Summit Tips & Tricks

Tom Papatheodore
Oak Ridge Leadership Computing Facility (OLCF)
Summit New User Training

June 3, 2020
Basic Batch Script Example

#!/bin/bash
# BEGIN LSF DIRECTIVES
#BSUB -P <PROJECT_ID>
#BSUB -J <JOB_NAME>
#BSUB -o <JOB_NAME>.o%J
#BSUB -e <JOB_NAME>.e%J
#BSUB -W <WALLTIME_HH:MM>
#BSUB -nnodes <NUMBER_OF_NODES>
# END LSF DIRECTIVES

number_of_nodes=$(cat $LSB_DJOB_HOSTFILE | uniq | head -n 1 | wc -l)

# JSRUN OPTIONS (CONFIGURE ME!)
number_of_resource_sets=12
resource_sets_per_node=6
physical_cores_per_resource_set=7
gpus_per_resource_set=1
mpi_ranks_per_resource_set=1
physical_cores_per_mpi_rank=7
export OMP_NUM_THREADS=1

jsrun -n ${number_of_resource_sets} -r ${resource_sets_per_node} -c ${physical_cores_per_resource_set} -g ${gpus_per_resource_set} -a ${mpi_ranks_per_resource_set} -bpacked:${{physical_cores_per_resource_set}} js_task_info |& sort -k2 -n

Do not copy and paste this from here! There can be unintended formatting issues.

Instead, get it from here:
https://github.com/olcf-tutorials/Batch-Script-Examples
Thread-Safe XL Compiler Variants

When using OpenMP with IBM’s XL compilers, the thread-safe variants are required. These variants have the same name as the non-thread-safe variants but with an additional \_r appended.

So, for example, to compile a C program with OpenMP, you would need to use xlc\_r instead of xlc.
MPI Compiler Wrappers

The MPI compiler wrappers (e.g., `mpicc`) conveniently remove the need to link in Spectrum MPI, but there are times when it's helpful to see what is actually being invoked.

```bash
$ module -t list
xl/16.1.1-5
spectrum-mpi/10.3.1.2-20200121
hsi/5.0.2.p5
xalt/1.2.0
lsf-tools/2.0
darshan-runtime/3.1.7
DefApps

$ mpicc --showme
```
Using Libraries

OLCF provides many software packages and scientific libraries via environment modules. To use them in an application, you must direct the compiler to their location.

```bash
$ module show essl
---------------------------------------------------------------------
/sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core/essl/6.1.0-2:
---------------------------------------------------------------------
whatis("ESSL 6.1.0-2 ")
prepend_path("LD_LIBRARY_PATH","/sw/summit/essl/6.1.0-2/essl/6.1/lib64")
append_path("LD_LIBRARY_PATH","/sw/summit/xl/16.1.1-1/lib")
prepend_path("MANPATH","/sw/summit/essl/6.1.0-2/essl/6.1/man")
setenv("OLCF_ESSL_ROOT","/sw/summit/essl/6.1.0-2")
help([[ESSL 6.1.0-2
]])
```

Use `module show` to find the path to a package/library

Once the module is loaded, the `OLCF_PACKAGE_NAME_ROOT` environment variable holds the path to that package/library installation, so add include and/or library paths using `-I$OLCF_PACKAGE_NAME_ROOT/include` and `-L$OLCF_PACKAGE_NAME_ROOT/lib -llibname`
CUDA Multi-Process Service (MPS)

Allows multiple processes (e.g., MPI ranks) to concurrently share the resources on a single GPU.

To enable, use LSF flag: -alloc_flags “gpumps”

```bash
$ jsrun -n1 -c4 -g1 -a4 -bpacked:1 ./vector_add
CUDA Error - vector_addition.cu:36: 'all CUDA-capable devices are busy or unavailable'
CUDA Error - vector_addition.cu:36: 'all CUDA-capable devices are busy or unavailable'
CUDA Error - vector_addition.cu:36: 'all CUDA-capable devices are busy or unavailable'

---------------------------
__SUCCESS__
---------------------------
N       = 1048576
Threads Per Block = 256
Blocks In Grid    = 4096
```

Without MPS, you might see errors like these:

For more information, see the OLCF user documentation or NVIDIA's more comprehensive document:

https://docs.olcf.ornl.gov/systems/summit_user_guide.html#volta-multi-process-service
CUDA-Aware MPI

Allows GPU buffers (e.g., memory allocated with `cudaMalloc`) to be used directly in MPI calls, rather than needing to manually transfer data to/from a CPU buffer (e.g., using `cudaMemcpy`) before/after passing data in MPI calls.

To enable, use `jsrun` flag: `--smpiargs="-gpu"

**COMMON POINT OF CONFUSION:**

CUDA-Aware MPI is separate from GPUDirect!

By itself, CUDA-Aware MPI does not specify whether data is staged through CPU memory or transferred directly between GPUs.

GPUDirect is a technology that can be implemented on a system to enhance CUDA-Aware MPI by allowing data transfers directly between GPUs on the same node (peer-to-peer) and/or directly between GPUs on different nodes (with RDMA support) without the need to stage data through CPU memory. Both are implemented on Summit but must be enabled.

A simple tutorial on using CUDA-Aware MPI: [https://github.com/olcf-tutorials/MPI_ping_pong](https://github.com/olcf-tutorials/MPI_ping_pong)
Per-User Login Node Resource Limits

The login nodes are shared by all Summit users, so cgroups are used to ensure availability of resources (CPU cores, GPUs, and memory) among users.

The limits are summarized as follows:

- Each user is limited to **16 hardware threads, 16 GB of memory, and 1 GPU**

- If a process from any of a user’s login sessions reaches
  - 4 hours of CPU-time, all login sessions will be limited to 0.5 hardware threads
  - 8 hours of CPU-time, the process is automatically killed

- To reset the cgroup limits on a login node once the 4- or 8-hour CPU-time reduction has been reached, kill the offending process and start a new login session to the node.

**NOTE:** Login node limits are set per user and not per individual login session. All user processes on a node are contained within a single cgroup and will share the cgroup’s limits.
Inspecting Backfill on Summit

**bjobs** and **jobstat** help identify what’s currently running and scheduled to run, but sometimes it’s helpful to know how much of the system is *not* currently in use (or scheduled for use).

**bslots** can be used to inspect backfill windows and answer the question “How many nodes are currently available, and for how long will they remain available?”

By requesting resources within a backfill window, you can potentially shorten your queued time (and improve overall system utilization).

- Summit compute nodes have 1 “slot” per physical CPU core, for a total of 42 per node ([2x] Power9 CPUs, each with 21 cores). So the output from **bslots** can be divided by 42 to see how many nodes are currently available.
- By default, **bslots** includes launch node slots, which can cause incorrect compute node values. The output can be adjusted to reflect only available compute node slots with the flag `-R"select[CN]"`.

(3360 slots) / (42 slots / node) = 80 nodes

---

https://docs.olcf.ornl.gov/systems/summit_user_guide.html#inspecting-backfill
### jslist

Displays the running, pending, and completed job steps

...for completed steps...

Show Resource Sets...

... but just the most recent step.

```
$ jslist -R -d --last 1

<table>
<thead>
<tr>
<th>ID</th>
<th>ID</th>
<th>nrs</th>
<th>per RS</th>
<th>cpus</th>
<th>per RS</th>
<th>gpus</th>
<th>per RS</th>
<th>exit</th>
<th>status</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>21</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Complete</td>
<td></td>
</tr>
</tbody>
</table>

RS 0 HOST h27n04:
  SOCKET 0:    cpus: 0-20 gpus: 0 1 2 mem: 1000
RS 1 HOST h27n04:
  SOCKET 1:    cpus: 21-41 gpus: 3 4 5 mem: 1000
```
**nvcc Tips**

By default, `nvcc` uses `g++` as the host compiler (regardless of which compiler module you have loaded). To change which host compiler is used, you can use the `-ccbin` flag

```bash
$ nvcc -ccbin=xlc++_r ...
```

To pass a flag directly to the host compiler, you can use the `-Xcompiler` flag

```bash
$ nvcc -ccbin=xlc++_r -Xcompiler -qsmp=omp ...
```

Passing `-x cu` causes all `.cpp` files to be treated as `.cu` (i.e., passes them through CUDA toolchain)

```bash
$ nvcc -x cu example.cpp
```

You can also use CUDA runtime without compiling with `nvcc` by including

```bash
-I$OLCF_CUDA_ROOT/include -L$OLCF_CUDA_ROOT/lib64 -lcudart
```
OLCF Training

The OLCF provides training to our users in a variety of ways.

**OLCF Training Calendar:** [https://www.olcf.ornl.gov/for-users/training/training-calendar](https://www.olcf.ornl.gov/for-users/training/training-calendar)
- Find upcoming (and past) OLCF training events

**OLCF Tutorials:** [https://github.com/olcf-tutorials](https://github.com/olcf-tutorials)
- Self-guided tutorials on selected topics related to OLCF systems

**OLCF GPU Hackathons:** [https://gpuhackathons.org/events](https://gpuhackathons.org/events)
- Multi-day coding events, where teams of developers work on their own applications alongside mentors with GPU programming expertise.

**OLCF Training Archive:** [https://docs.olcf.ornl.gov/training/training_archive.html#](https://docs.olcf.ornl.gov/training/training_archive.html#)
- Slides/recordings from previous OLCF training events
Questions?

Summit in Annex Bldg

Frontier here