OAK RIDGE National Laboratory

Summit Scheduler and Job Launch Introduction

OLCF Summit Training Workshop

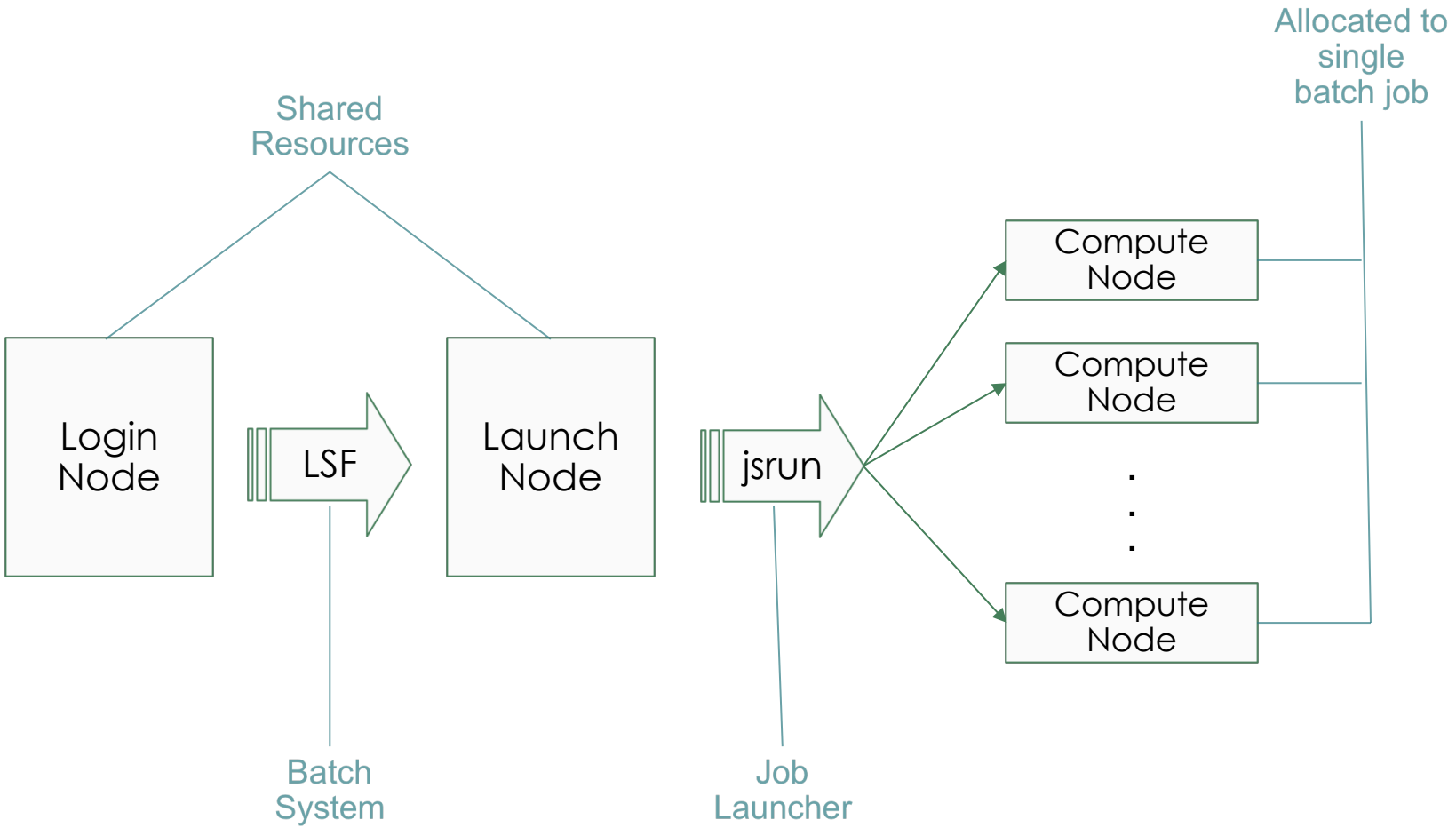
Chris Fuson

Feb 11, 2019

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

U.S. DEPARTMENT OF ENERGY

Summit Login, Launch, Compute Nodes



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

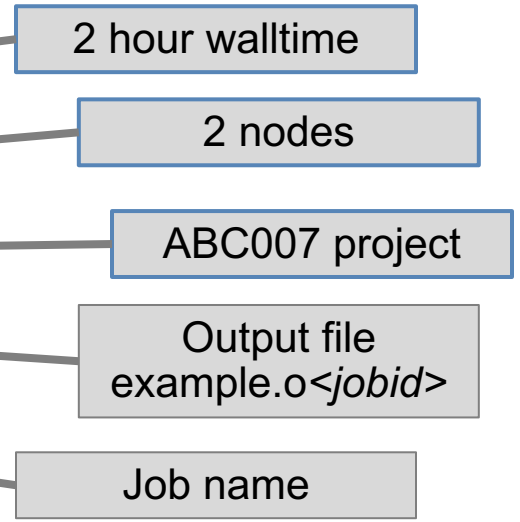
LSF Example Batch Script

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>
```

Common bsub Options

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

**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

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/jobstat
Investigate Job	checkjob	bhist
Alter Queued Job	qalter	bmod
Remove Queued Job	qdel	bkill
Hold Queued Job	qhold	bstop
Release Held Job	qrls	bresume

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

```
summit-login1 1082> jobstat
---------------------- Running Jobs: 7 (4158 of 4608 nodes, 90.23%) ----------------------
JobId    Username   Project      Nodes   Remain      StartTime         JobName
221070   userA      CSC100         512    44:22    11/30 11:01:25    run745-A3
221090   userA      CSC100         272   1:35:12   11/30 11:52:15    run745-B2
221092   userB      CSC006           1   1:06:47   11/30 11:23:50    Not_Specified
221105   userC      CSC007        3200   2:59:40   11/30 12:16:43    Not_Specified
221095   userD      CSC201           2   1:29:29   11/30 11:46:32    Not_Specified
221088   userE      CSC100         170   1:31:06   11/30 11:48:09    20_a_1
221097   userF      CSC100           1   1:52:26   11/30 12:09:29    Job3
---------------------------------------- Eligible Jobs: 2 ----------------------------------------
JobId    Username   Project      Nodes  Walltime     QueueTime      Priority    JobName
221108   userC      CSC007        4200  10:00:00   11/30 12:16:07    520.00   Not_Specified
221101   userC      CSC007        1048   6:00:00   11/30 12:12:28    515.00   Not_Specified
---------------------------------------- Blocked Jobs: 4 -----------------------------------------
JobId    Username   Project      Nodes   Walltime    BlockReason
221099   userA      CSC100        1048    6:00:00   JOBS limit defined for the user or user group has been reach
221110   userC      CSC007        1800    8:00:00   JOBS_PER_SCHED_CYCLE defined for the user or user
221107   userC      CSC007           1      45:00   JOBS_PER_SCHED_CYCLE defined for the user or user
221151   userC      CSC007          16    3:00:00   JOBS_PER_SCHED_CYCLE defined for the user or user
```

Default Job Name

Running job limit reached

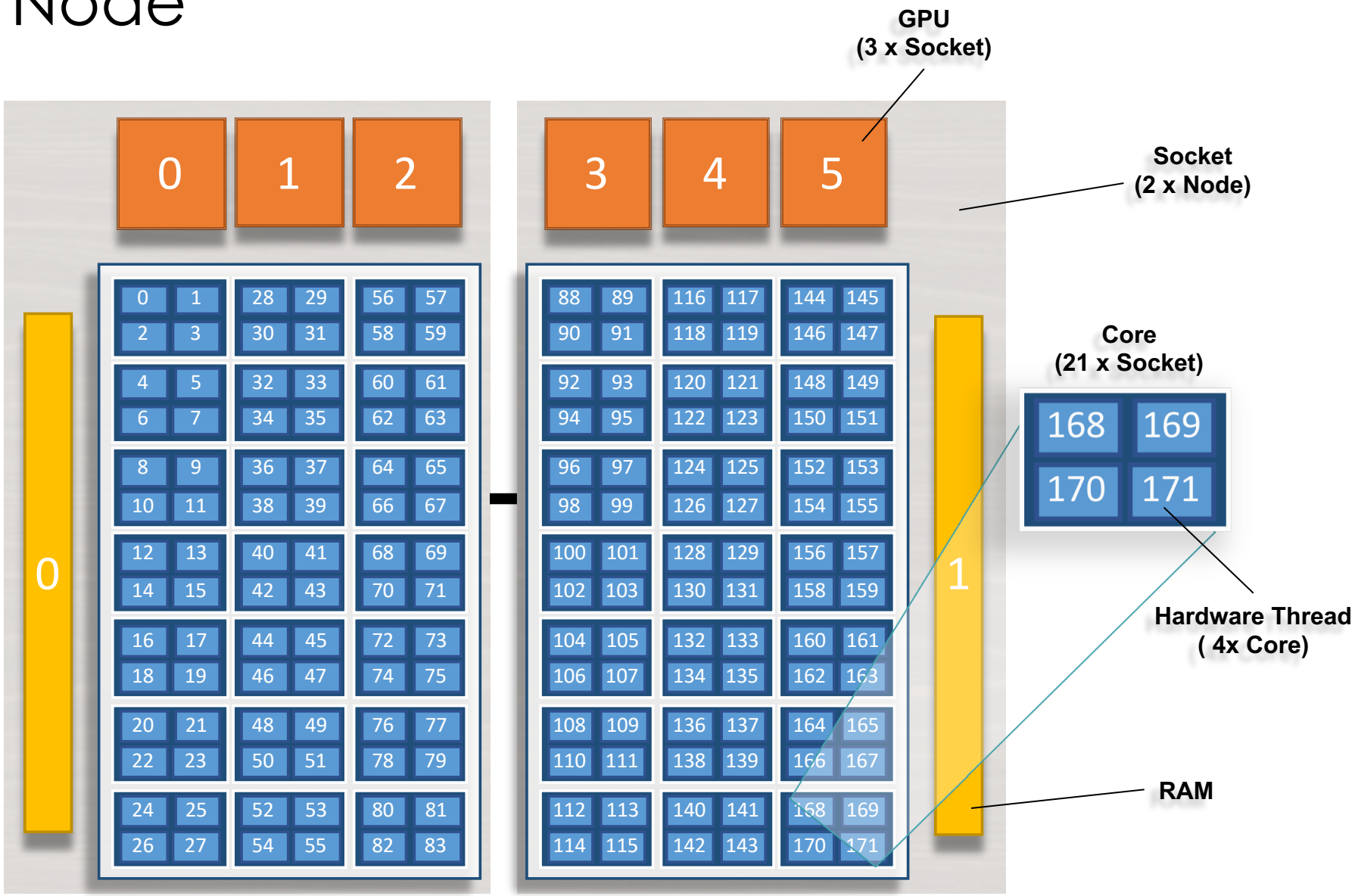
Eligible job limit reached

Summit Queue Policy

Bin	Min Nodes	Max Nodes	Max Walltime (hrs)	Aging Boost (days)
1	2,765	4,608	24	15
2	922	2,764	24	10
3	92	921	12	0
4	46	91	6	0
5	1	45	2	0

- Eligible to run limit: 2

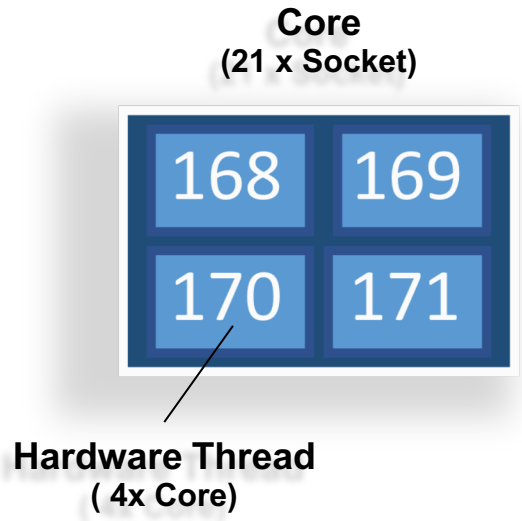
Summit Node



**Numbering skips due to core isolation*

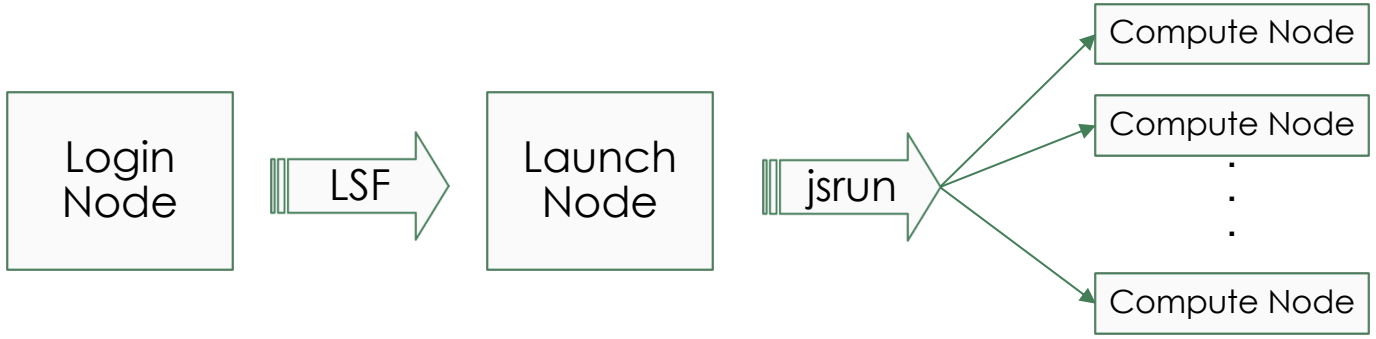
Hardware Thread Levels

- 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
  - #BSUB –alloc_flags smt2
  - #BSUB –alloc_flags smt4 (default)
- jsrun controls task/thread layout



jsrun Introduction

- Launch job on compute resources
- Similar functionality to aprun and mpirun
- 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:

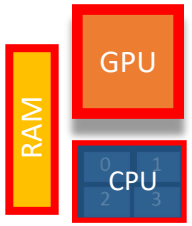
```
jsrun [ -n #Resource Sets ] [tasks, threads, and GPUs w/in each Resource Set] program
```

- 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

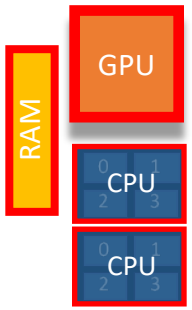
- RS provides the ability to subdivide node's resources into smaller groups.
- The following examples show how a node could be subdivided and how many RS will fit on a node.

1



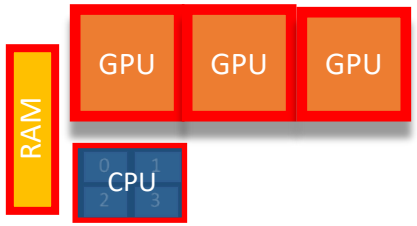
- 1 Task access 1 GPU
- Fit 3 RS per socket, 6 per node

2



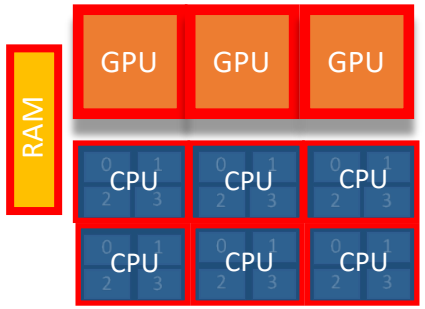
- 2 Tasks access 1 GPU
- Fit 3 RS per socket, 6 per node

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

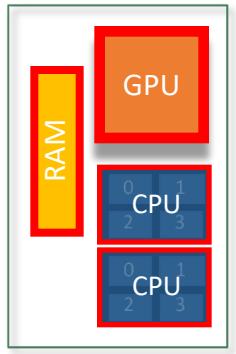
Resource Sets: Multiple Methods

- 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)

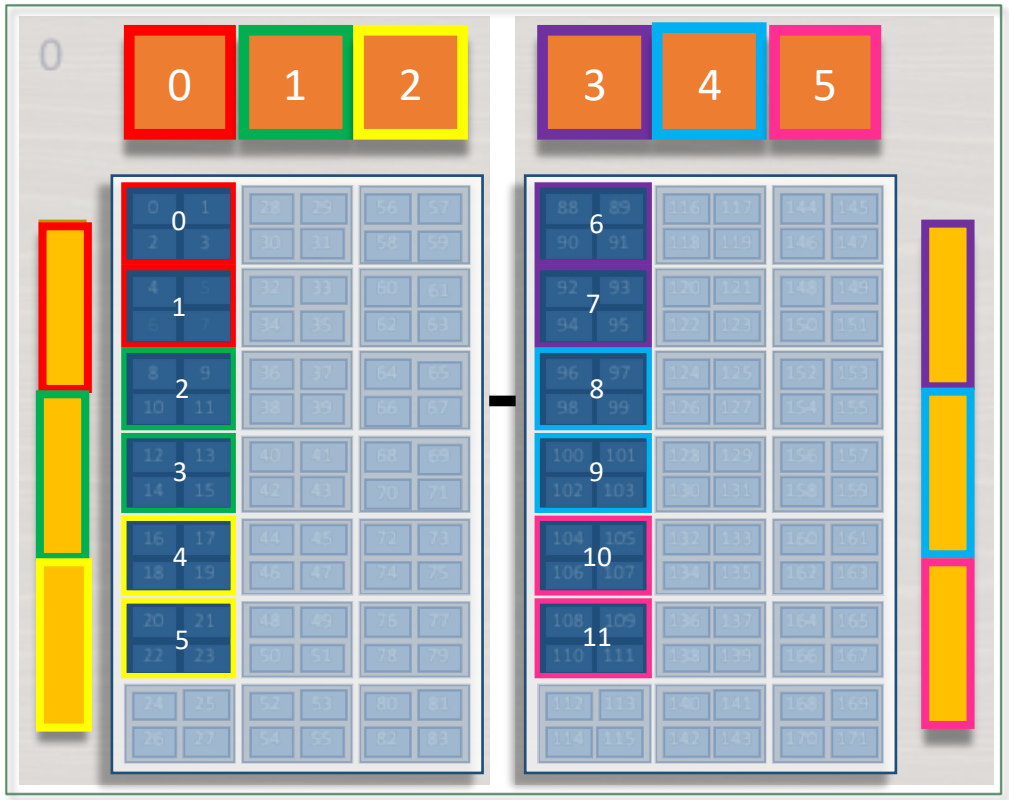
Individual RS



x6

- CPUs see single assigned GPU

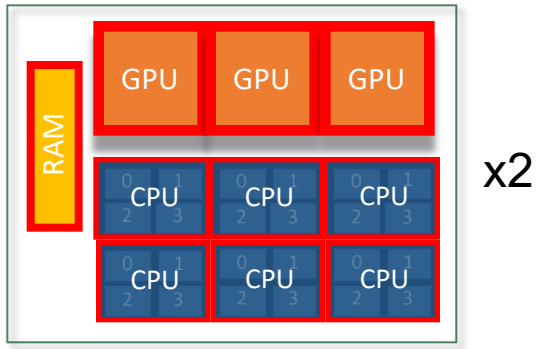
RS Mapped to Node



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

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

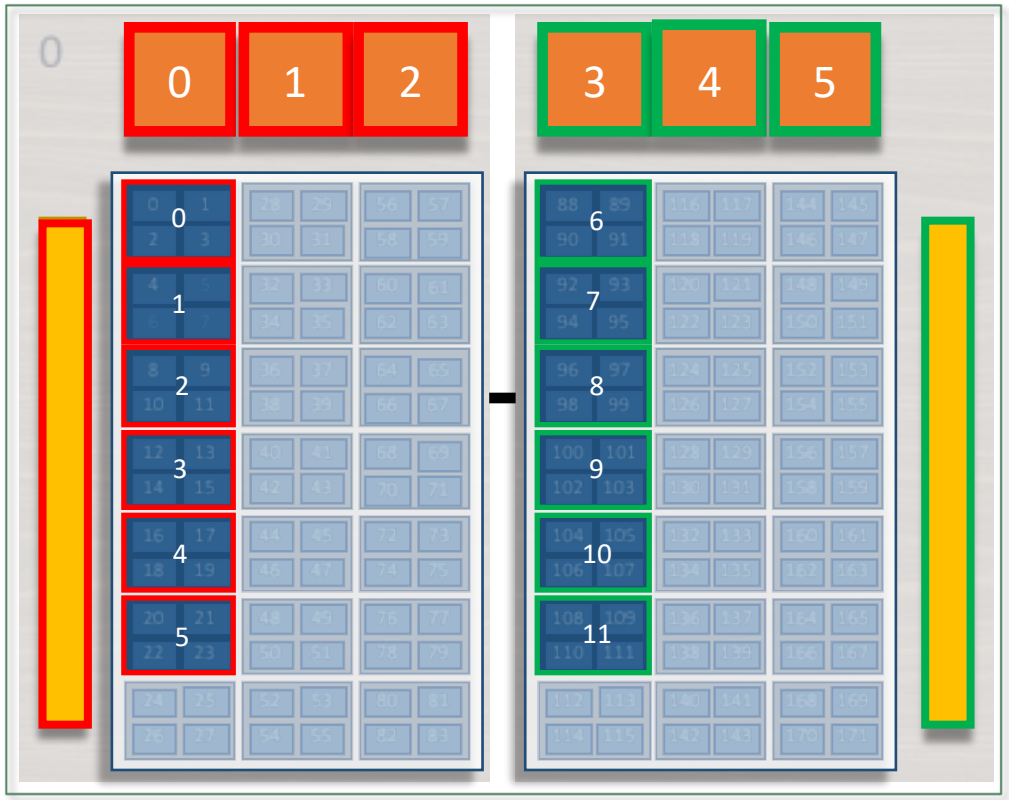
Individual RS



x2

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

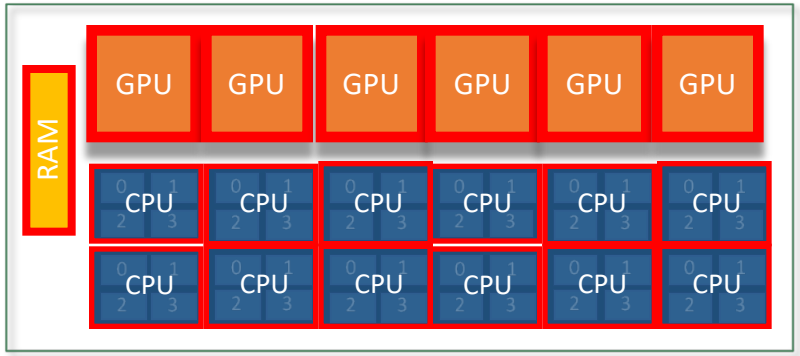
RS Mapped to Node



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

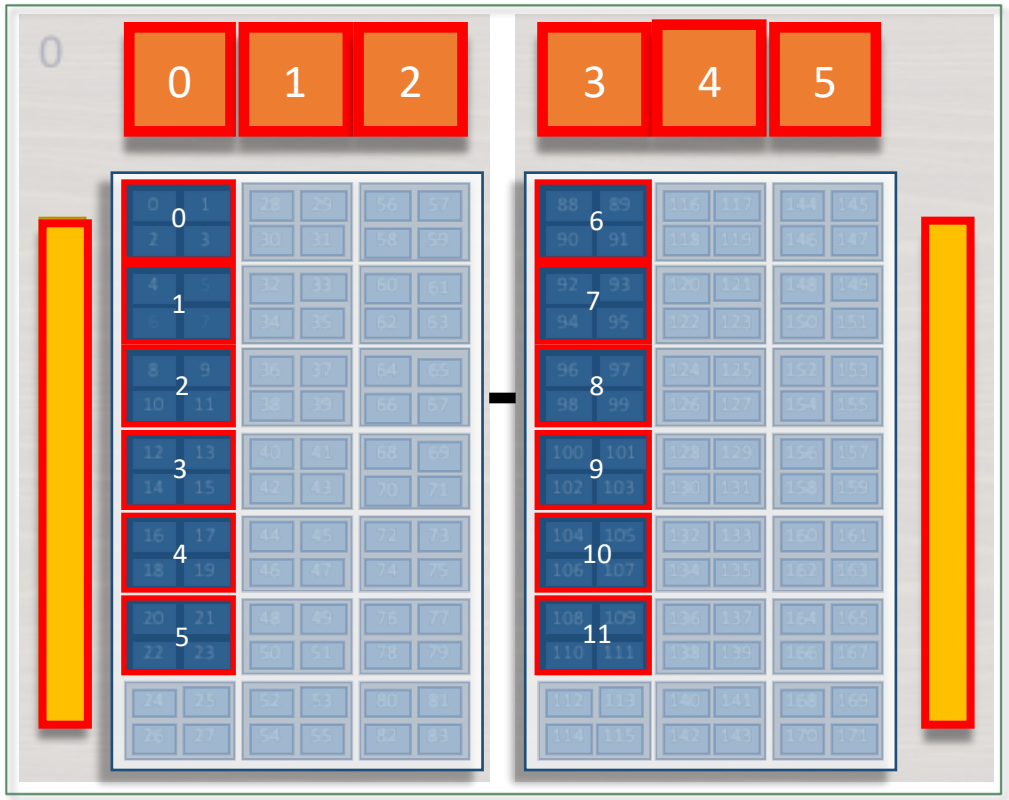
Single resource set per node: 6 GPUs, 12 cores

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 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

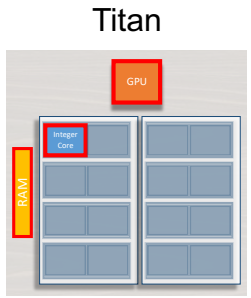
jsrun [-n #Resource Sets] [tasks, threads, and GPUs w/in each Resource Set] program

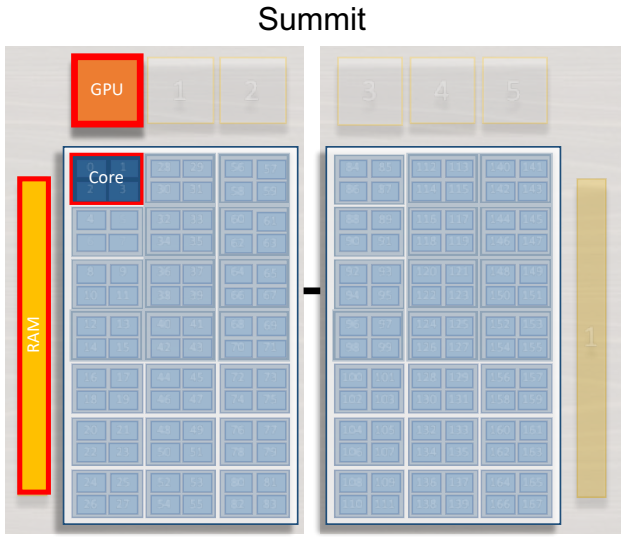
Flags (*long*)	Flags (*short*)	Description
--nrs	**-n**	Number of resource sets
--rs_per_host	-r	Number of resource sets per host (node)
--tasks_per_rs	**-a**	Number of MPI tasks/ranks per resource set
--cpu_per_rs	**-c**	Number of CPUs (cores) per resource set.
--gpu_per_rs	**-g**	Number of GPUs per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

**for additional flags see the jsrun man page*

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



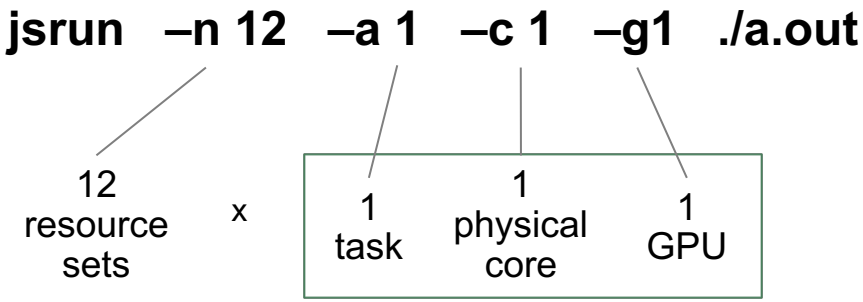


GPUs per Task	MPI Tasks	Threads per Task	aprun	jsrun
1	1	0	aprun –n1	jsrun -n1 -g1 -a1 -c1
1	2	0	aprun –n2	jsrun –n1 -g1 -a2 –c2
1	1	4	aprun –n1 –d4	jsrun -n1 -g1 -a1 -c4 –bpacked:4
1	2	8	aprun –n2 –d8	jsrun -n1 -g1 -a2 –c16 –bpacked:8

Basic jsrun Examples

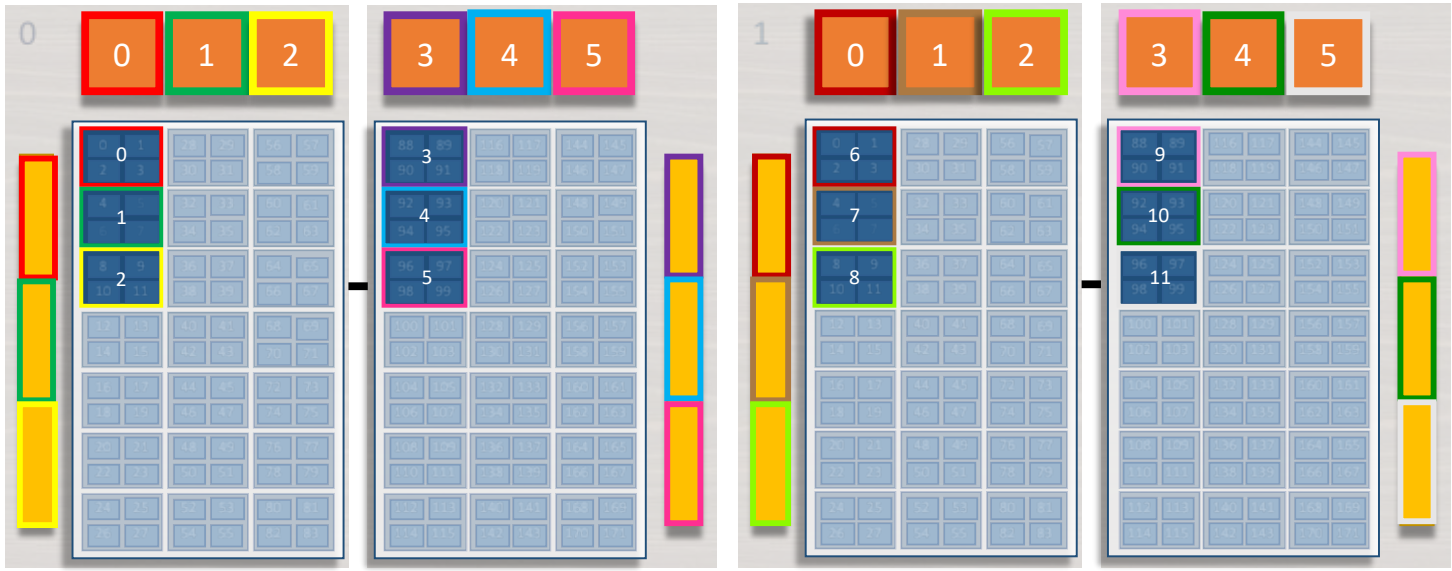
Description	Jsrun command	Layout notes
64 MPI tasks, no GPUs	***jsrun –n 64 ./a.out***	2 nodes: 42 tasks node1, 22 tasks on node2
12 MPI tasks each with access to 1 GPU	***jsrun –n 12 –a 1 –c 1 –g1 –bpacked:1 ./a.out***	2 nodes, 3 tasks per socket
12 MPI tasks each with 4 threads and 1 GPU	***jsrun –n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

12 MPI tasks each with access to 1 GPU



Specify key flags each submission, do not rely on defaults

First RS (red) contains
- task 0
- core 0
- GPU 0



2 Tasks, 1 GPU per RS

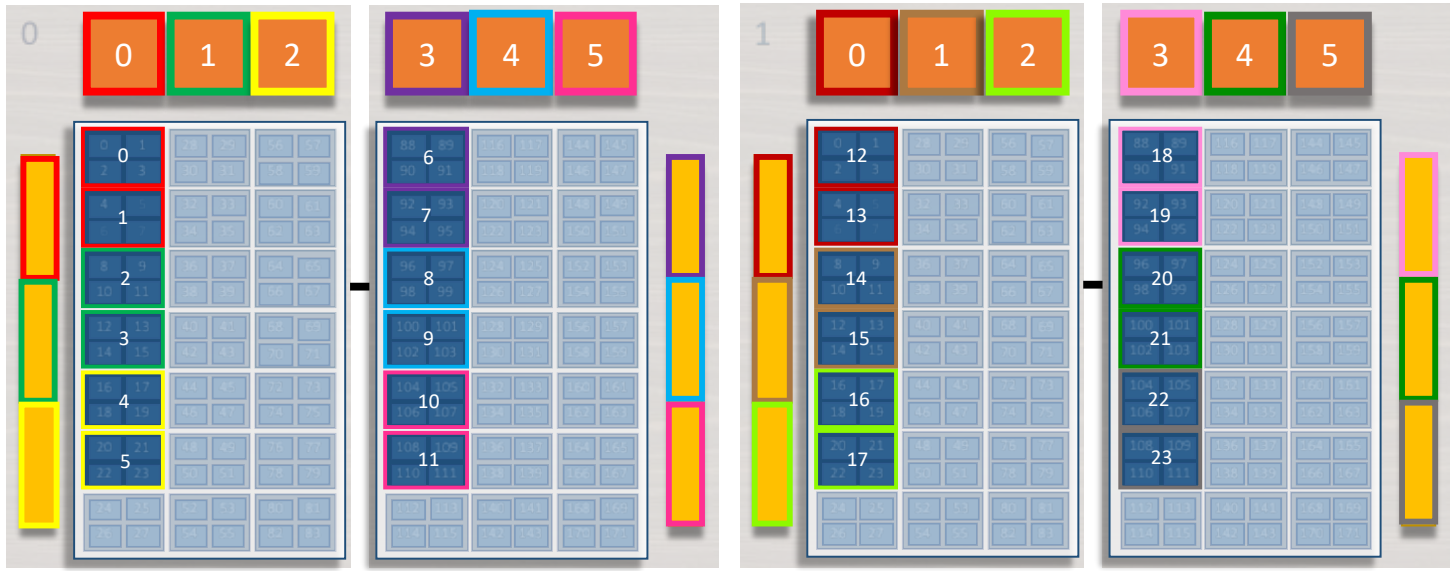
jsrun –n 12 –a 2 –c 2 –g1 –d packed ./a.out

- 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 *(default)*

First RS (red) contains
- 2 tasks (0-1)
- 2 cores (0,4)
- 1 GPU (0)



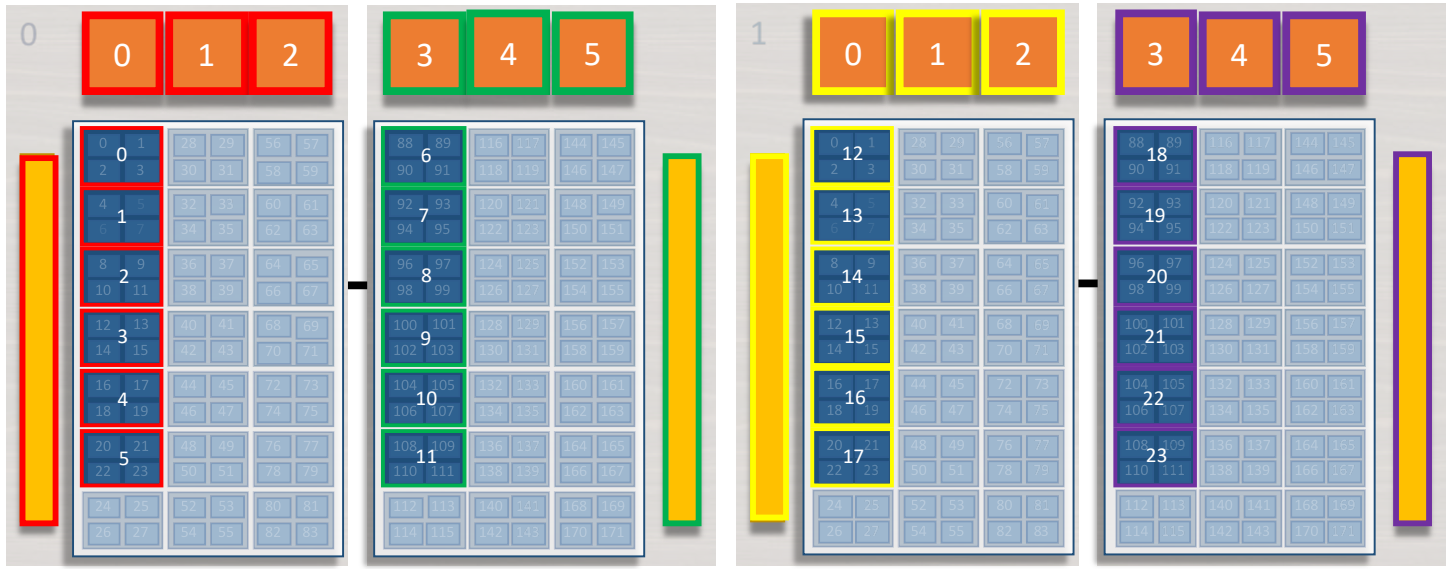
6 MPI Tasks, 3 GPUs per RS

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)



1 MPI Task, 4 Threads, 1 GPU per RS

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

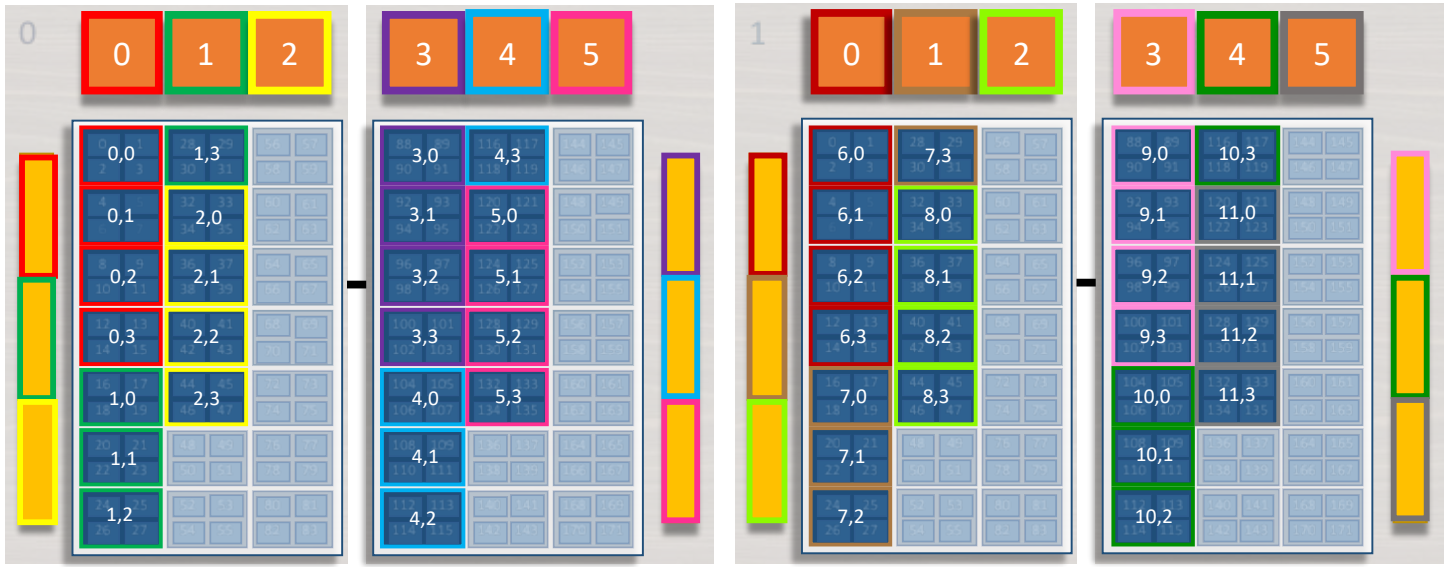
User should set OMP_NUM_THREADS = 4

For rank 0 jsrun will set

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

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
- **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

```
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.

```
summit-batch1> jsrun –n1 -a1 –c2 -bpacked: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.

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)

Viewing jsrun Layout

- Execute code within interactive batch job to view jsrun layout
- Lab maintained example code:
  - www.olcf.ornl.gov/for-users/system-user-guides/summit/running-jobs/#hello_jsrun

```
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.

```
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.

Notice default rank placement order cycles between RS.

```
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
```

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

```
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}
```

Default binding to one physical core

```
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}
```

Binding packed:smt: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}
```

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
```

No visible GPUs

```
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
```

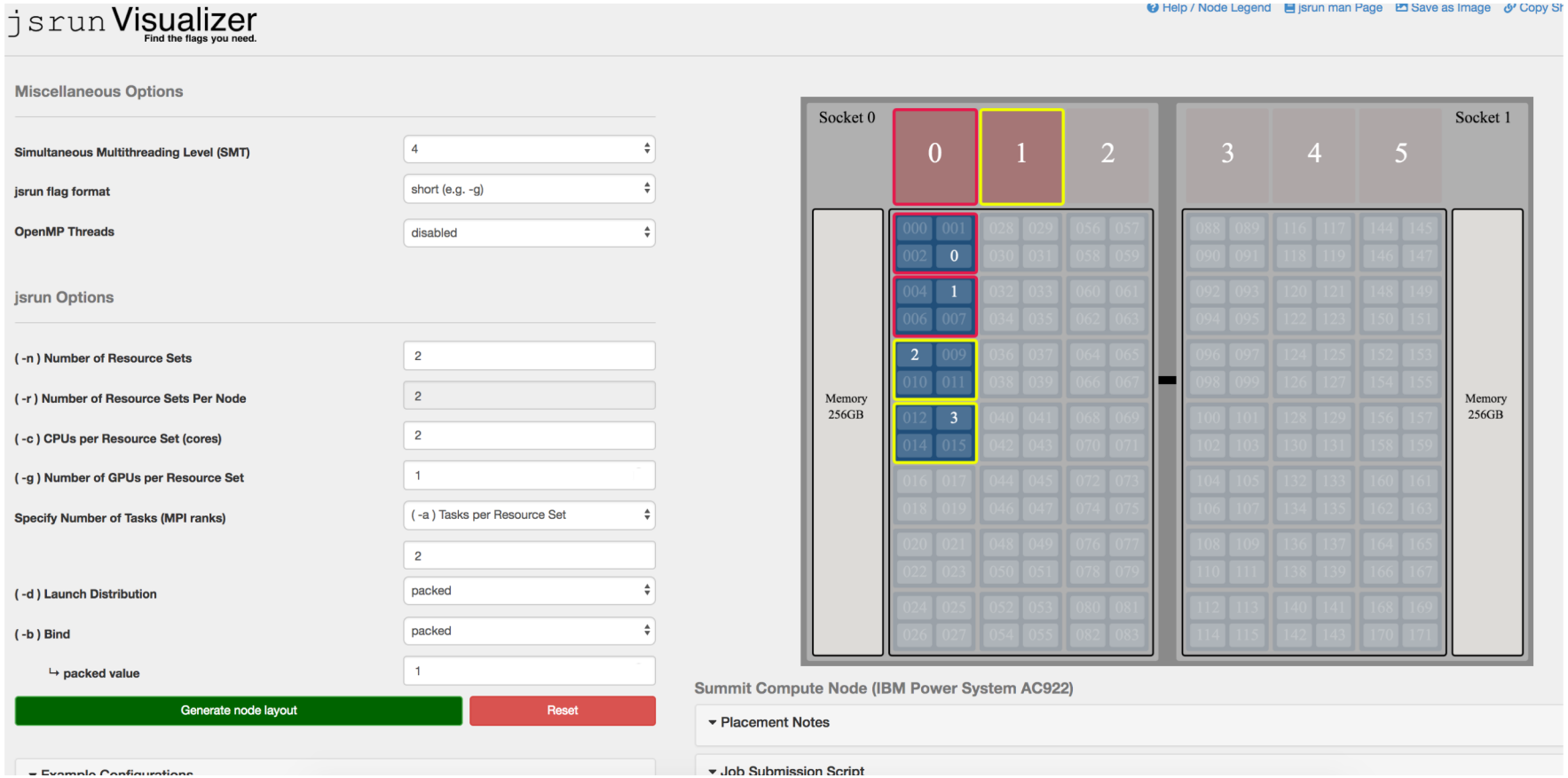
3 visible GPUs

```
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
```

Bus ID shows two GPUs per socket visible

Viewing jsrun Layout (jsrunVisualizer)

- jsrunVisualizer, web based tool to view jsrun layout
- https://jsrunvisualizer.olcf.ornl.gov



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 1023> jslist
      parent                cpus       gpus       exit
  ID      ID      nrs     per RS     per RS     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
```

All GPUs on node, 1 CPU

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

Moving Forward

- PTF11 software stack
  - Bug fixes, features
- New Features
  - Explicit Resource File (--erf_input)
  - Resource set per socket (-K)
- Documentation
  - www.olcf.ornl.gov/for-users/system-user-guides/summit
  - Man pages
    - jsrun, bsub
- Help/Feedback
  - help@olcf.ornl.gov