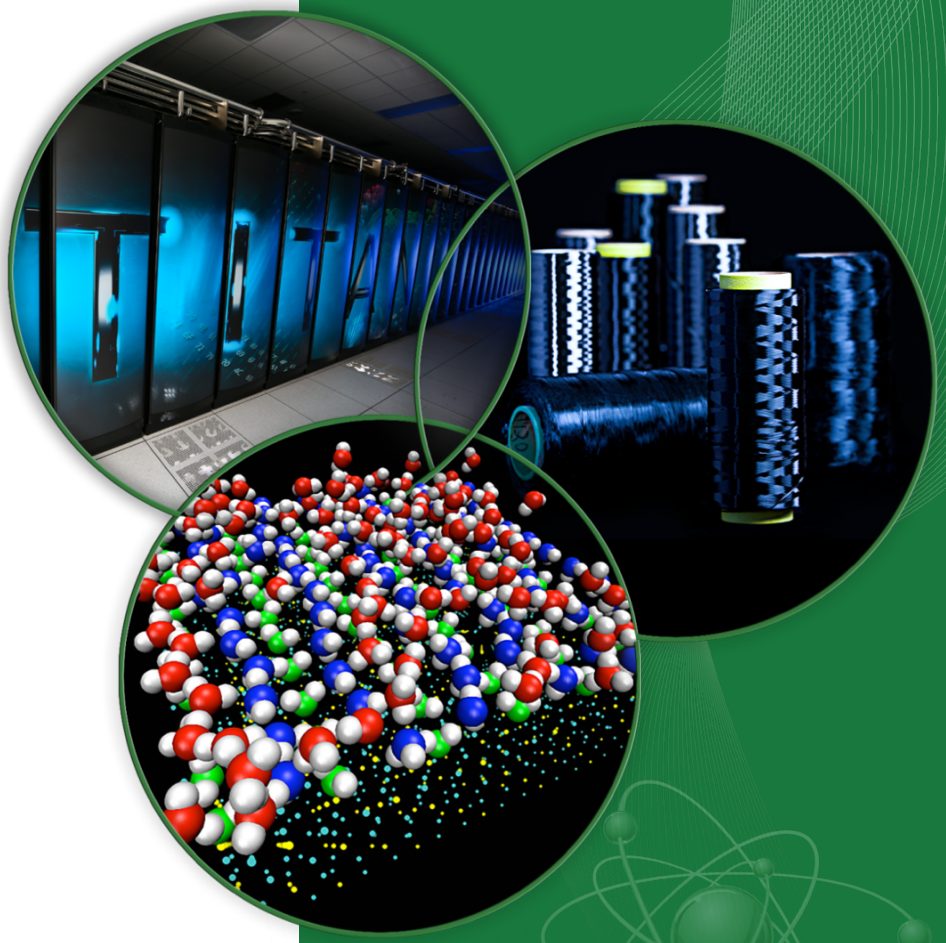
Summit jsrun Introduction

OLCF February User Call

Chris Fuson

February 28, 2018



ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Summit Parallel Job Execution

Batch System

LSF

- Allocates resources
- Batch scheduler
- Similar functionality to PBS/MOAB
- Allocates entire nodes

Job Launcher

jsrun

- Developed by IBM for the Oak Ridge and Livermore CORAL systems
- Similar functionality to aprun and mpirun

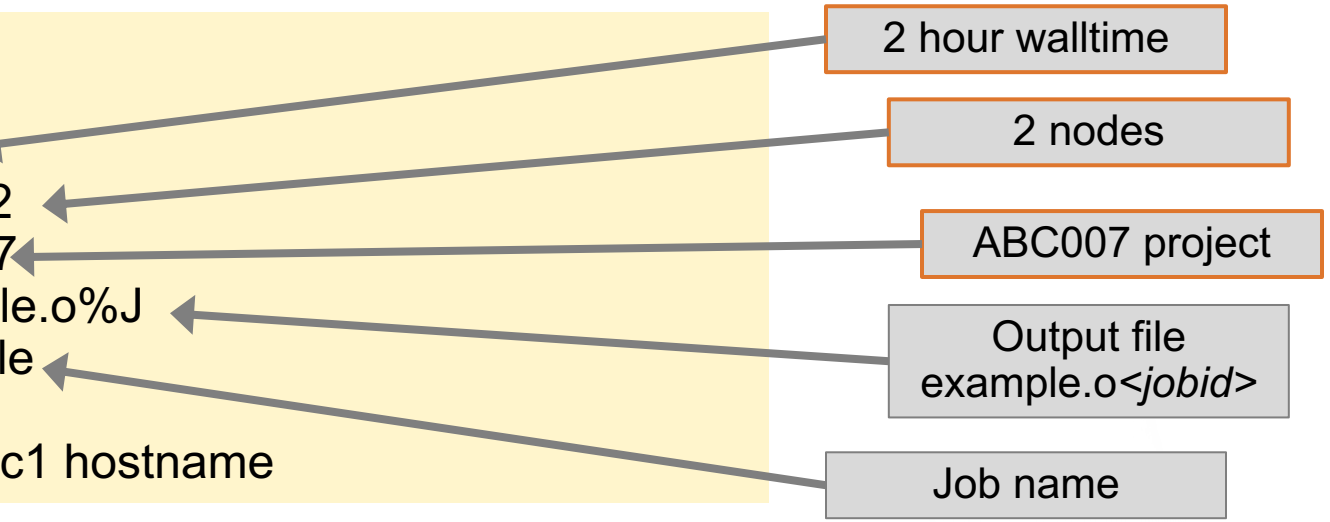
LSF Example Batch Script

Batch script example

```
#!/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

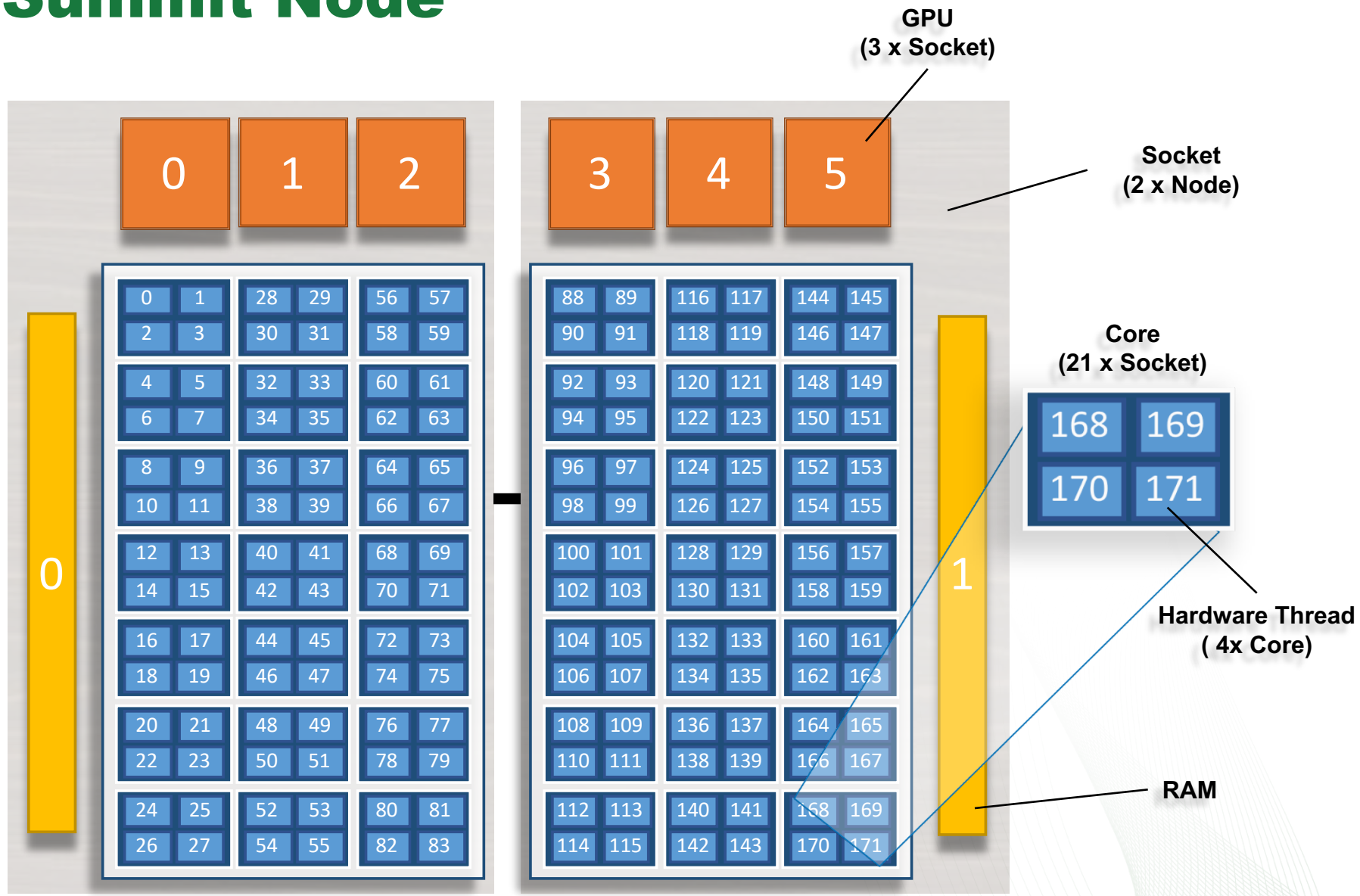
Presentation examples use the following to allocate resources

```
summit-login1> bsub -Is -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

Function	PBS/MOAB	LSF
Submit	qsub	bsub
Monitor Queue	showq/qstat	bjobs
Alter Queued Job	qalter	bmod
Remove Queued Job	qdel	bkill
Hold Queued Job	qhold	bstop
Release Held Job	qrls	bresume

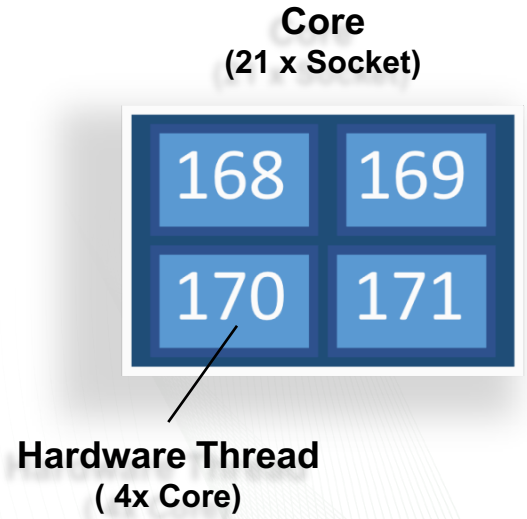
Summit Node



**Numbering skips due to core isolation*

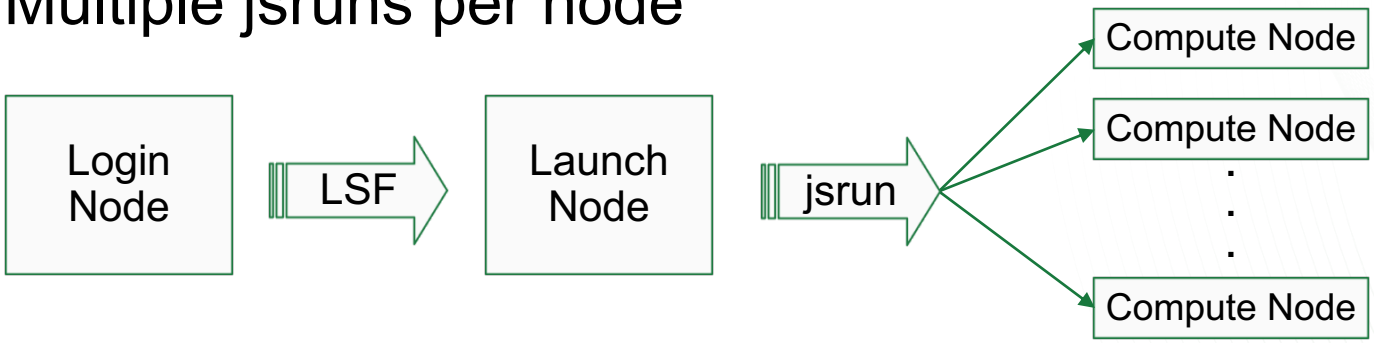
Hardware Thread Levels

- 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

Description	Jsrun command	Layout notes
64 MPI tasks, no GPUs	***jsrun –n 64 ./a.out***	2 nodes: 42 tasks node1, 22 tasks on node2
12 MPI tasks each with access to 1 GPU	***jsrun –n 12 –a 1 –c 1 –g1 ./a.out***	2 nodes, 3 tasks per socket
12 MPI tasks each with 4 threads and 1 GPU	***jsrun –n 12 –a 1 –c 4 –g1 –bpacked:4 ./a.out***	2 nodes, 3 tasks per socket
24 MPI tasks two tasks per GPU	***jsrun –n 12 –a 2 –c 2 –g1 ./a.out***	2 nodes, 6 tasks per socket
4 MPI tasks each with 3 GPUs	***jsrun -n 4 –a 1 –c 1 –g 3 ./a.out***	2 nodes: 1 task per socket

Resource Set Introduction

- jsrun format:

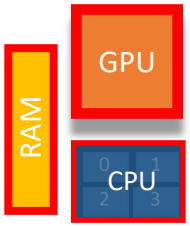
jsrun [-n #Resource Sets] [tasks, threads, and GPUs w/in each Resource Set] program

- 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

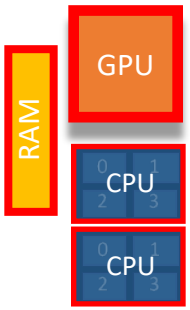
- RS provides the ability to subdivide node's resources into smaller groups.
- The following examples show how a node could be subdivided and how many RS will fit on a node.

1



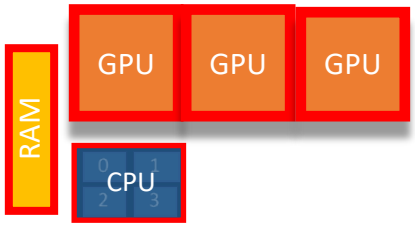
- 1 Task access 1 GPU
- Fit 3 RS per socket, 6 per node

2



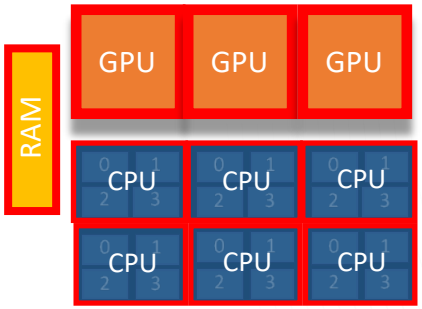
- 2 Tasks access 1 GPU
- Fit 3 RS per socket, 6 per node

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

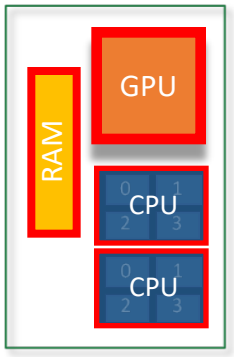
Resource Sets: Multiple Methods

- 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 GPU

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

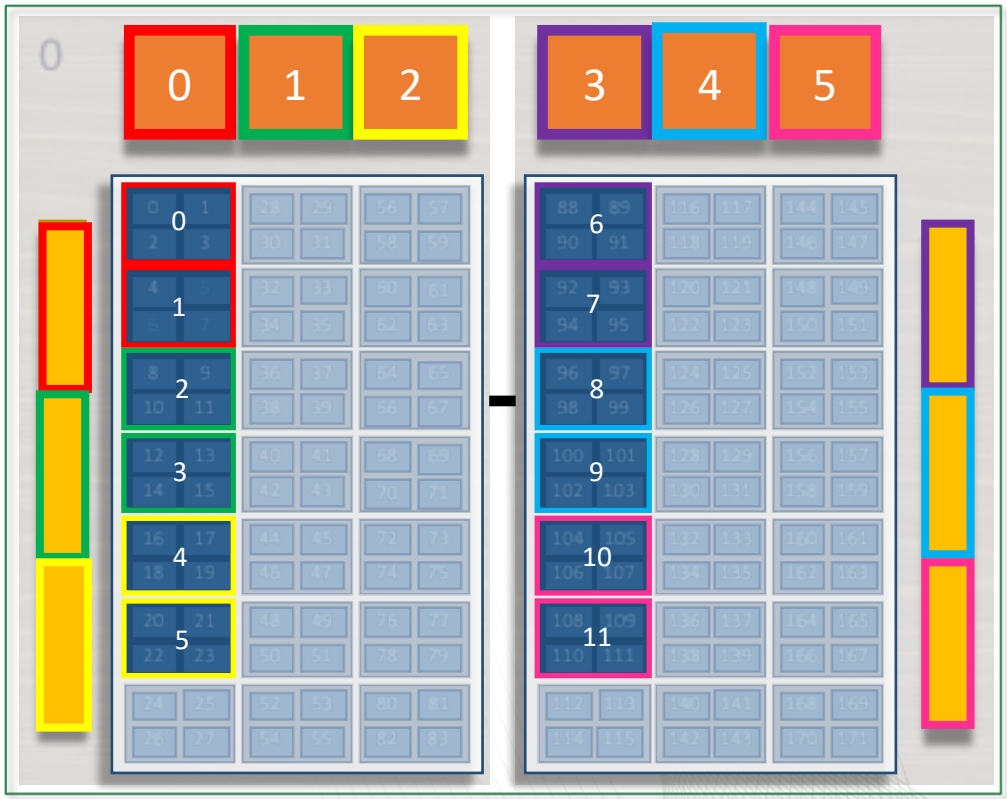
Individual RS



x6

- CPUs can only see single assigned GPU

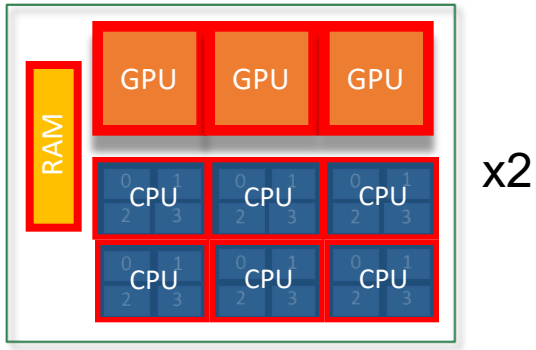
RS Mapped to Node



2) RS Example: 2 Tasks per GPU Resource Set per Socket

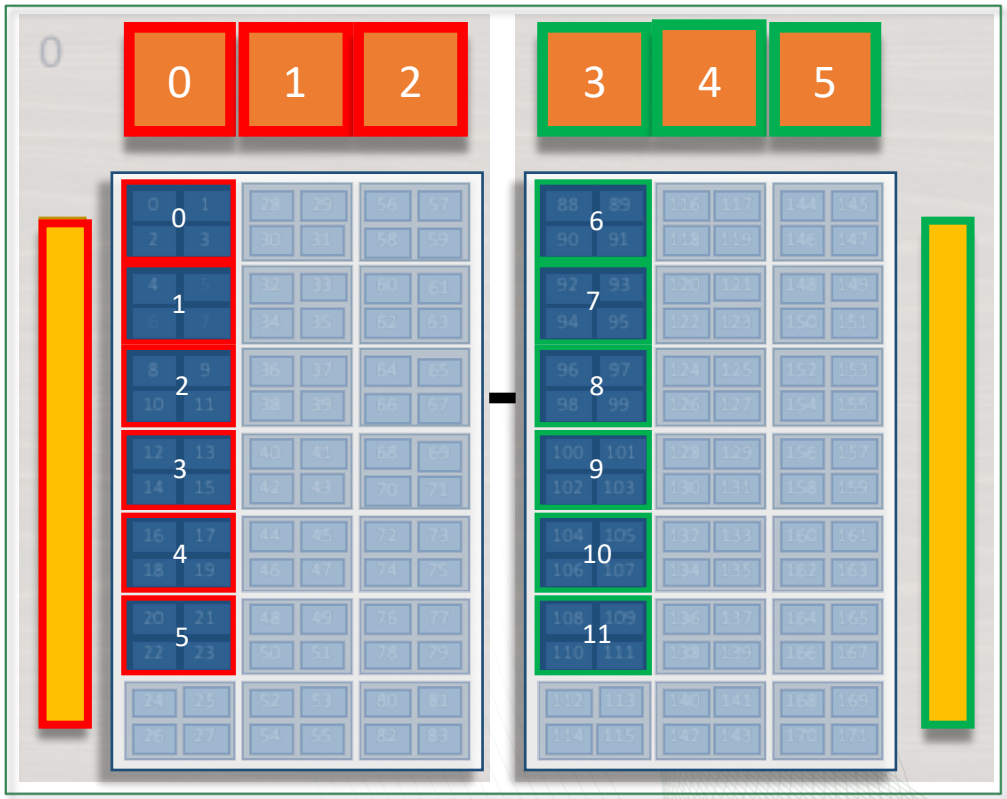
2 resource sets per node: 3 GPUs and 6 cores per socket

Individual RS



x2

RS Mapped to Node

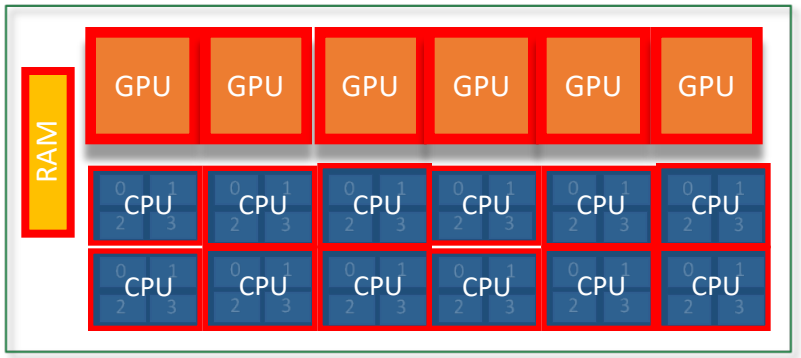


- All 6 CPUs can see 3 GPUs. Code must manage CPU -> GPU communication.
- CPUs on socket0 can not access GPUs or Memory on socket1.

3) RS Example: 2 Tasks per GPU Resource Set per Node

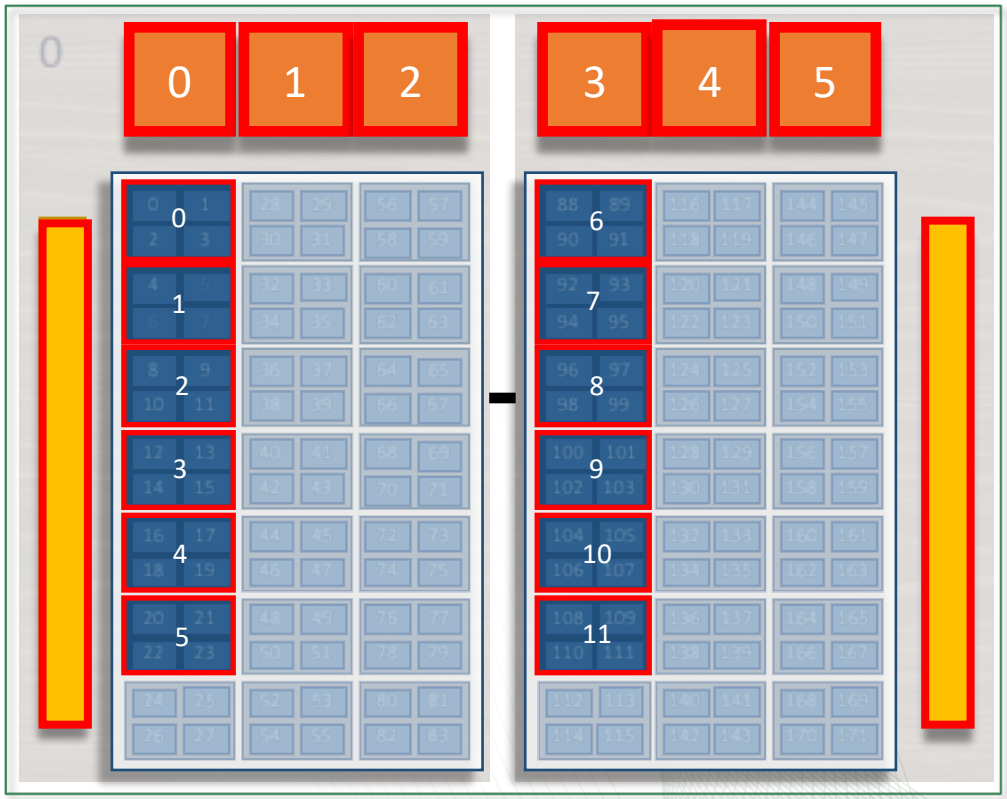
Single resource set per node: 6 GPUs, 12 cores

Individual RS



x1

RS Mapped to Node



- 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

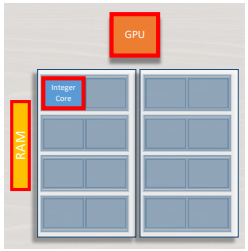
jsrun [-n #Resource Sets] [tasks, threads, and GPUs w/in each Resource Set] program

Flags (*long*)	Flags (*short*)	Description
--nrs	-n	Number of resource sets
--tasks_per_rs	-a	Number of tasks per resource set
--cpu_per_rs	-c	Number of CPUs (cores) per resource set.
--gpu_per_rs	-g	Number of GPUs per resource set
--bind	-b	Binding of tasks within a resource set. Can be none, rs, or packed:#
--rs_per_host	-r	Number of resource sets per host (node)
--latency priority	-l	Latency Priority. Controls layout priorities. Can currently be cpu-cpu or gpu-cpu
--launch_distribution	-d	How tasks are started on resource sets

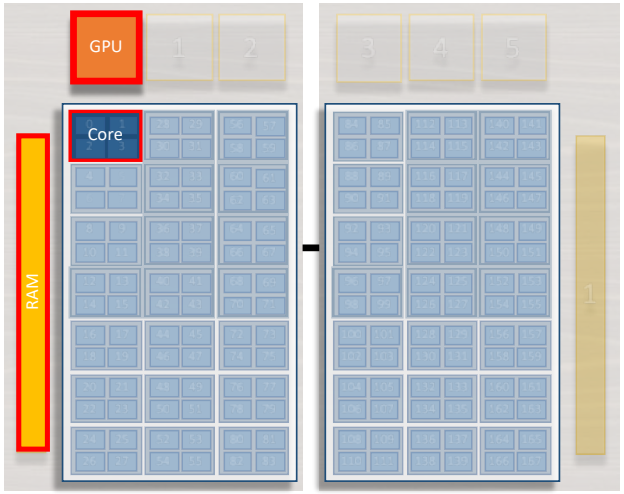
jsrun to aprun Comparisons

- 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

Titan



Summit

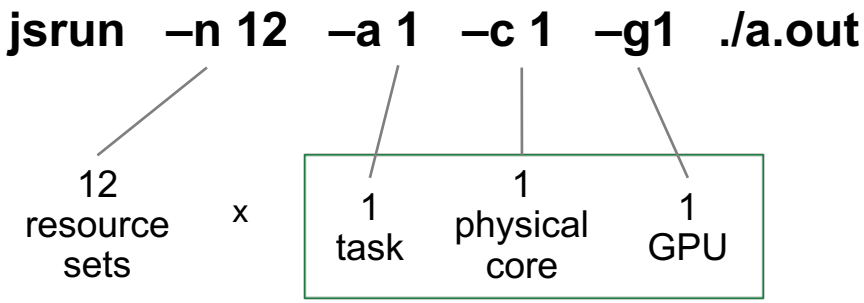


GPUs per Task	MPI Tasks	Threads per Task	aprun	jsrun
1	1	0	aprun –n1	jsrun -n1 -g1 -a1 -c1
1	2	0	aprun –n2	jsrun –n1 -g1 -a2 –c2
1	1	4	aprun –n1 –d4	jsrun -n1 -g1 -a1 -c4 –bpacked:4
1	2	8	aprun –n2 –d8	jsrun -n1 -g1 -a2 –c16 –bpacked:8

Basic jsrun Examples

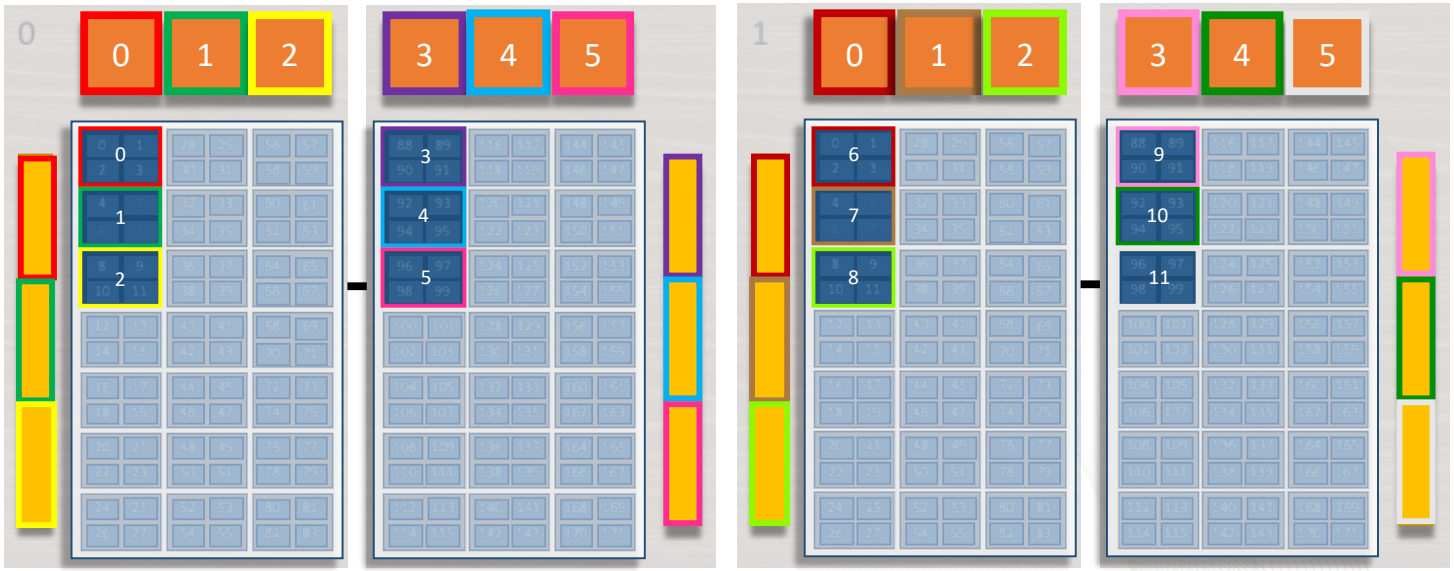
Description	Jsrun command	Layout notes
64 MPI tasks, no GPUs	***jsrun –n 64 ./a.out***	2 nodes: 42 tasks node1, 22 tasks on node2
12 MPI tasks each with access to 1 GPU	***jsrun –n 12 –a 1 –c 1 –g1 ./a.out***	2 nodes, 3 tasks per socket
12 MPI tasks each with 4 threads and 1 GPU	***jsrun –n 12 –a 1 –c 4 –g1 –bpacked:4 ./a.out***	2 nodes, 3 tasks per socket
24 MPI tasks two tasks per GPU	***jsrun –n 12 –a 2 –c 2 –g1 ./a.out***	2 nodes, 6 tasks per socket
4 MPI tasks each with 3 GPUs	***jsrun -n 4 –a 1 –c 1 –g 3 ./a.out***	2 nodes: 1 task per socket

12 MPI Tasks: 1 GPU each



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

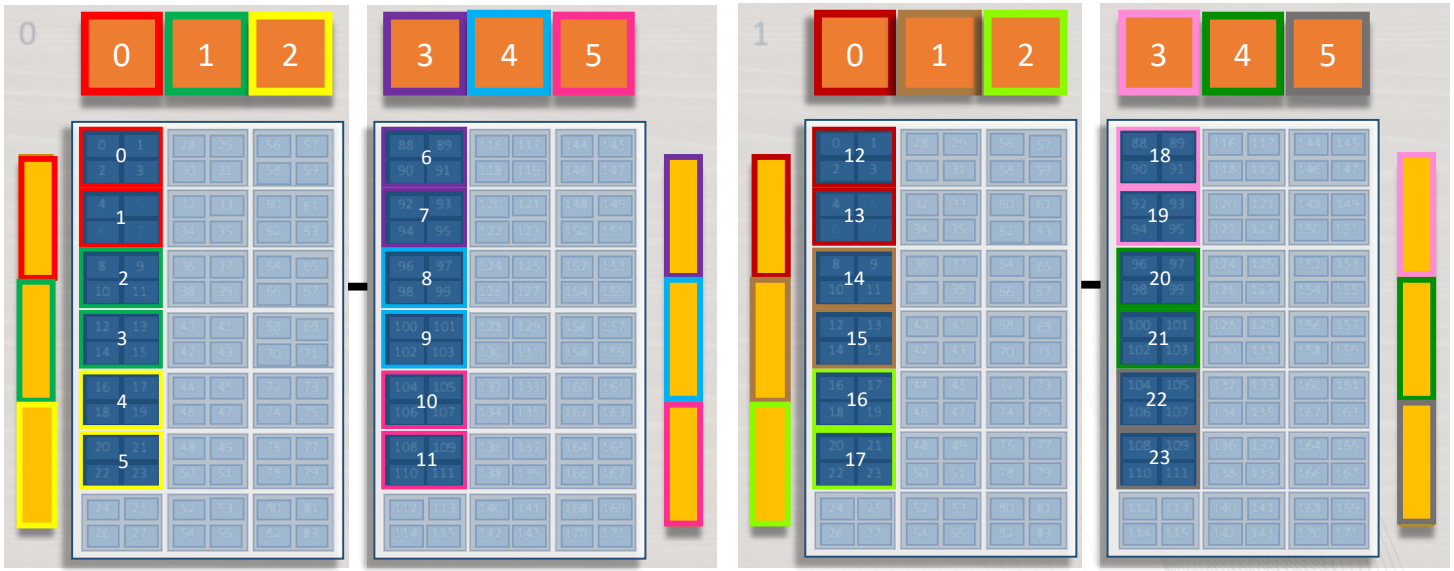
jsrun –n 12 –a 2 –c 2 –g1 –d packed ./a.out

- 12 resource sets
- x
- 2 tasks
- 2 physical cores
- 1 GPU
- assign tasks sequentially filling RS first

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)



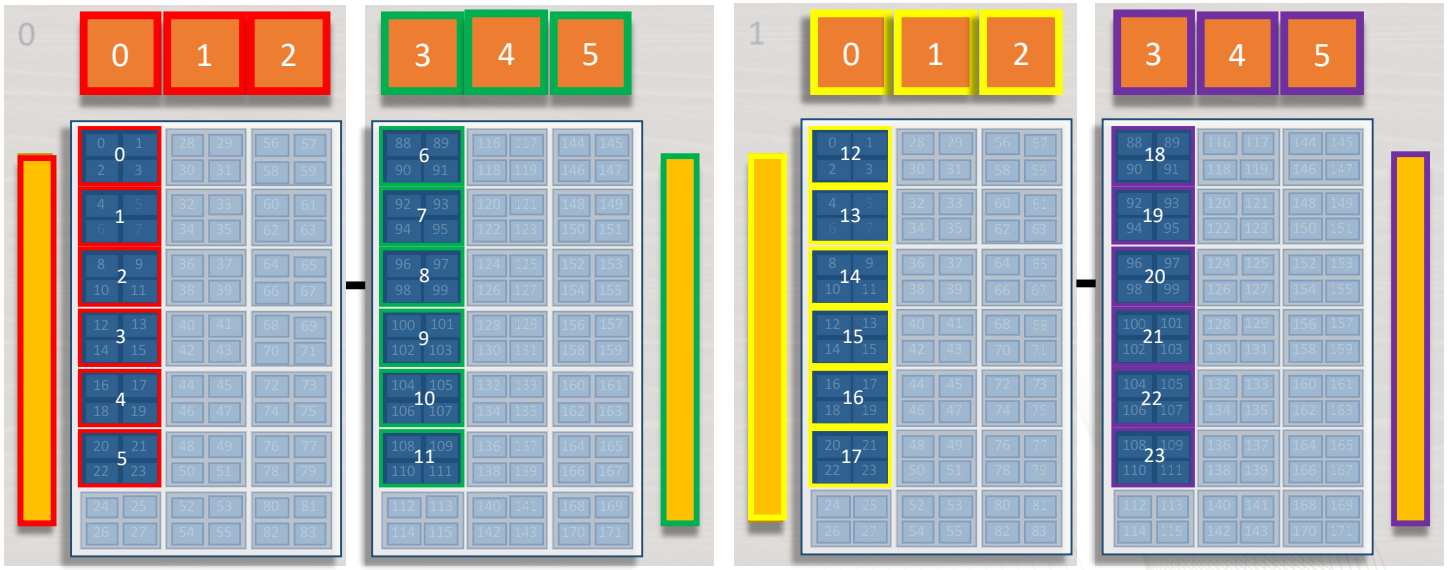
24 MPI Tasks: 3 GPUs each

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

- 4 resource sets
- x
- 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
- x
- 1 task
- 4 physical cores
- 1 GPU
- bind tasks to 4 cores in resource set
- assign tasks sequentially filling RS first

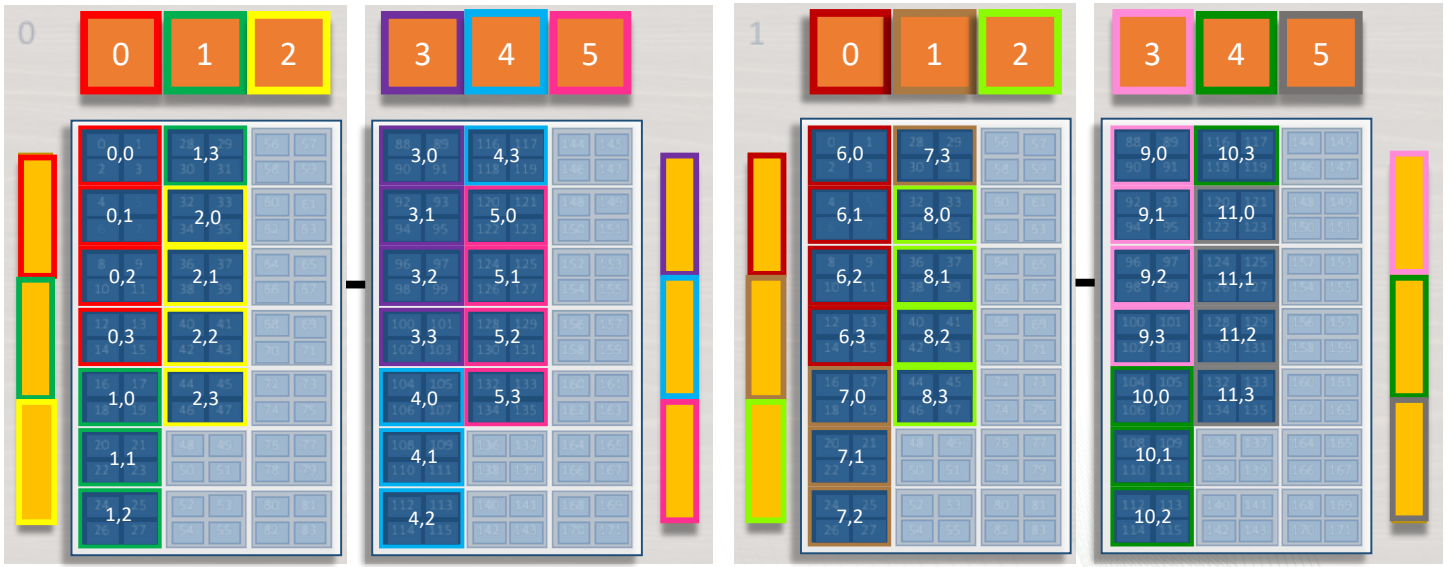
User should set OMP_NUM_THREADS = 4

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)



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
- x
- 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