Summit Scheduler and Job Launch

Introduction

OLCF New User Training

Chris Fuson
June 03, 2020
SSH → bsub → jsrun

- **Login Nodes (Shared)**
- **Batch Nodes (Shared)**
- **Compute Nodes (Exclusive)**
SSH → Login Nodes (Shared) → bsub → Batch Nodes (Shared) → jsrun → Compute Nodes (Exclusive)
Summit Parallel Job Execution

**Batch System (LSF)**
- Batch scheduler, allocates resources
- Similar functionality to PBS/MOAB/SLURM
- Allocates entire nodes

**Job Launcher (jsrun)**
- Developed by IBM for the Oak Ridge and Livermore CORAL systems
- Similar functionality to aprun/srun/mpirun
LSF Example Batch Script

Batch script example

```
#!/bin/bash

#BSUB -W 2:00
#BSUB -n nodes 2
#BSUB -P abc007
#BSUB -o example.o%J
#BSUB -J example

hostname
jsrun -n2 -r1 -a1 -c1 hostname
```

Batch submission

```
summit-login1> bsub example.lsf
Job <29209> is submitted to default queue <batch>.
summit-login1>
```
## Common bsub Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Example Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-W</td>
<td>#BSUB –W 1:00</td>
<td>Requested Walltime [hours:]minutes</td>
</tr>
<tr>
<td>-nnodes</td>
<td>#BSUB –nnodes 1024</td>
<td>Number of nodes (CORAL systems)</td>
</tr>
<tr>
<td>-P</td>
<td>#BSUB –P ABC123</td>
<td>Project to which the job should be charged</td>
</tr>
<tr>
<td>-J</td>
<td>#BSUB –J MyJobName</td>
<td>Name of the job. If not specified, will be set to ‘Not_Specified’.</td>
</tr>
<tr>
<td>-o</td>
<td>#BSUB –o jobout.%J</td>
<td>File into which job STDOUT should be directed (%J will be replaced with the job ID number)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If not specified will be set to ‘JobName.%J’</td>
</tr>
<tr>
<td>-e</td>
<td>#BSUB –e joberr.%J</td>
<td>File into which job STDERR should be directed</td>
</tr>
<tr>
<td>-w</td>
<td>#BSUB –w ended(1234)</td>
<td>Place dependency on previously submitted jobID 1234</td>
</tr>
<tr>
<td>-N</td>
<td>#BSUB –N</td>
<td>Send job report via email once job completes (N) or begins (B)</td>
</tr>
<tr>
<td>-B</td>
<td>#BSUB -B</td>
<td></td>
</tr>
<tr>
<td>-alloc_flags</td>
<td>#BSUB –alloc_flags gpumps</td>
<td>Used to request GPU Multi-Process Service (MPS) and to set SMT (Simultaneous Multithreading) levels.</td>
</tr>
<tr>
<td></td>
<td>#BSUB –alloc_flags smt1</td>
<td></td>
</tr>
</tbody>
</table>

*More details and flags can be found in the bsub manpage*
LSF Interactive Batch Job

- Allows access to compute resources interactively
- Through batch system similar to batch script submission, but returns prompt on launch node
- Run multiple jsrun with only one queue wait, very useful for testing and debugging

**Syntax**
- Use `-Is` and the shell to be started
- Most other batch flags valid
- Add batch flags to command line

```bash
summit-login1> bsub -ls -P abc007 -nnodes 2 -W 2:00 $SHELL
Job <29507> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on batch1>>
summit-batch1 307> hostname
batch1
summit-batch1 308> jsrun -n2 -r1 hostname
a01n01
a01n02
summit-batch1 309>
```

Presentation examples use the following to allocate resources
Common LSF Commands

<table>
<thead>
<tr>
<th>Function</th>
<th>PBS/MOAB</th>
<th>SLURM</th>
<th>LSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit</td>
<td>qsub</td>
<td>sbatch/salloc</td>
<td>bsub</td>
</tr>
<tr>
<td>Monitor Queue</td>
<td>showq/qstat</td>
<td>squeue</td>
<td>bjobs/jobstat</td>
</tr>
<tr>
<td>Investigate Job</td>
<td>checkjob</td>
<td>scontrol/sacct</td>
<td>bhist</td>
</tr>
<tr>
<td>Alter Queued Job</td>
<td>qalter</td>
<td>scontrol update</td>
<td>bmod</td>
</tr>
<tr>
<td>Remove Queued Job</td>
<td>qdel</td>
<td>scancel</td>
<td>bkill</td>
</tr>
<tr>
<td>Hold Queued Job</td>
<td>qhold</td>
<td>scontrol hold</td>
<td>bstop</td>
</tr>
<tr>
<td>Release Held Job</td>
<td>qrls</td>
<td>scontrol release</td>
<td>bresume</td>
</tr>
</tbody>
</table>
Viewing the Batch Queue with bjobs

• ‘bjobs’
  – Will display *only your jobs by default* if no options given

• ‘bjobs -u all’
  – Will show all queued jobs

• ‘bjobs –l jobID’
  – Will show details of given jobID

• As with MOAB, jobs can be organized into three high level categories
  – 1) Running  2) Pending Eligible  3) Pending Ineligible

• ‘bjobs –uall –pei’
  – Will show pending jobs separated into eligible and ineligible
Viewing the Batch Queue with `jobstat`

- OLCF developed tool to help view queue

### Running Jobs

<table>
<thead>
<tr>
<th>JobId</th>
<th>Username</th>
<th>Project</th>
<th>Nodes</th>
<th>Remain</th>
<th>StartTime</th>
<th>JobName</th>
</tr>
</thead>
<tbody>
<tr>
<td>221070</td>
<td>userA</td>
<td>CSC100</td>
<td>512</td>
<td>44:22</td>
<td>11/30 11:01:25</td>
<td>run745-A3</td>
</tr>
<tr>
<td>221090</td>
<td>userA</td>
<td>CSC100</td>
<td>272</td>
<td>1:35:12</td>
<td>11/30 11:52:15</td>
<td>run745-B2</td>
</tr>
<tr>
<td>221092</td>
<td>userB</td>
<td>CSC006</td>
<td>1</td>
<td>1:06:47</td>
<td>11/30 11:23:50</td>
<td></td>
</tr>
<tr>
<td>221105</td>
<td>userC</td>
<td>CSC007</td>
<td>3200</td>
<td>2:59:40</td>
<td>11/30 12:16:43</td>
<td>Not_Specified</td>
</tr>
<tr>
<td>221095</td>
<td>userD</td>
<td>CSC201</td>
<td>2</td>
<td>1:29:29</td>
<td>11/30 11:46:32</td>
<td>Not_Specified</td>
</tr>
<tr>
<td>221088</td>
<td>userE</td>
<td>CSC100</td>
<td>170</td>
<td>1:31:06</td>
<td>11/30 11:48:09</td>
<td>20_a_1</td>
</tr>
<tr>
<td>221097</td>
<td>userF</td>
<td>CSC100</td>
<td>1</td>
<td>1:52:26</td>
<td>11/30 12:09:29</td>
<td>Job3</td>
</tr>
</tbody>
</table>

### Eligible Jobs

<table>
<thead>
<tr>
<th>JobId</th>
<th>Username</th>
<th>Project</th>
<th>Nodes</th>
<th>Walltime</th>
<th>QueueTime</th>
<th>Priority</th>
<th>JobName</th>
</tr>
</thead>
<tbody>
<tr>
<td>221108</td>
<td>userC</td>
<td>CSC007</td>
<td>4200</td>
<td>10:00:00</td>
<td>11/30 12:16:07</td>
<td>520.00</td>
<td>Not_Specified</td>
</tr>
<tr>
<td>221101</td>
<td>userC</td>
<td>CSC007</td>
<td>1048</td>
<td>6:00:00</td>
<td>11/30 12:12:28</td>
<td>515.00</td>
<td>Not_Specified</td>
</tr>
</tbody>
</table>

### Blocked Jobs

<table>
<thead>
<tr>
<th>JobId</th>
<th>Username</th>
<th>Project</th>
<th>Nodes</th>
<th>Walltime</th>
<th>BlockReason</th>
</tr>
</thead>
<tbody>
<tr>
<td>221099</td>
<td>userA</td>
<td>CSC100</td>
<td>1048</td>
<td>6:00:00</td>
<td>JOB limit defined for the user or user group has been reached</td>
</tr>
<tr>
<td>221110</td>
<td>userC</td>
<td>CSC007</td>
<td>1800</td>
<td>8:00:00</td>
<td>JOB limit defined for the user or user group has been reached</td>
</tr>
<tr>
<td>221107</td>
<td>userC</td>
<td>CSC007</td>
<td>1</td>
<td>45:00</td>
<td>JOB defined for the user or user group has been reached</td>
</tr>
<tr>
<td>221151</td>
<td>userC</td>
<td>CSC007</td>
<td>16</td>
<td>3:00:00</td>
<td>JOB defined for the user or user group has been reached</td>
</tr>
</tbody>
</table>
# Summit Queue Policy

<table>
<thead>
<tr>
<th>Bin</th>
<th>Min Nodes</th>
<th>Max Nodes</th>
<th>Max Walltime (hrs)</th>
<th>Aging Boost (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,765</td>
<td>4,608</td>
<td>24</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>922</td>
<td>2,764</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>92</td>
<td>921</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
<td>91</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>45</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

- Eligible to run limit: 4
- [https://docs.olcf.ornl.gov/systems/summit_user_guide.html#scheduling-policy](https://docs.olcf.ornl.gov/systems/summit_user_guide.html#scheduling-policy)
### Summit Node

<table>
<thead>
<tr>
<th>Socket 0</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory (256GB)</td>
<td>000</td>
<td>028</td>
<td>029</td>
</tr>
<tr>
<td></td>
<td>002</td>
<td>030</td>
<td>031</td>
</tr>
<tr>
<td></td>
<td>004</td>
<td>032</td>
<td>033</td>
</tr>
<tr>
<td></td>
<td>006</td>
<td>034</td>
<td>035</td>
</tr>
<tr>
<td></td>
<td>008</td>
<td>036</td>
<td>037</td>
</tr>
<tr>
<td></td>
<td>010</td>
<td>038</td>
<td>039</td>
</tr>
<tr>
<td></td>
<td>012</td>
<td>040</td>
<td>041</td>
</tr>
<tr>
<td></td>
<td>014</td>
<td>042</td>
<td>043</td>
</tr>
<tr>
<td></td>
<td>016</td>
<td>044</td>
<td>045</td>
</tr>
<tr>
<td></td>
<td>018</td>
<td>046</td>
<td>047</td>
</tr>
<tr>
<td></td>
<td>020</td>
<td>048</td>
<td>049</td>
</tr>
<tr>
<td></td>
<td>022</td>
<td>050</td>
<td>051</td>
</tr>
<tr>
<td></td>
<td>024</td>
<td>052</td>
<td>053</td>
</tr>
<tr>
<td></td>
<td>026</td>
<td>054</td>
<td>055</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Socket 1</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory (256GB)</td>
<td>088</td>
<td>089</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>090</td>
<td>091</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>092</td>
<td>093</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>094</td>
<td>095</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>096</td>
<td>097</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>098</td>
<td>099</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>101</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>103</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>105</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>106</td>
<td>107</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>108</td>
<td>109</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>111</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td>112</td>
<td>113</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>114</td>
<td>115</td>
<td>142</td>
</tr>
</tbody>
</table>

- **Socket (2 x Node)**
- **Core (21 x Socket)**
- **GPU (3 x Socket)**
- **Hardware Thread (4x Core)**
- **RAM**

*Numbering skips due to core isolation*
Hardware Thread Levels

- Each physical core contains 4 hardware threads
- Simultaneous Multithreading (SMT)
- Power9 supports 3 levels: 1, 2, or 4 virtual cores
- SMT level set for each batch job
  - #BSUB – alloc_flags smt1
  - #BSUB – alloc_flags smt2
  - #BSUB – alloc_flags smt4 (default)
- jsrun controls task/thread layout
jsrun Introduction

- Launch job on compute resources
- Similar functionality to srun and mpirun
- Launch nodes
  - Non-jsrun commands executed on launch node
Resource Set Introduction

- jsrun format:

  `jsrun [ -n #Resource Sets ] [tasks, threads, and GPUs w/in each Resource Set] program`

- Resource set
  - Subgroup of resources within a node
    - GPUs, CPUs
  - *Just cgroups under the covers*
  - Building blocks of jsrun
  - Provides the ability to create subsets of nodes
    - Flexibility to add resources based on code’s requirements
  - Limitations
    - Can span sockets; can not span nodes
    - Entire cores; not hyper-thread level
    - Homogeneous by default
Choosing a Resource Set

• Understand how your code expects to interact with the system.
  – How many tasks/threads per GPU?
  – Does each task expect to see a single GPU? Do multiple tasks expect to share a GPU? Is the code written to internally manage task to GPU workload based on the number of available cores and GPUs?

• Create resource sets containing the needed GPU to task binding
  – Based on how your code expects to interact with the system, you can create resource sets containing the needed GPU and core resources.
  – If a code expects to utilize one GPU per task, a resource set would contain one core and one GPU. If a code expects to pass work to a single GPU from two tasks, a resource set would contain two cores and one GPU.

• Decide on the number of resource sets needed
  – Once you understand tasks, threads, and GPUs in a resource set, you simply need to decide the number of resource sets needed.
## Jsrun Format and Options

```
jsrun [ -n #Resource Sets ] [tasks, threads, and GPUs w/in each Resource Set] program
```

<table>
<thead>
<tr>
<th>Flags (long)</th>
<th>Flags (short)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--nrs</code></td>
<td><code>-n</code></td>
<td>Number of resource sets</td>
</tr>
<tr>
<td><code>--rs_per_host</code></td>
<td><code>-r</code></td>
<td>Number of resource sets per host (node)</td>
</tr>
<tr>
<td><code>--tasks_per_rs</code></td>
<td><code>-a</code></td>
<td>Number of MPI tasks/ranks per resource set</td>
</tr>
<tr>
<td><code>--cpu_per_rs</code></td>
<td><code>-c</code></td>
<td>Number of CPUs (cores) per resource set.</td>
</tr>
<tr>
<td><code>--gpu_per_rs</code></td>
<td><code>-g</code></td>
<td>Number of GPUs per resource set</td>
</tr>
<tr>
<td><code>--bind</code></td>
<td><code>-b</code></td>
<td>Binding of tasks within a resource set. Can be none, rs, or packed:#</td>
</tr>
<tr>
<td><code>--latency priority</code></td>
<td><code>-l</code></td>
<td>Latency Priority. Controls layout priorities. Can currently be cpu-cpu or gpu-cpu. Upper v/s lower case.</td>
</tr>
<tr>
<td><code>--launch_distribution</code></td>
<td><code>-d</code></td>
<td>How tasks are distributed between resource sets. Can be cyclic, packed, plane.</td>
</tr>
</tbody>
</table>

*for additional flags see the jsrun man page*
jsrun examples

jsrun -n1 a.out

1 resource set

• What do you get by default?
  – Almost nothing
Explicitly requesting resources is to your benefit.
- Don’t rely on defaults.

How many of these resource sets can we fit on a single node?
**jsrun examples**

**jsrun** `-n6` `-a1` `-c1` `-g1`

- Increase the number of RS
- Each RS will contain same number of specified tasks, cores, GPUs
jsrun examples

jsrun -n6 -a1 -c7 -g1

- Increase cores per RS to utilize all available physical cores.
- Still only one task per RS
jsrun examples

jsrun -n6 -a7 -c7 -g1 -dpacked

6 resource sets x 7 tasks 7 physical cores 1 GPU

assign tasks sequentially filling RS first

- Launch distribution controls the order of tasks started across multiple resource sets
jsrun examples

**jsrun** -n6 -a7 -c7 -g1 -dcyclic

6 resource sets x 7 tasks x 7 physical cores x 1 GPU

assign tasks round robin across RS

- Change the order of tasks started across multiple resource sets to round robin

![Diagram showing resource allocation and task distribution across multiple sockets and physical cores.](image-url)
jsrun examples (threads)

jsrun -n12 -a1 -c4 -g1 -b packed:4 -d packed

12 resource sets × 1 task → 4 physical cores → 1 GPU

bind tasks to 4 cores in resource set
assign tasks sequentially filling RS first

# 2-node allocation
# OMP_NUM_THREADS = 4
# rank0's OMP_PLACES = {0:4},{4:4},{8:4},{12:4}
jsrun Binding Flag

- **-b, --bind**
- Binding of tasks within a resource set
- OMP_PLACES, affinity
- Should specify binding to help prevent unwanted oversubscription

**Options:**
- none
  - No binding
- rs
  - Bind to cores in resource set
  - **Not Recommended**
- packed: #
  - Default: packed: 1
  - Number of CPUs bound to task
- packed: smt: #
  - Hardware threads bound to task

```bash
summit-batch1> jsrun --n1 -a1 -c2 ./jsrun_layout | sort
MPI Rank 000 of 001 on HWThread 000 of Node h41n08, OMP_threadID 0 of 2
MPI Rank 000 of 001 on HWThread 000 of Node h41n08, OMP_threadID 1 of 2
```

Threads placed on same core with default binding.

```bash
summit-batch1> jsrun --n1 -a1 -c2 -b packed:2 ./jsrun_layout | sort
MPI Rank 000 of 001 on HWThread 000 of Node h41n08, OMP_threadID 0 of 2
MPI Rank 000 of 001 on HWThread 004 of Node h41n08, OMP_threadID 1 of 2
```

Use ‘–b packed:2’ to bind each rank to 2 cores.
Viewing jsrun Layout

- Execute code within interactive batch job to view jsrun layout
- Lab maintained example code:
  - [https://docs.olcf.ornl.gov/systems/summit_user_guide.html#hello-jsrun](https://docs.olcf.ornl.gov/systems/summit_user_guide.html#hello-jsrun)

```bash
summit-batch1> jsrun -n2 -a2 -d cyclic ./jsrun_layout | sort
... Warning: more than 1 task/rank assigned to a core
MPI Rank 000 of 004 on HWThread 000 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 001 of 004 on HWThread 004 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 002 of 004 on HWThread 000 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 003 of 004 on HWThread 004 of Node h41n08, OMP_threadID 0 of 1
```

Without `-c` multiple ranks are placed on single core.

```bash
summit-batch1> jsrun -n2 -a2 -c2 -d cyclic ./jsrun_layout | sort
MPI Rank 000 of 004 on HWThread 000 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 001 of 004 on HWThread 008 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 002 of 004 on HWThread 004 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 003 of 004 on HWThread 012 of Node h41n08, OMP_threadID 0 of 1
```

Adding cores to RS provides a core for each rank.

```bash
summit-batch1> jsrun -n2 -a2 -c2 -d packed ./jsrun_layout | sort
MPI Rank 000 of 004 on HWThread 000 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 001 of 004 on HWThread 004 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 002 of 004 on HWThread 008 of Node h41n08, OMP_threadID 0 of 1
MPI Rank 003 of 004 on HWThread 012 of Node h41n08, OMP_threadID 0 of 1
```

Notice default rank placement order cycles between RS.

Changing distribution order to packed changes RS rank placement.
Viewing jsrun Layout

- **js_task_info**: binary provided by jsrun developers
- Examples ran with default SMT4

```plaintext
summit-batch1> jsrun -n1 -a4 -c4 -bpacked:1 -dpacked js_task_info | sort
Task 0 ( 0/4, 0/4 ) is bound to cpu[s] 0-3 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={0:4}
Task 1 ( 1/4, 1/4 ) is bound to cpu[s] 4-7 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={4:4}
Task 2 ( 2/4, 2/4 ) is bound to cpu[s] 8-11 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={8:4}
Task 3 ( 3/4, 3/4 ) is bound to cpu[s] 12-15 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={12:4}

summit-batch1> jsrun -n1 -a4 -c4 -bpacked:smt:4 -dpacked js_task_info | sort
Task 0 ( 0/4, 0/4 ) is bound to cpu[s] 0-3 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={0:4}
Task 1 ( 1/4, 1/4 ) is bound to cpu[s] 4-7 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={4:4}
Task 2 ( 2/4, 2/4 ) is bound to cpu[s] 8-11 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={8:4}
Task 3 ( 3/4, 3/4 ) is bound to cpu[s] 12-15 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={12:4}

summit-batch1> jsrun -n1 -a4 -c4 -bpacked:smt:1 -dpacked js_task_info | sort
Task 0 ( 0/4, 0/4 ) is bound to cpu[s] 0 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={0}
Task 1 ( 1/4, 1/4 ) is bound to cpu[s] 1 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={1}
Task 2 ( 2/4, 2/4 ) is bound to cpu[s] 2 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={2}
Task 3 ( 3/4, 3/4 ) is bound to cpu[s] 3 on host a01n18 with OMP_NUM_THREADS=1 and with OMP_PLACES={3}
```

- **Default binding**
  - to one physical core
- **Binding packed**:smt:4
- **Binding packed**:smt:1
- All tasked placed on single physical core
Viewing jsrun Layout

- Execute tools on compute nodes through jsrun

```
summit-batch3> jsrun -n1 -g0 sh -c 'nvidia-smi --query-gpu=gpu_name,gpu_bus_id --format=csv'
No devices were found
```

```
summit-batch3> jsrun -n1 -g3 sh -c 'nvidia-smi --query-gpu=gpu_name,gpu_bus_id --format=csv'
name, pci.bus_id
Tesla V100-SXM2-16GB, 00000004:04:00.0
Tesla V100-SXM2-16GB, 00000004:05:00.0
Tesla V100-SXM2-16GB, 00000004:06:00.0
```

```
summit-batch3> jsrun -n1 -g4 -c42 sh -c 'nvidia-smi --query-gpu=gpu_name,gpu_bus_id --format=csv'
name, pci.bus_id
Tesla V100-SXM2-16GB, 00000004:04:00.0
Tesla V100-SXM2-16GB, 00000004:05:00.0
Tesla V100-SXM2-16GB, 00000035:03:00.0
Tesla V100-SXM2-16GB, 00000035:04:00.0
```

No visible GPUs

3 visible GPUs

Bus ID shows two GPUs per socket visible
Viewing jsrun Layout (jsrunVisualizer)

- **Job Step Viewer** - [https://jobstepviewer.olcf.ornl.gov/](https://jobstepviewer.olcf.ornl.gov/)
  - Generate a graphical view of an application's runtime layout on Summit.
  - Used to create all resource set images in this presentation.

```bash
$ module load job-step-viewer
$ jsrun -n6 -g1 -c7 -a1
```

1. Load the `job-step-viewer` module.
2. Test out a `jsrun` line by itself or run an executable as normal.
3. Visit the provided URL.
Multiple Jsruns **Sequentially**

```bash
#!/bin/bash

#BSUB -W 2:00
#BSUB -nnodes 2
#BSUB -P abc007

cd $MEMBERWORK/abc007

jsrun -n12 -r6 -g1 -a2 -c2 ./a.out fileA
jsrun -n1 -r1 -a12 -c12 ./post_process fileA fileB
jsrun -n12 -r6 -g1 -a2 -c2 ./a.out fileB
jsrun -n1 -r1 -a12 -c12 ./post_process fileB fileC
jsrun -n12 -r6 -g1 -a2 -c2 ./a.out fileC
```

**Nodes Required**
- 2 nodes: 00:25:00
- 1.1 nodes: 00:05:00

**Time Required**
- 00:25:00
- 00:05:00

**Allocated nodes should be as large as largest jsrun**

**Walltime should be long enough to run all jsruns and support tasks**

**All tasks executed sequentially**

**jsrun post_process will not launch until the previous jsrun a.out completes**

<table>
<thead>
<tr>
<th>Simultaneous Nodes</th>
<th>Total Walltime</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>01:30:00</td>
</tr>
</tbody>
</table>
Multiple Jsruns Simultaneously

```bash
#!/bin/bash

#BSUB -W 1:00
#BSUB -nnodes 6
#BSUB -P abc007

cd $MEMBERWORK/abc007

jsrun -n12 -r6 -g1 -a2 -c2 ./a.out fileA &
jsrun -n12 -r6 -g1 -a2 -c2 ./a.out fileB &
jsrun -n12 -r6 -g1 -a2 -c2 ./a.out fileC &

wait
```

Walltime should be long enough to run longest running jsrun

Allocated nodes should be as large as the sum of all simultaneous jsruns

Placing jsruns in background allows each to run at same time. Jsrun will place each on separate resources.

UNIX wait ensures script does not exit before backgrounded work completes.

Without wait, batch job will exit before jsruns are complete.
Multiple Simultaneous Job Steps

- jsrun placement managed by IBM’s CSM (Cluster System Management)
- Aware of all jsrun allocations within LSF job; allows multiple per node, multi node, ...
- Following example ran on 2-node allocation

```
summit-batch3> jsrun -n1 -a1 -c1 -g6 -bpacked:1 csh -c "js_task_info; sleep 30" &
Task 0 ( 0/1, 0/1 ) is bound to cpu[s] 0-3 on host a01n02

summit-batch3> jsrun -n1 -a1 -c42 -g0 -bpacked:1 csh -c "js_task_info; sleep 30" &
Task 0 ( 0/1, 0/1 ) is bound to cpu[s] 0-3 on host a01n01

summit-batch3> jsrun -n1 -a1 -c1 -g1 -bpacked:1 csh -c "js_task_info; sleep 30" &
```

```
summit-batch3> jslist
ID     ID       nrs      per RS      per RS     exit     status         status
------- ------- ------- ------- ------- ------- ------- --------
17     0         1            1               6            0              Running
18     0         1          42               0            0              Running
19     0         1            1               1            0              Queued
1      0         1            1               3            0             Complete
```

Requires all cores on node, placed on separate node

Not enough free resources, waiting on completion of running step

jslist command displays job steps

**Note:** In a batch job, backgrounded tasks require `wait` command
Questions?

- Documentation
  - docs.olcf.ornl.gov
  - Man pages
    - jsrun, bsub
- Help/Feedback
  - help@olcf.ornl.gov