Summit jsrun
Introduction

OLCF February User Call

Chris Fuson

February 28, 2018
## Summit Parallel Job Execution

### Batch System

<table>
<thead>
<tr>
<th>LSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Allocates resources</td>
</tr>
<tr>
<td>• Batch scheduler</td>
</tr>
<tr>
<td>• Similar functionality to PBS/MOAB</td>
</tr>
<tr>
<td>• Allocates entire nodes</td>
</tr>
</tbody>
</table>

### Job Launcher

<table>
<thead>
<tr>
<th>jsrun</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Developed by IBM for the Oak Ridge and Livermore CORAL systems</td>
</tr>
<tr>
<td>• Similar functionality to aprun and mpirun</td>
</tr>
</tbody>
</table>
LSF Example Batch Script

Batch script example

```bash
#!/bin/bash

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

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

Batch submission

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

```
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> jsrun -n2 -r1 hostname
a01n01
a01n02
summit-batch1 308>
```
## Common LSF Commands

<table>
<thead>
<tr>
<th>Function</th>
<th>PBS/MOAB</th>
<th>LSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit</td>
<td>qsub</td>
<td>bsub</td>
</tr>
<tr>
<td>Monitor Queue</td>
<td>showq/qstat</td>
<td>bjobs</td>
</tr>
<tr>
<td>Alter Queued Job</td>
<td>qalter</td>
<td>bmod</td>
</tr>
<tr>
<td>Remove Queued Job</td>
<td>qdel</td>
<td>bkill</td>
</tr>
<tr>
<td>Hold Queued Job</td>
<td>qhold</td>
<td>bstop</td>
</tr>
<tr>
<td>Release Held Job</td>
<td>qrls</td>
<td>bresume</td>
</tr>
</tbody>
</table>
**Summit Node**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>28 29</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>30 31</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>32 33</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>34 35</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>36 37</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>38 39</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>40 41</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>42 43</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>44 45</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>46 47</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>48 49</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>50 51</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>52 53</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>54 55</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
<td>58 59</td>
</tr>
<tr>
<td>60</td>
<td>61</td>
<td>62 63</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>66 67</td>
</tr>
<tr>
<td>68</td>
<td>69</td>
<td>70 71</td>
</tr>
<tr>
<td>72</td>
<td>73</td>
<td>74 75</td>
</tr>
<tr>
<td>76</td>
<td>77</td>
<td>78 79</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82 83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>89</td>
<td>116 117</td>
</tr>
<tr>
<td>90</td>
<td>91</td>
<td>118 119</td>
</tr>
<tr>
<td>92</td>
<td>93</td>
<td>120 121</td>
</tr>
<tr>
<td>94</td>
<td>95</td>
<td>122 123</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>124 125</td>
</tr>
<tr>
<td>98</td>
<td>99</td>
<td>126 127</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>128 129</td>
</tr>
<tr>
<td>102</td>
<td>103</td>
<td>130 131</td>
</tr>
<tr>
<td>104</td>
<td>105</td>
<td>132 133</td>
</tr>
<tr>
<td>106</td>
<td>107</td>
<td>134 135</td>
</tr>
<tr>
<td>108</td>
<td>109</td>
<td>136 137</td>
</tr>
<tr>
<td>110</td>
<td>111</td>
<td>138 139</td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>140 141</td>
</tr>
<tr>
<td>114</td>
<td>115</td>
<td>142 143</td>
</tr>
<tr>
<td>144</td>
<td>145</td>
<td>146 147</td>
</tr>
<tr>
<td>148</td>
<td>149</td>
<td>150 151</td>
</tr>
<tr>
<td>152</td>
<td>153</td>
<td>154 155</td>
</tr>
<tr>
<td>156</td>
<td>157</td>
<td>158 159</td>
</tr>
<tr>
<td>160</td>
<td>161</td>
<td>162 163</td>
</tr>
<tr>
<td>164</td>
<td>165</td>
<td>166 167</td>
</tr>
<tr>
<td>168</td>
<td>169</td>
<td>170 171</td>
</tr>
</tbody>
</table>

- **GPU (3 x Socket)**
- **Socket (2 x Node)**
- **Core (21 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 (default)
  – #BSUB – alloc_flags smt2
  – #BSUB – alloc_flags smt4
• jsrun controls task/thread layout
**jsrun Introduction**

- Launch job on compute resources
- Similar functionality to aprun and mpirun
- Still in development
- Launch nodes
  - Similar to Titan
  - Non-jsrun commands executed on launch node
  - Shared resource
- Multiple jsruns per node
## Basic jsrun Examples

<table>
<thead>
<tr>
<th>Description</th>
<th>Jsrun command</th>
<th>Layout notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 MPI tasks, no GPUs</td>
<td><code>jsrun -n 64 ./a.out</code></td>
<td>2 nodes: 42 tasks node1, 22 tasks on node2</td>
</tr>
<tr>
<td>12 MPI tasks each with access to 1 GPU</td>
<td><code>jsrun -n 12 -a 1 -c 1 -g1 ./a.out</code></td>
<td>2 nodes, 3 tasks per socket</td>
</tr>
<tr>
<td>12 MPI tasks each with 4 threads and 1 GPU</td>
<td><code>jsrun -n 12 -a 1 -c 4 -g1 -bpacked:4 ./a.out</code></td>
<td>2 nodes, 3 tasks per socket</td>
</tr>
<tr>
<td>24 MPI tasks two tasks per GPU</td>
<td><code>jsrun -n 12 -a 2 -c 2 -g1 ./a.out</code></td>
<td>2 nodes, 6 tasks per socket</td>
</tr>
<tr>
<td>4 MPI tasks each with 3 GPUs</td>
<td><code>jsrun -n 4 -a 1 -c 1 -g 3 ./a.out</code></td>
<td>2 nodes: 1 task per socket</td>
</tr>
</tbody>
</table>
Resource Set Introduction

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

• Resource set
  
  – Sub group of resources within a node
    • GPUs, CPUs, RAM
  
  – 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
Resource Sets: Subdivide a Node

- RS provides the ability to subdivide node’s resources into smaller groups.
- The following examples show how a node could be subdivided and how many RS will fit on a node.

1. 1 Task access 1 GPU
   - Fit 3 RS per socket, 6 per node

2. 2 Tasks access 1 GPU
   - Fit 3 RS per socket, 6 per node

3. 1 Task access 3 GPUs
   - Fit 1 RS per socket, 2 per node

4. 6 Tasks access 3 GPUs
   - Fit 1 RS per socket, 2 per node
Resource Sets: Multiple Methods

- Create resource sets based on code
- Example: two MPI tasks, single GPU
- 3 example methods
  1. RS containing 2 cores and 1 GPU
     • Cores can only see 1 GPU
  2. RS containing 6 cores and 3 GPUs
     • 6 cores can see 3 GPUs (socket)
  3. RS containing 12 cores and 6 GPUs
     • 12 cores can see 6 GPUs (node)
1) RS Example: 2 Tasks per GPU
Resource Set per per GPU

6 resource sets per node: 1 GPU, 2 cores per (Titan)

Individual RS

• CPUs can only see single assigned GPU
2) RS Example: 2 Tasks per GPU
Resource Set per Socket

2 resource sets per node: 3 GPUs and 6 cores per socket

- All 6 CPUs can see 3 GPUs. Code must manage CPU -> GPU communication.
- CPUs on socket0 can not access GPUs or Memory on socket1.
3) RS Example: 2 Tasks per GPU Resource Set per Node

Single resource set per node: 6 GPUs, 12 cores

- All 12 CPUs can see all node’s 6 GPUs. Code must manage CPU to GPU communication.

- CPUs on socket0 can access GPUs and Memory on socket1.

- Code must manage cross socket communication.
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</th>
<th>Flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(long)</td>
<td>(short)</td>
<td></td>
</tr>
<tr>
<td>--nrs</td>
<td>-n</td>
<td>Number of resource sets</td>
</tr>
<tr>
<td>--tasks_per_rs</td>
<td>-a</td>
<td>Number of tasks per resource set</td>
</tr>
<tr>
<td>--cpu_per_rs</td>
<td>-c</td>
<td>Number of CPUs (cores) per resource set.</td>
</tr>
<tr>
<td>--gpu_per_rs</td>
<td>-g</td>
<td>Number of GPUs per resource set</td>
</tr>
<tr>
<td>--bind</td>
<td>-b</td>
<td>Binding of tasks within a resource set. Can be none, rs, or packed: #</td>
</tr>
<tr>
<td>--rs_per_host</td>
<td>-r</td>
<td>Number of resource sets per host (node)</td>
</tr>
<tr>
<td>--latency priority</td>
<td>-l</td>
<td>Latency Priority. Controls layout priorities. Can currently be cpu-cpu or gpu-cpu</td>
</tr>
<tr>
<td>--launch_distribution</td>
<td>-d</td>
<td>How tasks are started on resource sets</td>
</tr>
</tbody>
</table>
**jsrun to aprun Comparisons**

- Comparing Titan’s aprun to Summit’s jsrun
- Due to node and launcher differences, no direct equivalent for many use cases
- Table below lists basic single GPU use cases

<table>
<thead>
<tr>
<th>GPUs per Task</th>
<th>MPI Tasks</th>
<th>Threads per Task</th>
<th>aprun</th>
<th>jsrun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>aprun –n1</td>
<td>jsrun -n1 -g1 -a1 -c1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>aprun –n2</td>
<td>jsrun –n1 -g1 -a2 –c2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>aprun –n1 –d4</td>
<td>jsrun -n1 -g1 -a1 -c4 –bpacked:4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
<td>aprun –n2 –d8</td>
<td>jsrun -n1 -g1 -a2 –c16 –bpacked:8</td>
</tr>
</tbody>
</table>
# Basic jsrun Examples

<table>
<thead>
<tr>
<th>Description</th>
<th>Jsrun command</th>
<th>Layout notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 MPI tasks, no GPUs</td>
<td><code>jsrun -n 64 ./a.out</code></td>
<td>2 nodes: 42 tasks node1, 22 tasks on node2</td>
</tr>
<tr>
<td>12 MPI tasks each with access to 1 GPU</td>
<td><code>jsrun -n 12 -a 1 -c 1 -g1 ./a.out</code></td>
<td>2 nodes, 3 tasks per socket</td>
</tr>
<tr>
<td>12 MPI tasks each with 4 threads and 1 GPU</td>
<td><code>jsrun -n 12 -a 1 -c 4 -g1 -bpacked:4 ./a.out</code></td>
<td>2 nodes, 3 tasks per socket</td>
</tr>
<tr>
<td>24 MPI tasks two tasks per GPU</td>
<td><code>jsrun -n 12 -a 2 -c 2 -g1 ./a.out</code></td>
<td>2 nodes, 6 tasks per socket</td>
</tr>
<tr>
<td>4 MPI tasks each with 3 GPUs</td>
<td><code>jsrun -n 4 -a 1 -c 1 -g 3 ./a.out</code></td>
<td>2 nodes: 1 task per socket</td>
</tr>
</tbody>
</table>
12 MPI Tasks: 1 GPU each

```
jsrun  -n 12  -a 1  -c 1  -g1  ./a.out
```

12 resource sets x 1 task = 12 tasks

1 physical core x 1 GPU = 1 GPU

Specify key flags each submission, do not rely on defaults

First RS (red) contains:
- task 0
- core 0
- GPU 0
### 24 MPI Tasks: 2 Tasks per GPU

**jsrun**  
- **n** 12  
- **a** 2  
- **c** 2  
- **g1**  
- **d** packed  

<table>
<thead>
<tr>
<th>Assignment Details</th>
<th>Resource Sets</th>
<th>Tasks</th>
<th>Physical Cores</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>12</strong> resource sets</td>
<td>x</td>
<td>2 tasks</td>
<td>2 physical cores</td>
<td>1 GPU</td>
</tr>
</tbody>
</table>

**Increase cores in RS as needed to prevent accidental oversubscription**

**Packed distribution option** places tasks sequentially *(not currently default)*

**First RS (red) contains**
- 2 tasks (0-1)
- 2 cores (0,4)
- 1 GPU (0)
24 MPI Tasks: 3 GPUs each

```
jsrun -n 4 -a 6 -c 6 -g3 -d packed -l GPU-CPU ./a.out
```

- 4 resource sets
- 6 tasks
- 6 physical cores
- 3 GPUs

Assign tasks sequentially filling RS first
Assign tasks for best CPU to GPU transfers

- -l latency flag impacts core layout

First RS (red) contains
- 6 tasks (0-5)
- 6 cores(0,4,...,20)
- 3 GPUs (0-2)
12 MPI Tasks: 4 Threads, 1 GPU each

```
jsrun --n 12 --a 1 --c 4 --g 1 --b packed:4 --d packed ./a.out
```

12 resource sets \times 1 task \times 4 physical cores \times 1 GPU

User should set OMP_NUM_THREADS = 4

For rank 0 jsrun will set

OMP_PLACES \{0\},\{4\},\{8\},\{12\}

First RS (red) contains
- 1 task (0)
- 4 threads (0-3)
- 4 cores (0,4,…12)
- 1 GPU (0)

Assign tasks sequentially filling RS first
jsrun Binding Flag

• -b, --bind
• Binding of tasks within a resource set
• OMP_PLACES, affinity
• Options:
  – none
    • No binding, allow
  – rs
    • Bind to cores in resource set
  – packed:#
    • Default: packed:1
    • Number of CPUs bound to task

Should specify binding in threaded launches to prevent unwanted oversubscription
Using Hardware Threads

- Each physical core contains 4 hardware threads
- Set level using LSF flag, use jsrun to oversubscribe core
  - alloc_flags smt1 (default)
    
    ```
    jsrun -n1 -c1 -a1 -bpacked:4 csh -c 'echo $OMP_PLACES'
    0
    ```

  - alloc_flags smt2
    
    ```
    jsrun -n1 -c1 -a1 -bpacked:4 csh -c 'echo $OMP_PLACES'
    {0:2}
    ```

  - alloc_flags smt4
    
    ```
    jsrun -n1 -c1 -a1 -bpacked:4 csh -c 'echo $OMP_PLACES'
    {0:4}
    ```
Hardware Threads: Multiple Threads per Core

```
jsrun -n 12 -a 1 -c 2 -g 1 -b packed:2 -d packed ./a.out
```

- **12 resource sets**
- **1 task**
- **2 physical cores**
- **1 GPU**
- **bind tasks to 2 cores in resource set**
- **assign tasks sequentially filling RS first**

User should set

```
OMP_NUM_THREADS = 4
```

#BSUB --alloc_flags smt2

For rank 0 jsrun will set

```
OMP_PLACES {0:2},{4:2}
```

First RS (red) contains
- 1 task (0)
- 4 threads (0-3)
- 2 cores (0,4)
- 1 GPU (0)
Moving Forward

• jsrun still under development
  – New releases installed as we receive them
  – Continue to provide developers feedback

• Documentation
  – www.olcf.ornl.gov/for-users/system-user-guides/summit
  – Man pages
    • jsrun, bsub

• Help/Feedback
  – help@olcf.ornl.gov