Introduction to Job Submission on Summit

Suzanne Parete-Koon NCCS HPC Engineer

ORNL is managed by UT-Battelle LLC for the US Department of Energy
Jsrun is the job launcher for Summit
bsub -Submit a Job to LSF

● bsub allocates 1 batch node plus the number of requested compute nodes.

bsub -alloc_flags defines allocation-wide configurations

● Applied to every compute node
  ○ CPU Simultaneous Multithreading Level (smt)
  ○ GPU Multi-Process Service (gpumps)
  ○ Burst Buffer (nvme)
  ○ Others (spectral, maximizegpfs,...)
● Multiple options require quoting, space separated
  ○ #BSUB -alloc_flags "gpumps smt1 nvme"
Jsrun is the job launcher for Summit

```bash
#!/bin/bash
#BSUB -P STF007
#BSUB -J myLSFjob
#BSUB -o out.%J
#BSUB -e err.%J
#BSUB -W 30
#BSUB -n nodes 1
#BSUB -alloc_flags nvme

hostname
jsrun -n1 hostname
hostname

[[suzanne@login3.summit ~]$ bsub batchScript.lsf
Job <1849558> is submitted to default queue <batch>.
[[suzanne@login3.summit ~]$ ]
```

https://docs.olcf.ornl.gov/systems/summit_user_guide.html#batch-scripts
Summit Node

- IBM Power System AC922
- Compute Node
- 2 Sockets
  - 3 NVIDIA V100 GPUs
  - 21 usable cores
    - {1, 2, 4}-way Multithreading (SMT)
  - 256 GB DDR4 RAM
- 1.6 TB NVMe (Burst Buffer)
- Exclusive access during job
- X-bus(64GB/s)
Resource Sets

`jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program  [program args]`

`jsrun -n6 -c7 -a1, -g1 ./a.out`
Resource Sets

jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]

- Used to shape the compute node to your application by grouping
  - Physical cores
  - GPUs
  - RAM
- Must contain 1 or more physical cores and 0 or more GPUs
- Allow for multiple job steps on a single node (Adv.)
- Highly flexible with Explicit Resource Files (Adv.)

- Not as scary as they seem

jsrun -n6 -c7 -a1, -g1 .a.out
Resource Sets

jsrun [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]

Some limitations

- A resource set may span sockets, but cannot span nodes
  - Creating resource sets within sockets can avoid cross-socket communication
- Memory access requires a physical core or GPU on its host socket
- Resource Sets are homogeneous by default
  - Heterogeneous resourcesets possible with ERF (Adv.)

```
jsrun -n6 -c7 -a1, -g1 .a.out
```
Designing a Resource Set

**jsrun** [-n #resource sets] [tasks, threads, and GPUs in each resource set] program [program args]

<table>
<thead>
<tr>
<th>Flags</th>
<th>Long</th>
<th>Short</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--nrs</td>
<td>-n</td>
<td></td>
<td>Number of resource sets</td>
<td>All available physical cores</td>
</tr>
<tr>
<td>--tasks_per_rs</td>
<td>-a</td>
<td></td>
<td>Number of MPI tasks (ranks) per resource set</td>
<td>Not set by default, instead total tasks (-p) set</td>
</tr>
<tr>
<td>--cpu_per_rs</td>
<td>-c</td>
<td></td>
<td>Number of CPUs (cores) per resource set.</td>
<td>1</td>
</tr>
<tr>
<td>--gpu_per_rs</td>
<td>-g</td>
<td></td>
<td>Number of GPUs per resource set</td>
<td>0</td>
</tr>
<tr>
<td>--bind</td>
<td>-b</td>
<td></td>
<td>Binding of tasks within a resource set. Can be none, rs, or packed:</td>
<td>packed:1</td>
</tr>
<tr>
<td>--rs_per_host</td>
<td>-r</td>
<td></td>
<td>Number of resource sets per host</td>
<td>No default</td>
</tr>
<tr>
<td>--latency_priority</td>
<td>-l</td>
<td></td>
<td>Latency Priority. Controls layout priorities. Can currently be</td>
<td>gpu-cpu,cpu-mem,cpu-cpu</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cpu-cpu or gpu-cpu</td>
<td></td>
</tr>
<tr>
<td>--launch_distribution</td>
<td>-d</td>
<td></td>
<td>How tasks are started on resource sets</td>
<td>packed</td>
</tr>
</tbody>
</table>
Designing a Resource Set

1. Understand how the code expects the node to appear
   ○ How many tasks/threads per GPU?
   ○ Does each task expect to see a single GPU?
   ○ Do multiple tasks expect to share a GPU (gpumps)?
   ○ Is the code written to internally manage task to GPU workloads based on the number of available cores and GPUs?
2. Create Resource Sets containing the needed GPU to task binding
   ○ Describe a resource set that meets the requirements above. If the code is written for one GPU per task, consider a resource set with just one GPU
3. Decide on the number of Resource Sets needed
   ○ After understanding task, thread, and GPU requirements, scale the number of Resource Sets (and number of nodes) as needed.

\texttt{jsrun [-n \#resource sets] [tasks, threads, and GPUs in each resource set] program [program args]}
Examples

**jsrun -n1**
- By default you get one core with one task.
- This is the same as

**jsrun -n1 -c1 -a1**
- Better not to rely on defaults

Let’s add a GPU . . .
Examples

`jsrun -n1 -c1 -a1 -g1`

- `-g` flag specifies the number of GPUs per resource set.
- How many of these sets can we fit on the node?
Examples

```bash
jsrun -n6 -c1 -a1 -g1
```

- How would this scale to two or more nodes?
Examples: Multiple Nodes

`jsrun -n12 -c1 -a1 -g1`

- `-n` will be the number of nodes you reserved multiplied by the number of resource sets on one node.
- If your resource sets don’t fill both sockets, `jsrun` may not map the resources set as you expect when you expand to multiple nodes.
- Let’s talk about that and threads next.
Examples: Threading

```
jsrun -n1 -c7 -a1 -EOMP_NUM_THREADS=7 -g1
```

- `-EOMP_NUM_THREADS=#` lets you set the number of OpenMP threads per task.
- Could also do `export OMP_NUM_THREADS=7` just before the `jsrun` command.
- All 7 threads are clustered on the same core!
- How do we fix this?
Examples: Threading -brs

\texttt{jsrun -n1 -c7 -a1 -EOMP_NUM_THREADS=7 -g1 -brs}

- \texttt{-b} controls the thread binding options are packed: #, rs, none
- allows you to set the number of physical cores available to an MPI task
- \texttt{-brs} binds the threads to fill the available cores. In the resource set
- How would I do this over 2 nodes?
Examples: Threading multiple nodes

`jsrun -n2.c7 -a1 -EOMP_NUM_THREADS=7 -g1 -brs`
Examples: Threading multiple nodes

```
jsrun -n2 c7 -a1 -EOMP_NUM_THREADS=7 -g1 -brs
```
Examples: Threading multiple nodes

\texttt{jsrun -n2 -r1 -c7 -a1 -EOMP_NUM_THREADS=7 -g1 -brs}

But this still might be a waste of allocation.

Remember you must have a core or GPU reserved on each socket to get access to that socket's memory, so what you see above does not give each resource set 512 GB, but rather only 265 GB.
Examples: Threading multiple nodes

```
jsrun -n2 --rs_per_socket 1 -c7 -a1 -EOMP_NUM_THREADS=7 -g1 -brs
```

- Setting the resource sets per socket to 1 gives you one set on each socket, each able to get full access to that node’s 265 GB of RAM.
- This gives you a more efficient use of allocation than the configuration on the previous slide.
Resources

- **OLCF user docs for running jobs:**
  [https://docs.olcf.ornl.gov/systems/summit_user_guide.html#running-jobs](https://docs.olcf.ornl.gov/systems/summit_user_guide.html#running-jobs)
  - Covers main jsrun options with examples of each.
  - Also options for monitoring your jobs.

- **man jsrun**
  - This has the complete descriptions of all Jsrun options and advice about how combinations of those options will be interpreted.

- **Job-step-viewer**
  - Demo: [https://jobstepviewer.olcf.ornl.gov](https://jobstepviewer.olcf.ornl.gov)
  - How To: [https://docs.olcf.ornl.gov/systems/summit_user_guide.html#job-step-viewer](https://docs.olcf.ornl.gov/systems/summit_user_guide.html#job-step-viewer)

- **New User Quick Start guide:** [https://docs.olcf.ornl.gov/quickstart/index.html](https://docs.olcf.ornl.gov/quickstart/index.html)
  - Has slides and recordings of past jsrun trainings (and many other new user topics)
  - Hands on exercises with answers for jsrun
Hands-On
New User Quick Start

Direct your browser to https://docs.olcf.ornl.gov/quickstart/index.html

Login to summit.
$ ssh your_user_name@summit.olcf.ornl.gov

(PIN followed by your RSA token code.)

- git clone https://github.com/olcf/NewUserQuickStart.git
  - cd NewUserQuickStart
- Do Basic Workflow to learn about Batch scripts and job submission
  - $ cd hands-on/Basic_Workflow/
- Do Jsrun basics to practice what you have learned in this lecture.
  - cd hands-on/jsrun_Job_Launcher/

I’ll go to the repo now and walk you through the exercises . . .