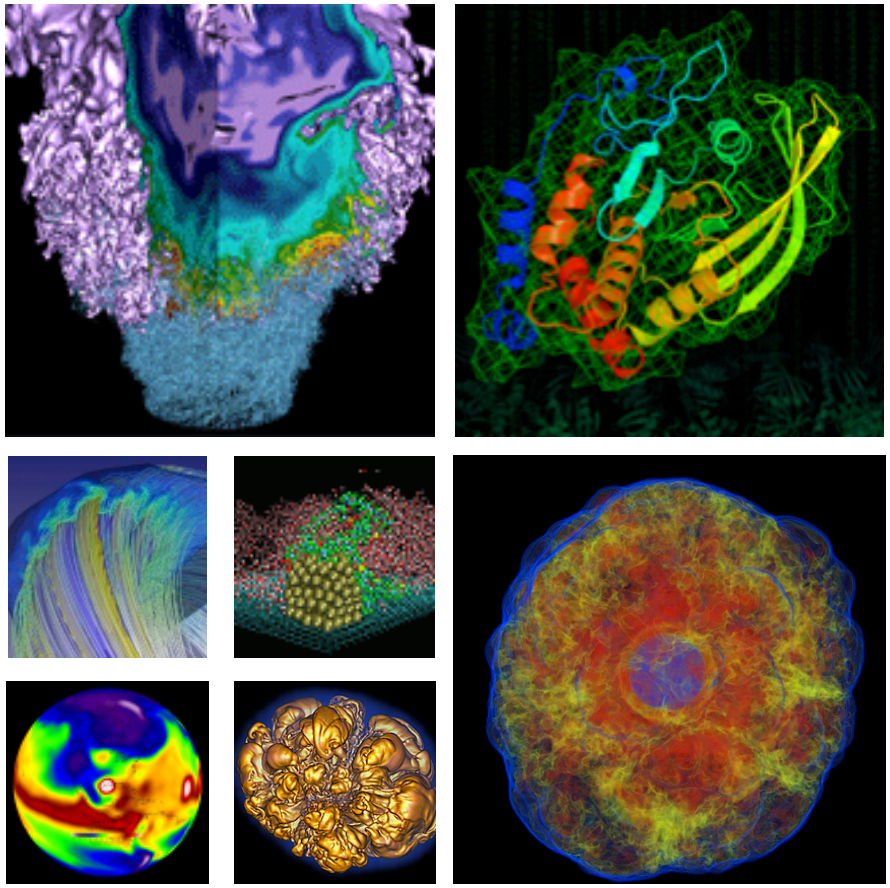
Running Jobs on Cori with SLURM



Helen He
NERSC User Engagement Group

Cori Phase 1 Training
June 14, 2016

NERSC 40 YEARS at the FOREFRONT 1974-2014

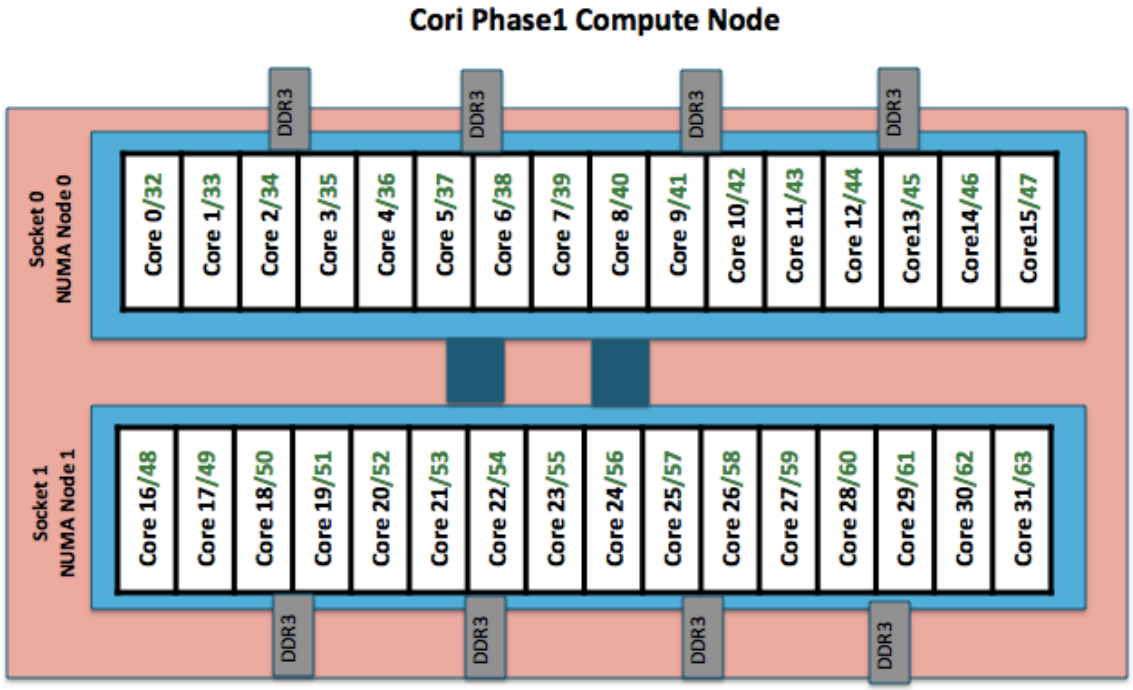
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Cori Phase 1 - Cray XC40

- 52,160 cores, 1,630 nodes
- “Aries” interconnect
- 2 x 16-core Intel ’Haswell' 2.3 GHz processors per node
- 32 processor cores per node, 64 with hyperthreading
- 128 GB of memory per node
- 203 TB of aggregate memory
- 4 GB memory / core for applications
- /scratch disk quota of 20 TB
- 30 PB of /scratch disk
- Choice of full Linux operating system or optimized Linux OS (Cray Linux)
- Intel, Cray, and GNU compilers

Cori Phase 1 Compute Nodes



To obtain processor info:

Get on a compute node:
% salloc -N 1

Then:
% cat /proc/cpuinfo
or
% hwloc-ls

- **Cori Phase 1: NERSC Cray XC40, 1,630 nodes, 52,160 cores**
  - **Each node has 2 Intel Xeon 16-core Haswell processors**
  - **2 NUMA domains per node, 16 cores per NUMA domain. 2 hardware threads per core.**
- **Memory bandwidth is non-homogeneous among NUMA domains**

User Jobs at NERSC

- **Most are parallel jobs (10s to 100,000+ cores)**
- **Also a number of "serial" jobs**
  - Typically "pleasantly parallel" simulation or data analysis
- **Production runs execute in batch mode**
- **Our batch scheduler is SLURM (native)**
- **Debug jobs are supported for up to 30 minutes**
- **Typically run times are a few to 10s of hours**
  - Each machine has different limits
  - Limits are necessary because of MTBF and the need to accommodate 6,000 users' jobs

What is SLURM

- In simple word, SLURM is a workload manager, or a batch scheduler
- SLURM stands for Simple Linux Utility for Resource Management
- SLURM unites the cluster resource management (such as Torque) and job scheduling (such as Moab) into one system. Avoids inter-tool complexity.
- SLURM was used in 6 of the top 10 computers (June 2015 TOP 500 list), including the #1 system, Tianhe-2, with over 3M cores
- More Cray sites adopt SLURM: NERSC, CSCS, KAUST, TACC, etc.

Advantages of Using SLURM

- **Fully open source**
- **SLURM is extensible (plugin architecture)**
- **Low latency scheduling. Highly scalable**
- **Integrated "serial" or "shared" queue**
- **Integrated Burst Buffer support**
- **Good memory management**
- **Built-in accounting and database support**
- **"Native" SLURM runs without Cray ALPS** (Application Level Placement Scheduler)
  - Batch script runs on the head compute node directly
  - Easier to use. Less chance for contention compared to shared MOM node

Login Nodes and Compute Nodes

Each machine has 2 types of nodes visible to users

- **Login nodes (external)**
  - Edit files, compile codes, submit batch jobs, etc.
  - Run short, serial utilities and applications
- **Compute nodes**
  - Execute your application
  - Dedicated resources for your job

Submitting Batch Jobs

- **To run a batch job on the compute nodes you must write a "batch script" that contains**
  - Directives to allow the system to schedule your job
  - An `srun` command that launches your parallel executable
- **Submit the job to the queuing system with the sbatch command**
  - `% sbatch my_batch_script`

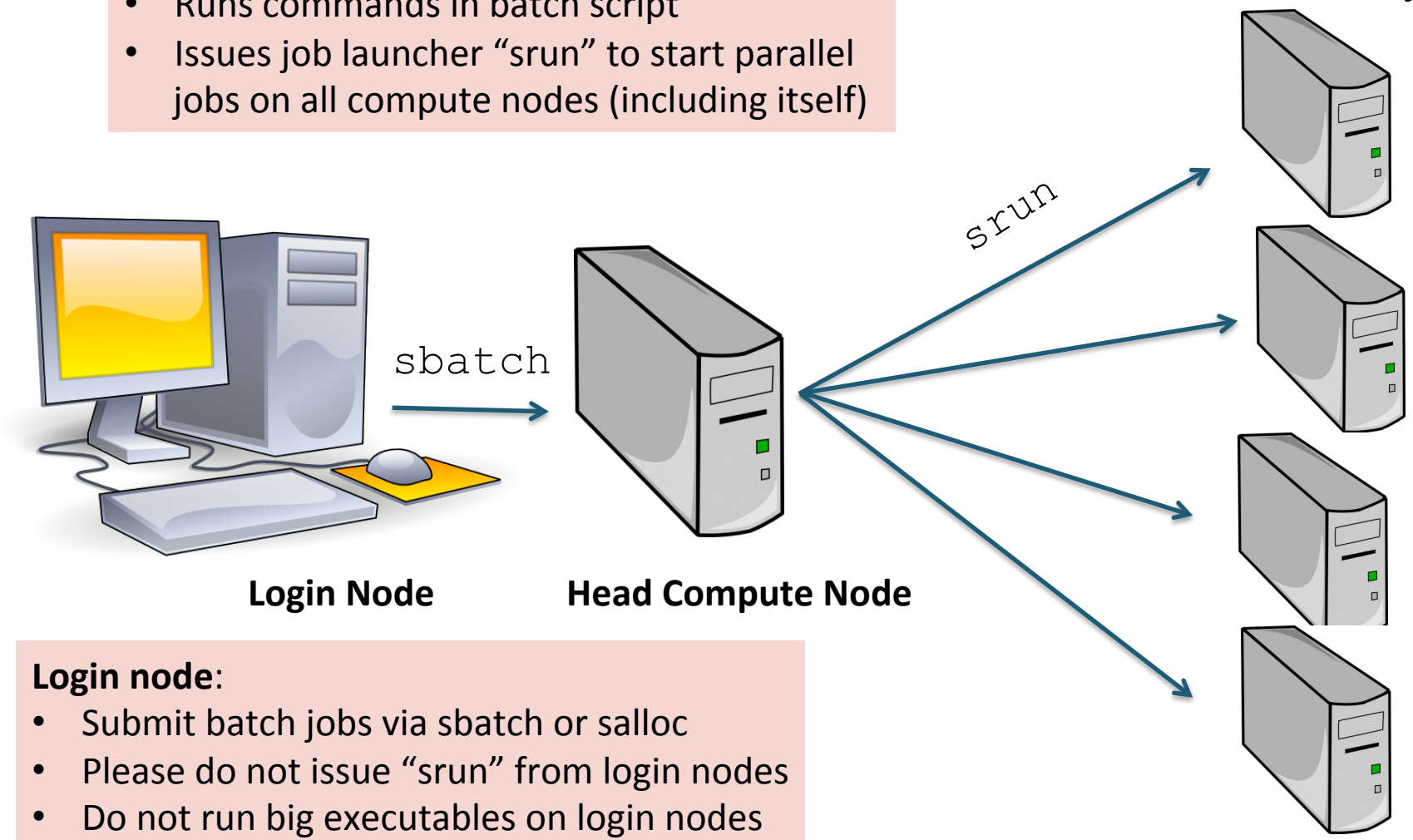
Interactive Parallel Jobs

- **You can run small parallel jobs interactively for up to 30 minutes**

```
login% salloc -N 2 -p debug -t 15:00
[wait for job to start]
compute% srun -n 64 ./mycode.exe
```

Launching Parallel Jobs with SLURM

Head compute node:
- Runs commands in batch script
- Issues job launcher "srun" to start parallel jobs on all compute nodes (including itself)



Login node:
- Submit batch jobs via sbatch or salloc
- Please do not issue "srun" from login nodes
- Do not run big executables on login nodes

Sample Cori Batch Script - MPI

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -n 1280
#SBATCH -J myjob
#SBATCH –L scratch

export OMP_NUM_THREADS=1
srun -n 1280 ./mycode.exe
```

Sample Cori Batch Script - MPI

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -n 1280
#SBATCH -J myjob
#SBATCH –L scratch

export OMP_NUM_THREADS=1
srun -n 1280 ./mycode.exe
```

- Need to specify which shell to use for batch script
- Use “-l” as login shell is optional
- Environment is automatically imported

Sample Cori Batch Script - MPI

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -n 1280
#SBATCH -J myjob
#SBATCH –L scratch

export OMP_NUM_THREADS=1
srun -n 1280 ./mycode.exe
```

Job directives: instructions for the batch system

- Submission partition (default is “debug”)
- How many compute nodes to reserve for your job
- How long to reserve those nodes
- More optional SBATCH keywords

Sample Cori Batch Script - MPI

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -n 1280
#SBATCH -J myjob
#SBATCH –L scratch

export OMP_NUM_THREADS=1
srun -n 1280 ./mycode.exe
```

SBATCH optional keywords:

- How many instances of applications to launch (# of MPI tasks)
- What is my job name
- What file system licenses are used

Sample Cori Batch Script - MPI

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -n 1280
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 ./mycode.exe
```

- By default, hyperthreading is on. SLURM sees 2 threads are available for each of the 32 physical CPUs on the node
- No need to set this if your application programming model is pure MPI.
- If your code is hybrid MPI/OpenMP, set this value to 1 to run in pure MPI mode

Sample Cori Batch Script - MPI

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -n 1280
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 ./mycode.exe
```

"srun" command launches parallel executables on the compute nodes

- srun flags overwrite SBATCH keywords
- No need to repeat flags in srun command if already defined in SBATCH keywords. (e.g. "srun ./my_executable" will also do in above example)

Sample Cori Batch Script - MPI

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -n 1280
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 ./mycode.exe
```

- There are 64 logical CPUs on each node
- With 40 nodes, using hyperthreading, up to 40*64=2,560 MPI tasks can be launched: “srun -n 2560 ./mycode.exe” is OK

Sample Batch Script - Hybrid MPI/OpenMP

```
#!/bin/bash -l
#SBATCH -p regular
#SBATCH -N 40
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
srun -n 160 -c 8 ./mycode.exe
```

- srun does most of optimal process and thread binding automatically. Only flags such as “-n” “-c”, along with OMP_NUM_THREADS are needed for most applications
- Hyperthreading is enabled by default. Jobs requesting more than 32 cores (MPI tasks * OpenMP threads) per node will use hyperthreads automatically

Long and Short Commands Options

Long command options

```
#!/bin/bash -l

#SBATCH --partition=regular
#SBATCH --job-name=test
#SBATCH --account=mpccc
#SBATCH --nodes=2
#SBATCH --time=00:30:00
#SBATCH --license=scratch

srun -n 64 ./mycode.exe
```

Short command options

```
#!/bin/bash -l

#SBATCH -p regular
#SBATCH -J test
#SBATCH -A mpccc
#SBATCH -N 2
#SBATCH -t 00:30:00
#SBATCH -L scratch

srun -n 64 ./mycode.exe
```

Running Multiple Parallel Jobs Sequentially

```
#!/bin/bash -l

#SBATCH -p regular
#SBATCH -N 4
#SBATCH -t 12:00:00
#SBATCH -L project,cscratch1

srun -n 128 ./a.out
srun -n 128 ./b.out
srun -n 128 ./c.out
```

- Need to request max number of nodes needed by each srun

Running Multiple Parallel Jobs Simultaneously

```
#!/bin/bash -l

#SBATCH -p regular
#SBATCH -N 8
#SBATCH -t 12:00:00
#SBATCH -L cscratch1

srun -n 44 -N 2 ./a.out &
srun -n 108 -N 4 ./b.out &
srun -n 40 -N 2 ./c.out &
wait
```

- Need to request total number of nodes needed by all sruns
- Notice the “&” and “wait” above
- It is best if each srun takes roughly same amount of time to complete, otherwise waste allocation

Running MPMD Jobs

```
# Config file:

% cat mpmd.conf
0-35 ./a.out
36-95 ./b.out
```

```
# Batch Script:

#!/bin/bash -l

#SBATCH -p regular
#SBATCH -N 4
#SBATCH -n 96      # total of 96 tasks
#SBATCH -t 02:00:00
#SBATCH –L SCRATCH

srun --multi-prog ./mpmd.conf
```

- Two executables will share one MPI_COMM_WORLD
- Request total number of nodes needed for a.out and b.out

Job Steps and Dependency jobs

- **Use job dependency features to chain jobs that have dependency**

```
cori% sbatch job1
Submitted batch job 5547

cori06% sbatch --dependency=afterok:5547 job2
or
cori06% sbatch --dependency=afterany:5547 job2
```

```
cori06% sbatch job1
submitted batch job 5547

cori06% cat job2
#!/bin/bash -l

#SBATCH -p regular
#SBATCH -N 1
#SBATCH -t 00:30:00
#SBATCH -d afterok:5547

cd $SLURM_SUBMIT_DIR
srun -n 32 ./a.out

cori06% sbatch job2
```

“shared” Partition on Cori

- Users see many jobs in “shared”, appears to use 1 node per job (displayed with the queue monitoring scripts), actually NOT.
- Serial jobs or small parallel jobs are shared on these nodes
- 40 nodes are set aside for the “shared” jobs
- “shared” jobs do not run on other nodes currently (may change in the future)
- High submit limits (10,000) and run limits (1,000)
- Jobs are getting very good throughput
- “shared” jobs are not charged by entire node, but by actual physical cores used

Running Serial Jobs

- **The "shared" partition allows multiple executables from different users to share a node**
- **Each serial job run on a single core of a "shared" node**
- **Up to 32 jobs from different users depending on their memory requirements**

```
#SBATCH -p shared
#SBATCH -t 1:00:00
#SBATCH --mem=4GB
#SBATCH -J my_job
./mycode.exe
```

- Do not specify #SBATCH -N"
- Default "#SBATCH -n" is 1
- Default memory is 1,952 MB
- Use -n or --mem to request more slots for larger memory
- Do not use "srun" for serial executable (reduces overhead)

- **Small parallel job that use less than a full node can also run in the "shared" partition**

“realtime” Partition

- Special permission to use “realtime” for real-time need of data intensive workflows
- Highest priority for “realtime” jobs so they start almost immediately. Could be disruptive to overall queue scheduling.
- “realtime” jobs can run in “shared” or “exclusive” mode for node usage
- Nodes are set aside for the “realtime” jobs (currently)
- “realtime” jobs can run on other nodes

Job Arrays

- **Use Job Arrays for submitting and managing collections of similar jobs**
  - Better managing jobs, not necessary faster turnaround
  - Each array task considered a single job for scheduling, submit/run limits
  - SLURM_ARRAY_JOB_ID set to the first job ID of the array
    SLURM_ARRAY_TASK_ID set to the job array index value

```
# Submit a job array with index values between 0 and 31
% sbatch --array=0-31 -N 1
# Submit a job array with index values of 1, 3, 5 and 7
% sbatch --array=1,3,5,7 -N 1 -n2
# Submit a job array with index values between 1 and 7, with a step size of 2
% sbatch --array=1-7:2 -N 1 -p regular
# submit a job array with index values between 1 and 7, and limit max running jobs to 2
% sbatch --array=1:7%2 -N 1 -p regular
```

stderr/stdout

- **By default, while your job is running, the slurm-$SLURM-JOBID.out is generated and contains both stderr and stdout**
- **stdout is buffered in segments of 8KB size**
- **stderr is not buffered**
- **Can rename via "#SBATCH -o" (for stdout) and "#SBATCH -e" (for stderr) flags**
- **Can also redirect from srun commands**
  - srun -n 48 ./a.out >& my_output_file (for csh/tcsh)
  - Srun –n 48 ./a.out > my_output_file 2>&1 (for bash)
- **srun "-u" option disables stdout buffer**
  - Not recommended to use as default since it slows down application significantly
  - However this help debugging to get complete output before job termination

Cluster Compatibility Mode (CCM) Applications

- Certain 3rd part application need ssh to other compute nodes from the head compute node
- The CCM mode allows 3rd party applications to run on the compute nodes without rebuild
- The "#SBATCH --ccm" option will setup the necessary environment to support CCM applications
- No separate CCM queue, CCM jobs can run in any queue now
- Since sbatch or salloc lands on a compute node, applications such as "matlab" can be launched directly without CCM support

Running Jobs Built with Intel MPI

```
#!/bin/bash -l

#SBATCH -p regular
#SBATCH -N 8
#SBATCH -t 03:00:00
#SBATCH -L project.SCRATCH

module load impi
mpiicc -openmp -o mycode.exe mycode.c

export OMP_NUM_THREADS=8
export I_MPI_PMI_LIBRARY=/opt/slurm/default/lib/pmi/libpmi.so
srun -n 32 -c 8 ./mycode.exe
```

Which File Systems to Use

- **Do not run from $HOME**
  - Meant for permanent space for many small files (source, small input, etc).
  - Performance not optimized for large IO
  - Running large IO jobs from $HOME can cause delayed interactive response time for other users
- **Use $SCRATCH (Lustre file system) or project directory to run your jobs, for better IO performance and larger space quota**
  - $SCRATCH: default 20 TB quota, 10M inodes. Files not accessed for 12 weeks are subject to be purged. Back up important files.
  - /global/project/projectdirs/*your_repo:* default 4 TB quota, 1M inodes. Not purged.

File System Licenses

- **Use "#SBATCH -L" or "#SBATCH --license=" to specify file systems needed**
- **Jobs that use file system licenses will not start if there is an issue with that particular file system**
  - Protects jobs from failing with file system issues
  - Allows selective maintenance on file systems
- **Example: #SBATCH -L scratch1,project**
- **"SCRATCH" can be used as short hand for your default scratch file system on Edison or Cori**
- **Currently optional, but strongly encouraged**
- **May be enforced in the near future**

More SBATCH Options

- **Which QOS to use: normal (default), premium, low**
  - #SBATCH --qos=premium
- **What's my stdout file name**
  - #SBATCH -o *my_stdout_name*
- **Which account to charge**
  - #SBATCH -A *my_repo*
- **When to send email**
  - #SBATCH --mail-type=BEGIN,END,FAIL
- **…**

1-node "Long" Jobs Up to 96 hrs

- Run in "regular" partition, no "premium" or "low" QOS priority
- % salloc -N 1 -p regular -t 96:00:00 -L SCRATCH
- Can only use a single node
- A user can have up to 4 long jobs running simultaneously
- A user can submit a maximum of 10 long jobs at a time
- A maximum of 10 nodes can be occupied by long jobs from all users
- Be aware of jobs won't start if time left before a system maintenance is less than 96 hrs

Recap: Running Jobs with SLURM

- Use “sbatch” to submit batch script or “salloc” to request interactive batch session.
- Use “srun” to launch parallel jobs
- Most SLURM command options have two formats (long and short)
- Need to specify which shell to use for batch script.
- Environment is automatically imported
- Lands in the submit directory
- Batch script runs on the head compute node
- No need to repeat flags in the srun command if already defined in SBATCH keywords.
- srun flags overwrite SBATCH keywords

Some SLURM Gotchas

- **Hyperthreading is enabled by default.**
  - SLURM sees 64 CPUs per node (each Cori node has 32 physical cores, total of 64 logical cores per node.)
  - srun will decide whether to use logical cores. (see last bullet)
- **Need to set OMP_NUM_THREADS=1 explicitly to run in pure MPI mode (with hybrid MPI/OpenMP applications built with OpenMP compiler flag enabled)**
- **Always use "#SBATCH -N" to request number of nodes. If asking nodes with "#SBATCH -n" only (for num_MPI_tasks), you may get half the #nodes desired.**
- **Automatic process and thread affinity is good. Hyperthreading will not be used if num_MPI_tasks * num_OpenMP_threads_per_node is <= 32. Can explore with advanced settings for more complicated binding options.**

Two SLURM Schedulers are in Work

- **Instant Scheduler (event triggered)**
  - Performs a quick and simple scheduling attempt at events such as job submission or completion and configuration changes.
- **Backfill Scheduler (at set intervals)**
  - Considers pending jobs in priority order, determining when and where each will start, taking into consideration the possibility of job preemption, gang scheduling, generic resource (GRES) requirements, memory requirements, etc.
  - If the job under consideration can start immediately without impacting the expected start time of any higher priority job, then it does so.

SLURM User Commands

- **sbatch:** submit a batch script
- **salloc:** request nodes for an interactive batch session
- **srun:** launch parallel jobs
- **scancel:** delete a batch job
- **sqs**: NERSC custom queue display with job priority ranking info
- **squeue:** display info about jobs in the queue
- **sinfo:** view SLURM configuration about nodes and partitions
- **scontrol**: view and modify SLURM configuration and job state
- **sacct:** display accounting data for jobs and job steps
- https://www.nersc.gov/users/computational-systems/cori/running-jobs/monitoring-jobs/

sqs: NERSC Custom Queue Monitoring Script

- **Provides two columns of ranking values to give users more perspective of their jobs in queue.**
  - Column RANK_BF shows the ranking using the best estimated start time (if available) at a backfill scheduling cycle, so the ranking is dynamic and changes frequently along with the changes in the queued jobs.
  - Column RANK_P shows the ranking with absolute priority value, which is a function of partition QOS, job wait time, and fair share. Jobs with higher priority won't necessarily run earlier due to various run limits, total node limits, and backfill depth we have set.

sqs Example Commands

% sqs (show user's own jobs)

% sqs -a (shows all jobs)

% sqs -a -p debug (shows only debug jobs)

% sqs -a -nr -np shared (no running jobs, no shared jobs)

% sqs -w (shows all my jobs in wide format with more info)

% sqs -s (short summary of queued jobs)

See man page or use "sqs --help" for more options.

squeue Example Commands

% squeue -u <username>

% squeue -j <jobid> --start

% squeue -o "%.18i %.3t %.10r %.10u %.12j %.8D %.10M %.10l %.20V %.12P %.20S %.15Q"

JOBID ST REASON USER NAME NODES TIME TIME_LIMIT SUBMIT_TIME PARTITION START_TIME PRIORITY

See man page or "squeue --help" for more options.

sinfo Example Commands

% sinfo -s

```
PARTITION AVAIL  TIMELIMIT   NODES(A/I/O/T)  NODELIST
debug*       up      30:00  5357/154/8/5519  nid[00008-00063,00072-00127,…]
regular      up 4-00:00:00  5110/147/6/5263  nid[00296-00323,00328-00383,…]
regularx     up 2-00:00:00  5357/154/8/5519  nid[00008-00063,00072-00127,…]
realtime     up   12:00:00  5420/158/8/5586  nid[00008-00063,00072-00127,…]
shared       up 2-00:00:00       55/0/0/55  nid[06089-06143]
```

% sinfo -p debug

```
PARTITION AVAIL JOB_SIZE  TIMELIMIT   CPUS  S:C:T   NODES STATE      NODELIST
debug*    up    1-infini     30:00     48 2:12:2      6 down*      nid[01343,02062,02137,03132,03150,03307]
debug*    up    1-infini     30:00     48 2:12:2      2 drained    nid[00008-00009]
debug*    up    1-infini     30:00     48 2:12:2      2 reserved   nid[00010-00011]
debug*    up    1-infini     30:00     48 2:12:2      2 mixed      nid[00077,00090]
debug*    up    1-infini     30:00     48 2:12:2   5349 allocated  nid[00012-00063,00072-00076, --]
debug*    up    1-infini     30:00     48 2:12:2    158 idle       nid[00083-00086,00091-00094, …]
```

See man page or “sinfo --help” for more options.

scontrol Example commands

% scontrol show partition <partition>

% scontrol show job <jobid>

% scontrol hold <jobid>

% scontrol release <jobid>

% scontrol update job <jobid> timelimit= 24:00:00

% scontrol update job <jobid> qos=premium

See man page or “scontrol --help” for more options.

sacct Example Commands

% sacct -u <username> --starttime=01/12/16T00:01 --endtime=01/15/16T12:00 -o jobid,elapsed,nnodes,start,end,submit

% sacct -a --starttime=01/12/16T00:01 --endtime=01/15/16T12:00 -o User,JobID,NNodes,State,Start,End,TimeLimit,Elapsed,ExitCode,DerivedExitcode,Comment

% sacct -u <username> -j <jobid> -o jobid,elapsed,nnodes,nodelist

See man page or "sacct --help" for more options.

Edison Queue Policy (as of 06/10/2016)

Specify these partitions with `#SBATCH -q partition_name`

Specify these QOS with `#SBATCH --qos=premium`

These limits are per user per partition/QOS limits

Partition	Nodes	Physical Cores	Max Wallclock	QOS[1]	Run Limit	Submit Limit	Relative Priority	Charge Factor[2]
debug[3]	1-512	1-12,288	30 mins	-	2	10	3	2
	1-15	1-360	96 hrs	--			5	2
regular	16-682	361-16,368	36 hrs	normal	24	100	5	2
				premium	8	20	2	4
				low	24	100	6	1
				scavenger[4]	8	100	7	0
	683-5462	16,369-130,181	36 hrs	normal	8	100	4	1.2
				premium	2	20	2	2.4
				low	8	100	6	0.6
				scavenger	8	100	7	0
shared[5]	1	1-24	48 hrs	normal	1,000	10,000	5	2 x (no. of cores used)
realtime[6]	custom	custom	custom	custom	custom	custom	1 (special permission)	--
xfer[7]			48 hrs	-	8	-	-	0

Jobs with insufficient allocations to run are directed to "scanvenger"

Cori Queue Policy (as of 06/10/2016)

Partition	Nodes	Physical Cores	Max Walltime per Job	QOS	Max Number of Running Jobs	Max Total Num Nodes per User for Running Jobs	Number of Jobs per User Submit Limit	Relative Priority	Charge Factor
debug	1-64	1-2,048	30 min	normal	1	64	5	3	2.5
regular	1	1-32	96 hrs	--	4	4	10	4	2.5
	1-2	1-64	48 hrs	normal	50	100	200	4	2.5
				premium	10	100	40	2	5.0
				low	50	100	200	5	1.25
				scavenger	10	100	40	6	0
	3-512	65-16,384	36 hrs	normal	10	512	50	4	2.5
				premium	2	512	10	2	5.0
				low	10	512	50	5	1.25
				scavenger	2	512	10	6	0
	513-1,420	16,385-45,440	12 hrs	normal	1	1,420	4	4	2.5
				premium	1	1,420	2	2	5.0
				low	1	1,420	4	5	1.25
				scavenger	1	1,420	2	6	0.0
shared	1	1-16	48 hrs	normal	1000	--	10,000	4	2.5/32
realtime	custom	custom	custom	custom	custom	--	1	1 (special permission)	--
xfer	1	1	12 hrs	--	--	--	1	--	0

Large user limits

serial workload

realtime workflow

Cori Phase 1 Data Features Implemented in SLURM

- **Cori Phase 1 also known as the "Cori Data Partition"**
- **Designed to accelerate data-intensive applications, with high throughput and "real time" need.**
  - "shared" partition. Multiple jobs on the same node. Larger submit and run limits. 40 nodes set aside
  - The 1-2 node bin in the "regular" for high throughput jobs. Large submit and run limits.
  - "realtime" partition for jobs requiring real time data analysis. Highest queue priority. Special permission only.
  - Internal sshd (CCM mode) in any queue
  - Large number of login/interactive nodes to support applications with advanced workflows
  - "burst buffer" usage integrated in SLURM, in early user period.
  - Encourage users to run jobs using 683+ nodes on Edison with queue priority boost and 40% charging discount there.

Charge Factors & Discounts

- **Each machine has a "machine charge factor" (MCF) that multiplies the "raw hours" used**
  - Edison MCF = 2.0
  - Cori MCF = 2.5
- **Each QOS has a "QOS charge factor" (QCF)**
  - premium QCF = 2.0
  - normal QCF = 1.0 (default)
  - low QCF = 0.5
  - scavenger QCF = 0
- **On Edison:**
  - Jobs requesting 683 or more nodes get a 40% discount

How Your Jobs Are Charged

- **Your repository is charged for each node your job was allocated for the entire duration of your job.**
  - The minimum allocatable unit is a node (*except for the "shared" partition on Cori*). Edison have 24 cores/node and Cori has 32 cores/node.

MPP hours = (# nodes) * (# cores / node) * (walltime used) * (QCF) * (MCF)

  - Example: 4 Cori nodes for 1 hour with "premium" QOS
    MPP hours = (4) * (32) * (1 hour) * (2) * (2.5) = 640 MPP hours
  - "shared" jobs are charged with physical CPUs used instead of entire node.
- **If you have access to multiple repos, pick which one to charge in your batch script**

```
#SBATCH -A repo_name
```

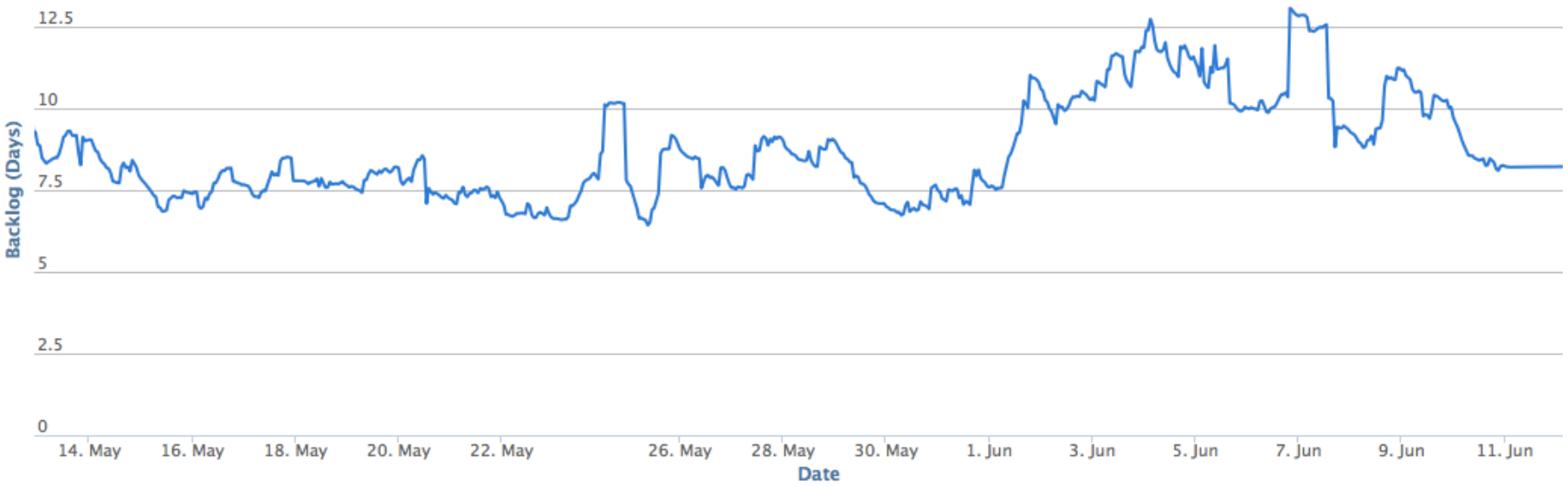
I submitted my job, but it is not running!

- That's what the batch queue is for!

- Your jobs will wait until the resources are available for them to run
- Your job's place in the queue is a mix of time and priority, so line jumping is allowed, but it may cost more

Cori Backlog Over Time

- Current backlog is ~8 days
- Plot obtained from MyNERSC



Average Queue Wait Time

- Historic data can be obtained from https://www.nersc.gov/users/queues/queue-wait-times/

	Hours Requested																																																	
Nodes	<1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	48+
1	0.7	40	46	18	15	4.5	16	22	24	45	27	61	8.3	6.8	10	13	21	24	11	13	14	12	5.0	7.9	27	12	27	50	38	0.0	59	0.0	24	8.8	0.0	5.4	68	0.0	117	0.0	0.0	0.0	0.0	139	83	0.0	78	60	21	115
2	0.1	4.0	4.7	7.5	23	18	21	11	33	76	13	31	56	32	1.8	71	0.0	281	0.0	0.0	107	0.0	0.0	103	67	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	194	0.0	0.0	0.0	71	0.0	0.0	0.0	0.0	0.0	0.0	78	93	0.0
3	0.4	7.2	23	12	17	26	6.9	23	29	77	72	0.0	39	120	0.0	90	0.0	0.0	0.0	0.0	25	0.0	0.0	86	313	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	181	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.4	8.9	13	17	20	22	15	0.0	6.9	0.0	0.0	31	93	0.0	0.0	0.0	0.0	0.0	0.0	0.0	102	0.0	188	0.0	84	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.9	2.8	42	42	11	24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	86	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.5	2.3	19	12	0.0	0.8	11	0.0	36	64	0.0	0.0	62	0.0	0.0	0.0	196	0.0	0.0	0.0	75	0.0	0.0	177	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	195	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.6	0.0	0.2	0.0	3.7	14	68	8.2	0.0	0.0	0.0	0.0	0.0	0.0	165	0.0	0.0	0.0	0.0	0.0	0.0	315	13	0.0	92	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	127	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	1.9	11	3.7	5.5	0.0	13	16	46	0.0	0.0	49	17	38	0.0	71	34	0.0	0.0	0.0	0.0	99	0.0	0.0	100	127	100	85	0.0	0.0	72	0.0	0.0	0.0	0.0	0.0	135	98	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	1.0	3.9	38	33	0.0	0.0	0.0	63	181	66	0.0	85	0.0	0.0	0.0	85	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	104	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	119	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.7	2.3	36	12	22	2.2	31	235	13	0.0	6.5	85	82	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	69	62	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	3.0	0.0	0.0	0.0	0.0	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.9	0.0	0.0	0.0	0.0	5.3	39	0.0	0.0	0.0	0.0	0.0	78	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	94	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	145	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	45	0.0	0.0	0.0	0.0	67	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.1	28	37	0.0	36	0.0	72	0.0	74	0.0	74	0.0	82	0.0	0.0	0.0	71	0.0	79	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	133	0.0	0.0	0.0	0.0	0.0	141	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.7	27	26	39	40	0.0	26	1.9	17	0.0	63	58	0.0	0.0	0.0	77	0.0	0.0	0.0	0.0	0.0	0.0	311	0.0	78	0.0	0.0	0.0	0.0	72	0.0	0.0	0.0	0.0	0.0	0.0	196	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17-19	0.4	0.0	1.4	17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	92	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20-23	0.5	14	59	0.0	1.1	15	33	0.0	0.0	0.0	0.0	0.0	87	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24-31	1.0	11	29	0.0	33	40	30	0.0	6.5	0.0	0.0	0.0	61	0.0	0.0	0.0	0.0	0.0	46	0.0	55	0.0	0.0	0.0	45	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	135	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
32-47	1.4	5.8	30	52	25	46	36	0.0	30	0.0	71	19	50	0.0	0.0	94	99	0.0	0.0	0.0	85	0.0	0.0	0.0	86	0.0	0.0	0.0	0.0	0.0	99	0.0	0.0	0.0	83	95	108	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48-63	1.5	9.7	16	0.7	3.0	43	86	0.0	0.0	0.0	304	0.0	84	0.0	0.0	98	0.0	342	0.0	0.0	105	0.0	0.0	0.0	24	0.0	0.0	0.0	0.0	0.0	64	0.0	0.0	0.0	0.0	0.0	42	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
64-127	2.0	140	4.2	18	22	0.0	7.6	0.0	0.0	0.0	49	67	1.0	0.0	0.0	123	0.0	340	59	0.0	111	0.0	0.0	0.0	79	0.0	32	0.0	121	0.0	0.0	0.0	93	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
128-255	7.5	44	75	25	0.0	0.0	4.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	129	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	116	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
256-511	10	42	53	0.0	51	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	66	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
512-1023	38	14	0.0	13	0.0	65	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	309	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1024-1535	86	0.0	0.0	0.0	113	0.0	0.0	0.0	0.0	0.0	0.0	0.0	167	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tips for Getting Better Throughput

- **Line jumping is allowed, but it may cost more**
- **Submit shorter jobs, they are easier to schedule**
  - Checkpoint if possible to break up long jobs
  - Short jobs can take advantage of 'backfill' opportunities
  - Run short jobs just before maintenance
- **Very important: make sure the wall clock time you request is accurate**
  - As noted above, shorter jobs are easier to schedule
  - Many users unnecessarily enter the largest wall clock time possible as a default
- **Bundle jobs (multiple "srun"s in one script, sequential or simultaneously)**
- **Use "shared" partition for serial jobs or very small parallel jobs.**
- **Queue wait time statistics**
  - https://www.nersc.gov/users/queues/queue-wait-times/

Places and Tools to Check Job Status

- **Completed jobs web page:**
  - https://www.nersc.gov/users/job-logs-statistics/completed-jobs/
- **MyNERSC Queues display**
  - https://my.nersc.gov/queues.php?machine=cori&full_name=Cori
- **Queue Wait Times**
  - http://www.nersc.gov/users/queues/queue-wait-times/
- **Scripts described on Queue Monitoring Page**
  - sqs, squeue, sstat, sprio, etc.
  - https://www.nersc.gov/users/computational-systems/cori/running-jobs/monitoring-jobs/

Not Covered in Details Here

- **Taskfarmer**
  - Manage single-core ("serial") or multi-core jobs
- **Using "realtime" partition**
- **"xfer" jobs**
  - Transfer data between Cori to archive
  - Run on external login nodes
- **Burst Buffer**
  - Non-volatile storage sits between memory and file system
  - Accelerate IO performance
- **Shifter**
  - Run jobs in custom environment
- **Advanced workflow**
- **Please refer to Cori running jobs web page**
  - http://www.nersc.gov/users/computational-systems/cori/running-jobs/

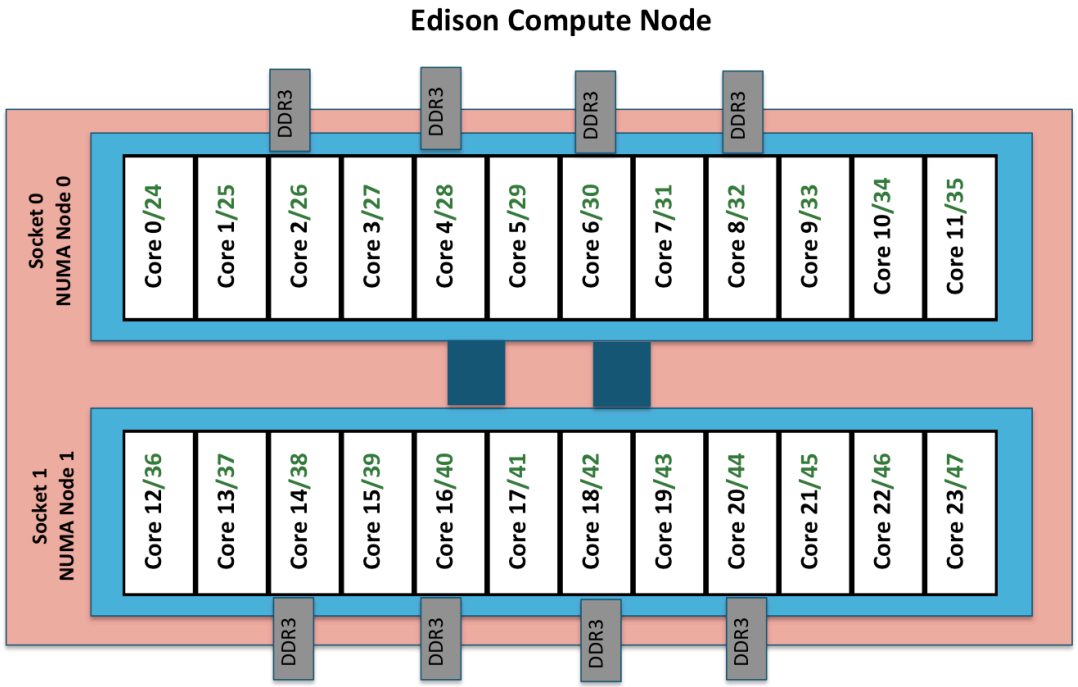
Demo and Hands on will be on Edison

- **Cori is currently down due to OS upgrade to prepare for Phase 2 KNL support**
- **Edison is also a Cray system with Intel Ivybridge processors.**
- **Edison has 24 cores per node as compared to 32 cores per node on Cori.**
- **Each Edison node has 64 GB of memory, as compared to 128 GB of memory for each Cori node.**
- **To do SLURM exercises**
  - % module load training
  - % cp -r $EXAMPLES/CoriP1/SLURM . (notice the “dot” at the end)

Edison - Cray XC30

- 133,824 cores, 5,576 nodes
- "Aries" interconnect
- 2 x 12-core Intel 'Ivy Bridge' 2.4 GHz processors per node
- 24 processor cores per node, 48 with hyperthreading
- 64 GB of memory per node
- 357 TB of aggregate memory
- 2.7 GB memory / core for applications
- /scratch disk quota of 10 TB
- 7.6 PB of /scratch disk
- Choice of full Linux operating system or optimized Linux OS (Cray Linux)
- Intel, Cray, and GNU compilers

Edison Compute Nodes



- **Edison: NERSC Cray XC30, 5,576 nodes, 133,824 cores.**
  - **2 NUMA domains per node, 12 cores per NUMA domain. 2 hardware threads per core.**
- **Memory bandwidth is non-homogeneous among NUMA domains.**

More Information

- **NERSC web pages**
  - http://www.nersc.gov/users/computational-systems/cori/running-jobs/
  - http://www.nersc.gov/users/computational-systems/edison/running-jobs/
- **SchedMD**
  - http://www.schedmd.com/
  - Man pages for SLURM commands

- **Contact NERSC Consulting**
  - Toll-free 800-666-3772
  - 510-486-8611, option #3
  - Email *consult@nersc.gov*

Thank You