ml6.0.10/lib -L/opt/cray/pe/dsmml/0.1.4/dsmml/lib -L/opt/cray/pe/xpmem/2.2.40-2.1.2.7_g3cfs325.shasta/lib

remove need to link certain libraries & headers, but sometimes it’s helpful to see what’s actually being invoked.

sometimes it’s helpful to see what’s actually being invoked.

$ export HIPCC_VERBOSE=3
$ hipcc test.cpp
HIPCC_VERBOSE=<value>, where value can be 1-7
Linking in HSA Library


Users might need to manually link in libhsa-runtime64.so in some codes.

• E.g., When setting PE_MPICH_GTL_DIR_amdgfx908 and PE_MPICH_GTL_LIBS_amdgfx908 to use GPU-Aware MPI.
• If needed, you can link in this library using: $ROCM_PATH/lib -lhsa-runtime64

You can learn more about HSA here:

Keeping Up with the Clangs

There are multiple versions of Clang compilers on Spock. If you’re uncertain of which version is being used in your environment, you can issue `which clang++` to find out.

```bash
$ module -t list
cce/11.0.4
craype/2.7.6
craype-x86-rome
libfabric/1.11.0.3.74
craype-network-ofi
cray-dsmml/0.1.4
perftools-base/21.02.0
xpmem/2.2.40-2.1.2.7__g3cf3325.shasta
cray-mpich/8.1.4
cray-libsci/21.04.1.1
cray-pmi/6.0.10
cray-pmi-lib/6.0.10
DefApps/default
PrgEnv-cray/8.0.0
rocm/4.1.0
$ which clang++
/sw/spock/spack-envs/views/rocm-4.1.0/llvm/bin/clang++
$ module unload rocm
$ which clang++
/opt/cray/pe/cce/11.0.4/cce-clang/x86_64/bin/clang++
```

NOTE: It is highly recommended to use the Cray compiler wrappers (`cc`, `cc`, and `ftn`) whenever possible. See the next section for more details.

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Programming Environment</th>
<th>Compiler Module</th>
<th>Language</th>
<th>Compiler Wrapper</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray</td>
<td>PrgEnv-cray</td>
<td>cce</td>
<td>C</td>
<td>cc</td>
<td>craycc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C++</td>
<td>cc</td>
<td>craycxx  or crayCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fortran</td>
<td>ftn</td>
<td>crayftn</td>
</tr>
<tr>
<td>AMD</td>
<td>Not yet available</td>
<td>rocm</td>
<td>C</td>
<td>Not yet available</td>
<td>$ROCM_PATH/llvm/bin/clang</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C++</td>
<td>Not yet available</td>
<td>$ROCM_PATH/llvm/bin/clang++</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fortran</td>
<td>Not yet available</td>
<td>$ROCM_PATH/llvm/bin/clangftn</td>
</tr>
</tbody>
</table>

- For ROCm Clang compilers, give full path
- For Cray Clang compilers, use compiler wrappers or `craycc`, `crayCC`, `crayftn`
Generate Detailed Node Image

From within an interactive job (or via batch script)...

```
$ salloc -A stf016 -t 30 -N 1
salloc: Pending job allocation 258383
salloc: job 258383 queued and waiting for resources
salloc: job 258383 has been allocated resources
salloc: Granted job allocation 258383
salloc: Waiting for resource configuration
salloc: Nodes spock27 are ready for job
```

```
$ module load hwloc
$ srun -n1 lstopo spock_compute_node.svg
```

`scp` to your laptop/desktop and open in browser or convert to png using e.g., inkscape

I don’t expect you to be able to read this in the slide : )
Helpful Resources

Spock Quick-Start Guide: [https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html](https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html)
• Make sure to look at the Known Issues section if you encounter problems.

Spock Mailing List: All Spock users are included on a mailing list where OLCF will communicate about downtimes, software upgrades, etc.


ROCm GitHub Pages:
• [https://github.com/ROCmSoftwarePlatform](https://github.com/ROCmSoftwarePlatform)
• [https://github.com/ROCm-Developer-Tools](https://github.com/ROCm-Developer-Tools)

Getting Help: If you have problems or need help running on Spock, please submit a support ticket to [help@olcf.ornl.gov](mailto:help@olcf.ornl.gov).
Questions?

Summit here

Frontier here

OAK RIDGE
National Laboratory
Backup Slides
Find CPUs available to each MPI rank

Prepend task number to lines of stdout/err.

This is what is being launched on the compute node

I want my 8 MPI ranks to be placed 2-per-NUMA, where ranks 0-1 are on NUMA 0, ranks 2-3 are on NUMA 1, etc.

How do I know which CPU cores are available to each MPI rank?

From within an interactive job...

```
$ srun -l -n8 -c2 /bin/bash -c 'grep Cpus_allowed_list /proc/self/status' | sort
0: Cpus_allowed_list: 0-1
1: Cpus_allowed_list: 16-17
2: Cpus_allowed_list: 32-33
3: Cpus_allowed_list: 48-49
4: Cpus_allowed_list: 2-3
5: Cpus_allowed_list: 18-19
6: Cpus_allowed_list: 34-35
7: Cpus_allowed_list: 50-51
```

```
$ srun -l -n8 -c8 -m *:block /bin/bash -c 'grep Cpus_allowed_list /proc/self/status' | sort
0: Cpus_allowed_list: 0-7
1: Cpus_allowed_list: 8-15
2: Cpus_allowed_list: 16-23
3: Cpus_allowed_list: 24-31
4: Cpus_allowed_list: 32-39
5: Cpus_allowed_list: 40-47
6: Cpus_allowed_list: 48-55
7: Cpus_allowed_list: 56-63
```

```
$ srun -l -n8 -c2 -m *:block /bin/bash -c 'grep Cpus_allowed_list /proc/self/status' | sort
0: Cpus_allowed_list: 0-1
1: Cpus_allowed_list: 2-3
2: Cpus_allowed_list: 4-5
3: Cpus_allowed_list: 6-7
4: Cpus_allowed_list: 8-9
5: Cpus_allowed_list: 10-11
6: Cpus_allowed_list: 12-13
7: Cpus_allowed_list: 14-15
```

```
$ srun -l -n8 -c2 /bin/bash -c 'grep Cpus_allowed_list /proc/self/status' | sort
0: Cpus_allowed_list: 0-1
1: Cpus_allowed_list: 16-17
2: Cpus_allowed_list: 32-33
3: Cpus_allowed_list: 48-49
4: Cpus_allowed_list: 2-3
5: Cpus_allowed_list: 18-19
6: Cpus_allowed_list: 34-35
7: Cpus_allowed_list: 50-51
```

```
$ srun -l -n8 -c2 /bin/bash -c 'grep Cpus_allowed_list /proc/self/status' | sort
0: Cpus_allowed_list: 0-1
1: Cpus_allowed_list: 16-17
2: Cpus_allowed_list: 32-33
3: Cpus_allowed_list: 48-49
4: Cpus_allowed_list: 2-3
5: Cpus_allowed_list: 18-19
6: Cpus_allowed_list: 34-35
7: Cpus_allowed_list: 50-51
```

```
$ srun -l -n8 -c2 /bin/bash -c 'grep Cpus_allowed_list /proc/self/status' | sort
0: Cpus_allowed_list: 0-1
1: Cpus_allowed_list: 16-17
2: Cpus_allowed_list: 32-33
3: Cpus_allowed_list: 48-49
4: Cpus_allowed_list: 2-3
5: Cpus_allowed_list: 18-19
6: Cpus_allowed_list: 34-35
7: Cpus_allowed_list: 50-51
```
## Slurm partitions

Spock’s compute nodes are separated into 2 Slurm partitions (queues): 1 for CAAR projects and 1 for ECP.

### CAAR Partition

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Max Walltime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 4</td>
<td>3 hours</td>
</tr>
<tr>
<td>5 – 16</td>
<td>1 hour</td>
</tr>
</tbody>
</table>

- 24 total compute nodes
- Each user can have
  - 1 running and 1 eligible job at a time (per project)
  - No limit on the number of jobs submitted.

### ECP Partition

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Max Walltime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 4</td>
<td>3 hours</td>
</tr>
</tbody>
</table>

- 12 total compute nodes
- Each user can have
  - 1 running and 1 eligible job at a time (per project)
  - Up to 5 jobs submitted

---

If CAAR or ECP teams require a temporary exception to this policy, please email [help@olcf.ornl.gov](mailto:help@olcf.ornl.gov) with your request and it will be given to the OLCF Resource Utilization Council (RUC) for review.
GPU-Aware MPI

When using GPU-Aware MPI with the Cray compiler wrappers, there are currently some extra steps needed...

The following modules must be loaded:

```
module load PrgEnv-cray
module load craype-accel-amd-gfx908
module load rocm
```

The following environment variables must be set before compiling so the executable picks up GTL

```
export PE_MPICH_GTL_DIR_amd_gfx908="-L/opt/cray/pe/mpich/8.1.4/gtl/lib"
export PE_MPICH_GTL_LIBS_amd_gfx908="-lmpi_gtl_hsa"
```

The following environment variables must be set before running

```
export MIR_CVAR_GPU_EAGER_DEVICE_MEM=0
export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_SMP_SINGLE_COPY_MODE=CMA
```

In addition, the following header files and libraries must be included:

```
-I${ROCM_PATH}/include
-L${ROCM_PATH}/lib -lamdhip64 -lhsa-runtime64
```

If you set this in a script, it will need to be removed once a known issue is resolved.