OAK RIDGE National Laboratory

Using the Summit Supercomputer

Tom Papatheodore – OLCF

Hands-on with the Summit Supercomputer
June 21, 2021

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

U.S. DEPARTMENT OF ENERGY

Outline

- Programming Environment
- Parallel Programming Models
- Resource Scheduler & Parallel Job Launcher

OAK RIDGE National Laboratory

Programming Environment

What is the Programming Environment?

At the highest level, the PE is your shell's build- and run-time environment.

- Compilers, compiler wrappers, tools, scientific libraries, runtimes, etc.
- See output of running `env`

OLCF offers many software packages for users

Managed by users through session environment variables.

- Search paths
  - `PATH, LD_LIBRARY_PATH, LIBRARY_PATH, PKG_CONFIG_PATH, …`
- Program options via environment variables
  - `OMPI_*, CC, FC, …`

Much of the available software cannot coexist simultaneously in your environment.

LMOD Environment Modules

Summit runs LMOD to manage environment complexity

Build- and runtime-environment software managed with LMOD

- https://lmod.readthedocs.io

Usage:

```
$ module -t list                      # list loaded modules
$ module avail                        # show modules that can be loaded given current env
$ module help <package>               # help info for package (if provided)
$ module load <package> <package>…    # add package(s) to environment
$ module unload <package> <package>…  # remove package(s) from environment
$ module reset                        # restore system defaults
$ module restore <collection>         # load a saved collection
$ module spider <package>             # deep search for modules
$ module purge                        # clear all modules from env
```

LMOD Environment Modules – `module avail`

- The **`module avail`** command shows *only what can be loaded given currently loaded packages.*
- Full or partial package names limit output to matches

```
$ module avail
…
----------------- /sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core -----------------
…
   cuda/10.1.243                    (D)      openssl/1.0.2
   cuda/10.2.89                              pango/1.41.0-py3
   cuda/11.0.1                               papi/5.5.1
   cuda/11.0.2                               papi/5.6.0
   cuda/11.0.3                               papi/5.7.0                (D)
   curl/7.63.0                               patchelf/0.9
…
  Where:
   L:  Module is loaded
   D:  Default Module

Use "module spider" to find all possible modules and extensions.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".
```

LMOD Environment Modules – `module spider`

- Use **`module spider`** (instead of **`module avail`**) to search for modules
  - Finds packages that cannot be loaded given current environment
  - Shows requirements needed to make package available

```
$ module spider hdf5
----------------------------------------------------------------------------
  hdf5:
----------------------------------------------------------------------------
     Versions:
        hdf5/1.8.18
        hdf5/1.10.0-patch1
        hdf5/1.10.3
        hdf5/1.10.4
     Other possible modules matches:
        hdf5_perf
----------------------------------------------------------------------------
  To find other possible module matches execute:

      $ module -r spider '.*hdf5.*'
----------------------------------------------------------------------------
  For detailed information about a specific "hdf5" package (including how to load the modules) use the module's full name.
  Note that names that have a trailing (E) are extensions provided by other modules.
  For example:

     $ module spider hdf5/1.10.4
```

Default Modules

- DefApps meta module
  - XL compiler
  - `Spectrum MPI`
  - `HSI – HPSS interface utilities`
  - `Xalt – library usage`
  - `LSF-Tools – Wrapper utility for LSF`
  - `Darshan-Runtime – An IO profiler; unload if using other profilers`

Compiler Environments

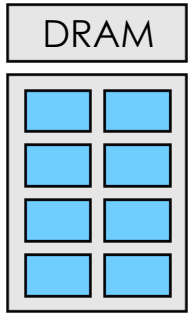
IBM XL (default)	GCC	PGI
xl/16.1.1-8	gcc/9.1.0 (**OpenACC)	pgi/20.4
xl/16.1.1-7	gcc/8.1.1 (**OpenACC)	pgi/20.1
xl/16.1.1-6	gcc/8.1.0 (**OpenACC)	pgi/19.10
xl/16.1.1-5 (D)	gcc/7.4.0	pgi/19.9 (D)
xl/16.1.1-4	gcc/6.4.0 (D)	pgi/18.10
	gcc/5.4.0	pgi/18.7
	gcc/4.8.5	
(OpenMP Offload)	(**OpenACC)	(OpenACC)

OAK RIDGE National Laboratory

Parallel Programming Models

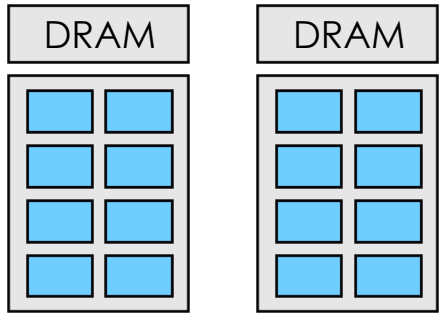
Parallel Programming Models

A = 



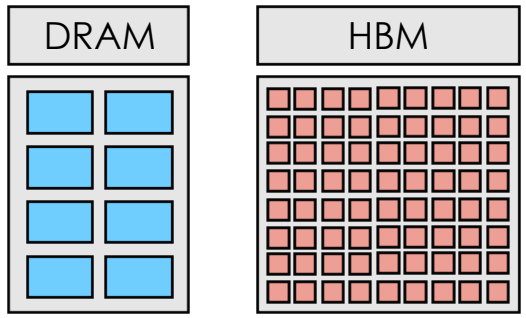
Shared-memory models

- E.g., OpenMP
  - All process threads can access same memory (single-node)



Distributed-memory models

- E.g., Message Passing Interface (MPI)
  - All processes (i.e., MPI ranks) have access to their own memory (multi-node)

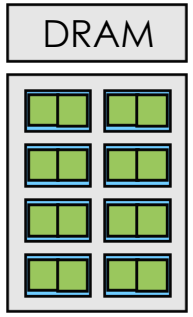


GPU

- CUDA, HIP
- OpenACC, OpenMP offload (directive-based models)
- Kokkos (portability)

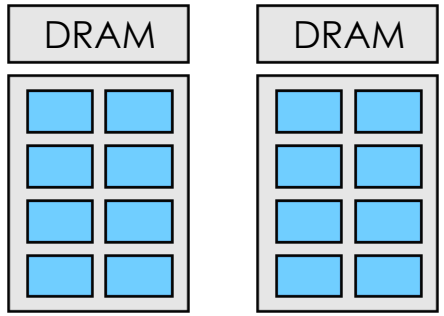
Parallel Programming Models

A = 



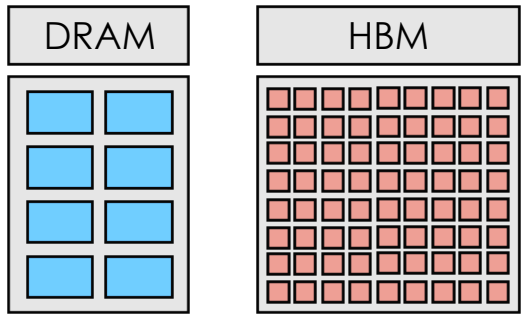
Shared-memory models

- E.g., OpenMP
  - All process threads can access same memory (single-node)



Distributed-memory models

- E.g., Message Passing Interface (MPI)
  - All processes (i.e., MPI ranks) have access to their own memory (multi-node)

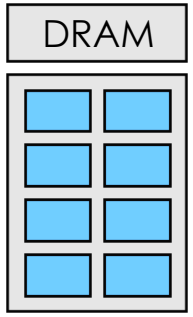


GPU

- CUDA, HIP
- OpenACC, OpenMP offload (directive-based models)
- Kokkos (portability)

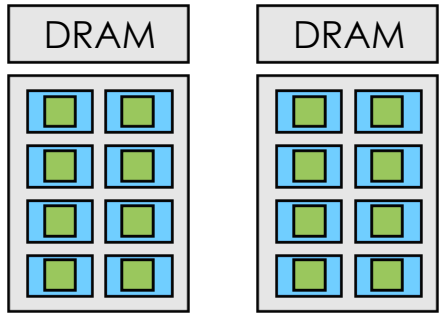
Parallel Programming Models

A = 



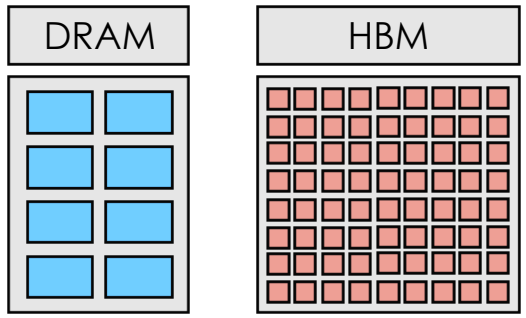
Shared-memory models

- E.g., OpenMP
  - All process threads can access same memory (single-node)



Distributed-memory models

- E.g., Message Passing Interface (MPI)
  - All processes (i.e., MPI ranks) have access to their own memory (multi-node)

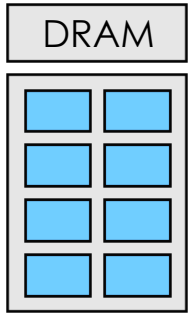


GPU

- CUDA, HIP
- OpenACC, OpenMP offload (directive-based models)
- Kokkos (portability)

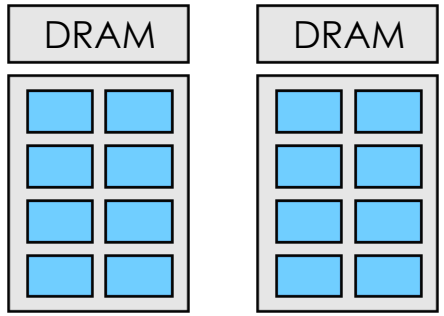
Parallel Programming Models

A = 



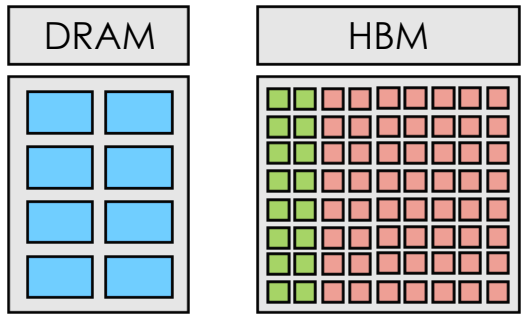
Shared-memory models

- E.g., OpenMP
  - All process threads can access same memory (single-node)



Distributed-memory models

- E.g., Message Passing Interface (MPI)
  - All processes (i.e., MPI ranks) have access to their own memory (multi-node)



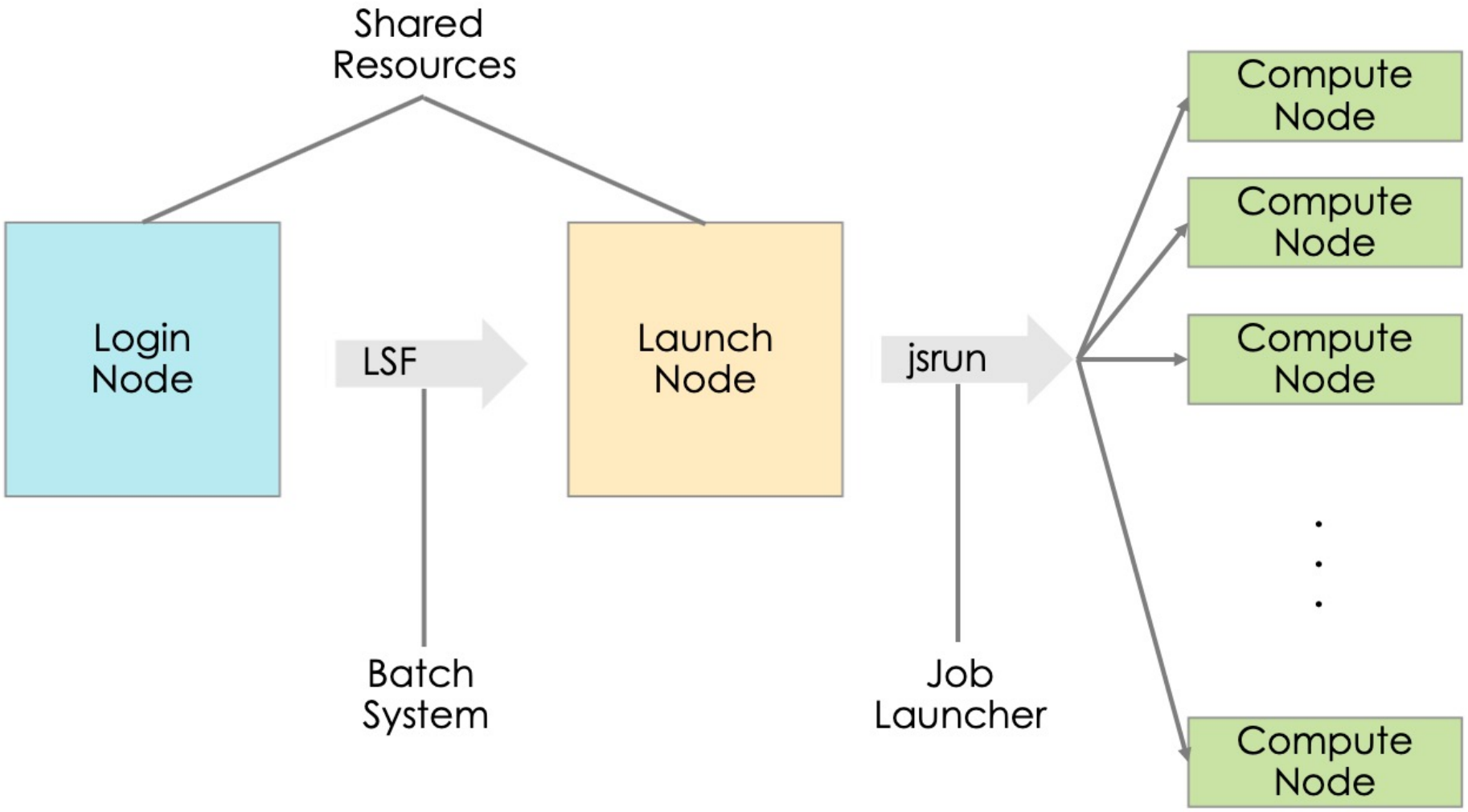
GPU

- CUDA, HIP
- OpenACC, OpenMP offload (directive-based models)
- Kokkos (portability)

OAK RIDGE National Laboratory

Resource Scheduler & Parallel Job Launcher

Login, Launch, and Compute Nodes



Parallel Job Execution

Batch Scheduling System

IBM Load Sharing Facility (LSF)

- Allocates resources
- Batch scheduler
- Similar functionality to PBS/MOAB or Slurm
- Allocates entire nodes (@OLCF)

Parallel Job Launcher

jsrun

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

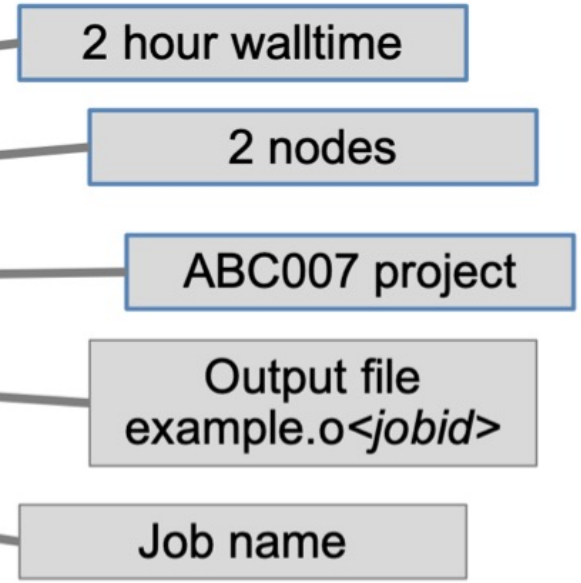
LSF Example Batch Script (non-interactive)

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.

See **`man bsub`** for full options list

Common LSF Commands

Function	LSF Command
Submit a batch script	bsub
Monitor Queue	bjobs / jobstat
Investigate Job	bhist
Alter Queued Job	bmod
Remove Queued Job	bkill
Hold Queued Job	bstop
Release Held Job	bresume

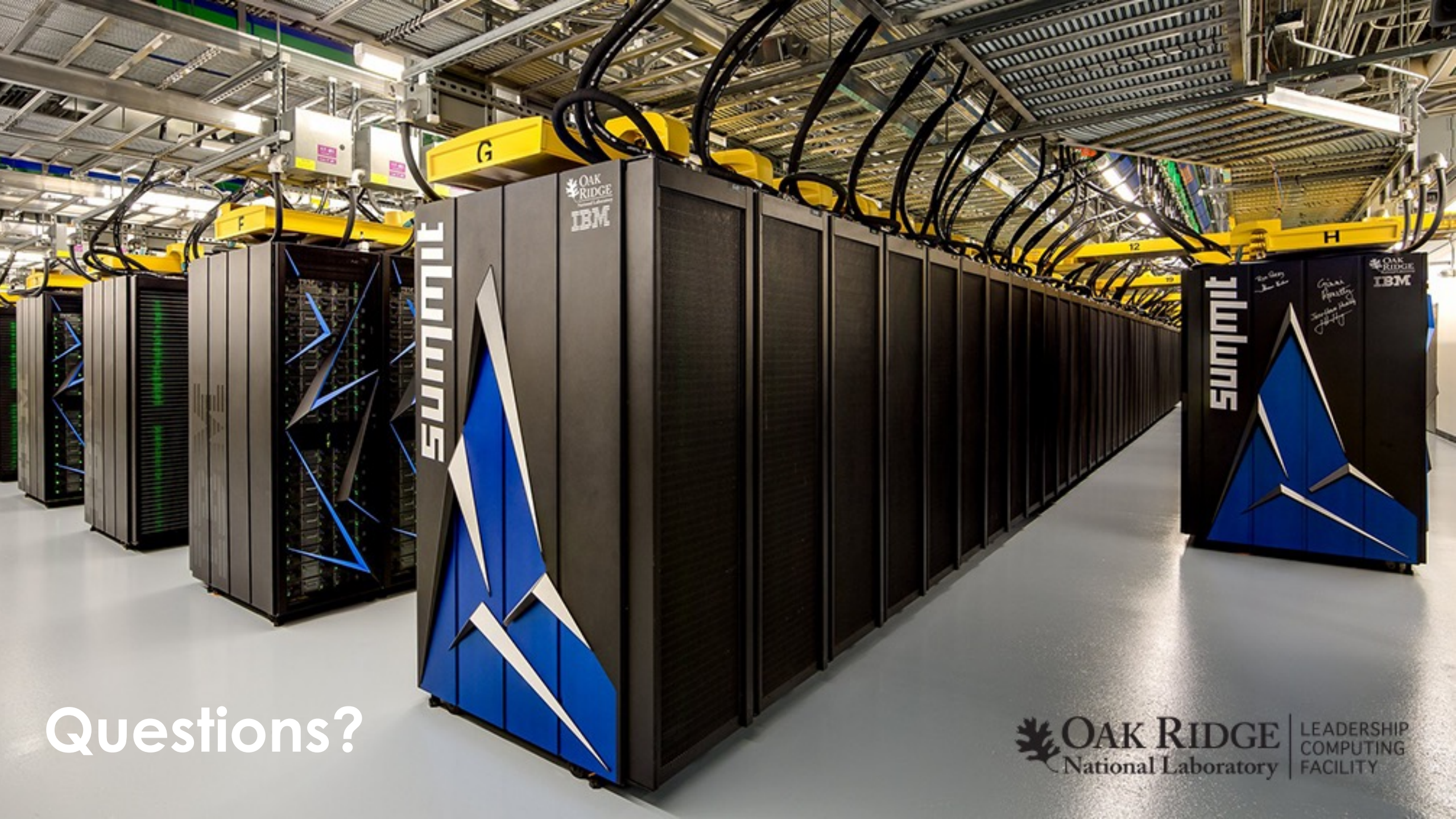
See manual pages for more info

jsrun – Basic Options

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

jsrun Flags		Description	Default Value
Long	Short		
--nrs	-n	Number of RS	All available physical cores
--tasks_per_rs	-a	Number of MPI tasks (ranks) per RS	N/A (total set instead [-p])
--cpu_per_rs	-c	Number of CPUs (physical cores) per RS	1
--gpu_per_rs	-g	Number of GPUs per RS	0
--bind	-b	Number of physical cores allocated per task	packed:1
--rs_per_host	-r	Number of RS per host (node)	N/A
--latency_priority	-l	Controls layout priorities	gpu-cpu,cpu-mem,cpu-cpu
--launch_distribution	-d	Order of tasks started on multiple RS	packed

See `man jsrun` for full list of options



Questions?

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY