Running jobs at NERSC (Cori, Edison)

Steve Leak, NERSC User Engagement Group
Key Points

• **HPC work is via batch system**
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• **Key commands:**
  – `sbatch` / `salloc` - submit a job
  – `srun` - start an (optionally MPI) application within a job
  – `sqs` - check the queue for my job status

• **Queues are long!**
  – Work *with* the system to get better turnaround time

• **Watch your budget! NERSC-hours and charge factors**

• **Help!** [consult@nersc.gov](mailto:consult@nersc.gov)

[www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs/]
Today’s Agenda

• How jobs work
• What *must* I request?
• My first job
• What else *can* I request?
• Where is my job?
• Working interactively
• Getting through the queue faster
• How usage is charged
• Which cores are running what? (advanced)
• Workflows - job arrays and dependencies
Key Points

• HPC work is via batch system
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• Key commands:
  – sbatch / salloc - submit a job
  – srun - start an (optionally MPI) application within a job
  – sqs - check the queue for my job status

• Queues are long!
  – Work with the system to get better turnaround time

• Watch your budget! NERSC-hours and charge factors

• Help! consult@nersc.gov

www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs/
Today’s Agenda

• How jobs work
  • What *must* I request?
  • My first job
  • What else *can* I request?
  • Where is my job?
  • Working interactively
  • Getting through the queue faster
  • How usage is charged
  • Which cores are running what? (advanced)
  • Workflows - job arrays and dependencies
How jobs work

Desktop / login node

- **Timeslicing**
  - core shared by multiple tasks
  - Works when the computer is mostly waiting for you

HPC

- You are waiting for the computer
- Subset of pooled resources dedicated to one job
How jobs work

- **Start on login node**
  - shared by many users, not for computational work
- **Access compute nodes** with `sbatch` or `salloc`
- **Batch script**
  - Copied to queue
  - Has directives for SLURM, and shell commands to perform on first compute node
- **Access your other allocated nodes** with `srun`
- **stdout, stderr saved to file**
  - (when running in batch mode)
Key Points

• HPC work is via batch system
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• Key commands:
  – sbatch / salloc - submit a job
  – srun - start an (optionally MPI) application within a job
  – sqs - check the queue for my job status

• Queues are long!
  – Work with the system to get better turnaround time

• Watch your budget! NERSC-hours and charge factors

• Help! consult@nersc.gov

www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs
Today’s Agenda

- How jobs work
- What must I request?
- My first job
- What else can I request?
- Where is my job?
- Working interactively
- Getting through the queue faster
- How usage is charged
- Which cores are running what? (advanced)
- Workflows - job arrays and dependencies
What the batch system needs to know:

• How many nodes (or CPUs) does this job need?
  – Mostly NERSC allocates and charges by the node
  – Jobs needing no more than half of one node - and willing to coexist - may request CPUs in the “shared” partition

• For how long does it need them?
  – Wallclock time limit

NERSC-specific extras:

• What type of CPU?
  – KNL or Xeon (haswell/ivybridge)?

• Which filesystems will this job use?
Nodes, cores, CPUs, threads, tasks - some definitions

- **Node** is the basic unit of allocation at NERSC
  - Think “one host” or “one server”
  - Single memory space, multiple CPU cores (24 or 32 or 68 ...)
    - And a core might support hyperthreading
Hyperthreading

- Fast timeslicing
  - Good when arithmetic units frequently wait on memory
- Core holds state of 2 (4 on KNL) processes, they share arithmetic units
- **SLURM views each hyperthread as a CPU**
- But most HPC jobs perform best when not sharing a core!
- Usually best to reserve 2 CPUs / core
A SLURM task is a reservation of CPUs and memory, up to one full node.

- A job has many tasks.

Eg: 3 possible tasks on 2 nodes.
What *must* I request?

What the batch system needs to know:

- **How many nodes (or CPUs) does this job need?**
  - Mostly NERSC allocates and charges by the node
  - Jobs needing no more than half of one node - and willing to coexist - may request CPUs in the “shared” partition
- **For how long does it need them?**
  - Wallclock time limit

NERSC-specific extras:

- **What type of CPU?**
  - KNL or Xeon (haswell/ivybridge)?
- **Which filesystems will this job use?**
• Optimal number of MPI ranks depends on your application and problem size (trial and error)
  – NERSC nodes mostly have 2-4 GB memory / core, 64-128 GB memory / node, and 24, 32 or 68 cores / node
  – You will need some minimum number of nodes for enough memory to hold the problem size
• Some applications are also multithreaded (OpenMP)
  – Trial and error to discover optimal number of OpenMP threads per MPI rank and of MPI ranks per node
• Most HPC applications run best with a full core (not just a hyperthread) dedicated to each OpenMP thread
#SBATCH -N 64        # request 64 nodes
srun -N 32 ./my_app  # start ./my_app on 32 of them
    # (default: 1 per node)
srun -n 128 ./my_app # start 128 instances of ./my_app,
    # across my 64 nodes (default is
    # to evenly distribute them in
    # block fashion)

Task 0: ./my_app
Task 1: ./my_app
Task 2: ./my_app
Task 3: ./my_app

Node 0
Node 1

Task 126: ./my_app
Task 127: ./my_app

Node 63

One MPI rank generally corresponds to one SLURM Task
#SBATCH -n 1024    # request sufficient nodes for 1024
#SBATCH -c 2       # SLURM tasks (MPI ranks), with 2 cpus
    # (hyperthreads), ie 1 full Xeon core,  
    # per task

#SBATCH -n 1024    # request nodes for 1024
#SBATCH --ntasks-per-node=24  # MPI ranks, with no more
    # than 24 ranks on any node

#SBATCH -n 1024    # request nodes for 1024 MPI ranks,  
#SBATCH -c 8       # with 8 cpus (ie 4 Xeon cores or
    # 2 KNL cores) per rank  
    # (suitable with OMP_NUM_THREADS=4)
• **A partition** is the subset of nodes your job can use
  – Each partition has rules about size of job that can use it

• **NERSC partitions**
  – **debug** - for small, short jobs needing quick turnaround (up to 64 or 512 nodes, up to 30 minutes - default partition for jobs that fit)
  – **regular** - for most real work (up to whole partition, or up to 4 days)
  – **shared** - for jobs needing half-a-node or less (up to 2 days)

Request partition with (eg):
#SBATCH -p debug
What must I request?

What the batch system needs to know:

• How many nodes (or CPUs) does this job need?
  – Mostly NERSC allocates and charges by the node
  – Jobs needing no more than half of one node - and willing to coexist - may request CPUs in the “shared” partition

• For how long does it need them?
  – Wallclock time limit

NERSC-specific extras:

• What type of CPU?
  – KNL or Xeon (haswell/ivybridge)?

• Which filesystems will this job use?
#SBATCH -t 30       # 30 minutes
#SBATCH -t 30:00    # 30 minutes
#SBATCH -t 1:00:00  # 1 hour
#SBATCH -t 1-0      # 1 day
#SBATCH -t 1-12     # 1.5 days

• Wallclock time, ie real elapsed time
• After this much time, SLURM can kill this job
What the batch system needs to know:

• How many nodes (or CPUs) does this job need?
  – Mostly NERSC allocates and charges by the node
  – Jobs needing no more than half of one node - and willing to coexist - may request CPUs in the “shared” partition

• For how long does it need them?
  – Wallclock time limit

NERSC-specific extras:

• What type of CPU?
  – KNL or Xeon (haswell/ivybridge)?

• Which filesystems will this job use?
What type of CPU?

- Cori has 2 types of nodes (haswell, knl)
- The KNL nodes have multiple modes for:
  - MCDRAM (flat, cache, hybrid)
  - NUMA arrangement of a node (quad, hemi, a2a, snc4, snc2)
- No default (you must specify)
- Specify via SLURM constraint option, -C
  
  #SBATCH -C haswell
  #SBATCH -C knl,quad,cache
  #SBATCH -C knl,quad,flat

- Supported also on Edison (ivybridge)
Which filesystems?

- **Context:** NERSC provides several filesystems
  - Most jobs use 1 or 2 of them ($SCRATCH or $PROJECT)
- **Problem:** sometimes a filesystem is unavailable
  - Maintenance, or a failure
  - Jobs that try to use an unavailable filesystem tend to crash
    - Maybe after several days in queue!
- **Solutions:**
  - Let them crash
  - Stop *all* jobs when *any* filesystem is unavailable
  - Require that jobs specify which filesystems they need, and stop only those jobs (from starting)
Which filesystems?

- Specify filesystems via SLURM “license” feature:
  
  #SBATCH -L SCRATCH
  #SBATCH -L scratch1, project

- Not used for $HOME
  - (Everything needs $HOME)

- Filesystems you can specify:
  - SCRATCH (scratch1, scratch2, cscratch1)
  - project
  - projecta, projectb, dna
  - scratch3 (Edison only)
  - cscratch1 (either cluster)
Today’s Agenda

• How jobs work
• What *must* I request?
• **My first job**
• What else *can* I request?
• Where is my job?
• Working interactively
• Getting through the queue faster
• How usage is charged
• Which cores are running what? (advanced)
• Workflows - job arrays and dependencies
My first job

A SLURM job script has two sections:

1. Directives telling SLURM what you would like it to do with this job
2. The script itself - shell commands to run on the first compute node

```
#!/bin/bash -l

#SBATCH -t 00:30:00
#SBATCH -N 2
#SBATCH --license=SCRATCH

export RUNDIR=$SCRATCH/run-$SLURM_JOBID
mkdir -p $RUNDIR
cd $RUNDIR

srun -n 4 bash -c 'echo "Hello, world, from node $(hostname)"'
```

```
elvis@nersc:~> vi myscript.q
elvis@nersc:~> sbatch -C $CRAY_CPU_TARGET myscript.q
Submitted batch job 2774102
```
A SLURM job script has two sections:

1. Directives telling SLURM what you would like it to do with this job
2. The script itself - shell commands to run on the first compute node

How many nodes? For how long?

No --qos == normal priority and cost

```
#!/bin/bash

#SBATCH -t 00:30:00
#SBATCH -N 2
#SBATCH --license=SCRATCH

export RUNDIR=$SCRATCH/run-$SLURM_JOBID
mkdir -p $RUNDIR
cd $RUNDIR

srun -n 4 bash -c 'echo "Hello, world, from node $(hostname)"'
```

```
elvis@nersc:~> vi myscript.q
```

$SCRATCH filesystem

Xeon nodes on current cluster (set by craype-{haswell,ivybridge} module)

Note: cannot use env vars in directives - but directives have equivalent command-line option
A SLURM job script has two sections:

1. Directives telling SLURM what you would like it to do with this job
2. The script itself - shell commands to run on the first compute node

```
#!/bin/bash -l

#SBATCH -t 00:30:00
#SBATCH -N 2
#SBATCH --license=SCRATCH

export RUNDIR=$SCRATCH/run-$SLURM_JOBID
mkdir -p $RUNDIR
cd $RUNDIR

srun -n 4 bash -c 'echo "Hello, world, from node $(hostname)"'
```

```
elvis@nersc:~> vi myscript.q

elvis@nersc:~/myscript.q $ vi
```

"sbatch" submits a job script

```
elvis@nersc:~> sbatch -C $CRAY_CPU_TARGET myscript.q
Submitted batch job 2774102
```
Key Points

• HPC work is via batch system
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• Key commands:
  – sbatch / salloc - submit a job
  – srun - start an (optionally MPI) application within a job
  – sqs - check the queue for my job status

• Queues are long!
  – Work with the system to get better turnaround time

• Watch your budget! NERSC-hours and charge factors

• Help! consult@nersc.gov

www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs/
NERSC job script generator tool

This tool generates a batch script template which also realizes specific process and thread binding configurations.

```bash
#!/bin/bash
#SBATCH -N 2
#SBATCH -C haswell
#SBATCH -p debug
#SBATCH -t 03:00:00

# OpenMP settings:
export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

#run the application:
mpirun -n 8 -c 16 --cpu_bind=cores myapp.x
```
Today’s Agenda

• How jobs work
• What *must* I request?
• My first job
• **What else *can* I request?**
• Where is my job?
• Working interactively
• Getting through the queue faster
• How usage is charged
• Which cores are running what? (advanced)
• Workflows - job arrays and dependencies
What else can I request?

• **Quick vs cheap (QOS)**
  – Spend more to jump the queue?
    Or wait longer to spend less?

• **Faster I/O (Burst Buffer - Cori only)**
  – Cori has 1.8PB of SSD-based “Burst Buffer” to support I/O intensive workloads
  – Jobs can request a job-temporary BB filesystem, or a persistent (up to a few weeks) reservation

• **Containerized runtime environment (Shifter)**
  – Docker images can be pulled to the NERSC Shifter Image Gateway and used as an alternate, portable runtime environment for a job
  – Performance benefits for some dynamically-linked executables (especially Python)
For jobs in "-p regular"

- Affects where in queue your job starts
  - And the cost!

(NERSC-hours)

#SBATCH --qos=premium

#SBATCH --qos=low

Priority increases with time

Resources are reserved for these jobs

- premium: 2x cost
- normal
- low: ½ x cost
- (scavenger) (free)

These jobs are started if there is a big enough gap in resource allocation.
What else can I request?

- **Quick vs cheap (QOS)**
  - Spend more to jump the queue?
  - Or wait longer to spend less?

- **Faster I/O (Burst Buffer - Cori only)**
  - Cori has 1.8PB of SSD-based “Burst Buffer” to support I/O intensive workloads
  - Jobs can request a job-temporary BB filesystem, or a persistent (up to a few weeks) reservation

- **Containerized runtime environment (Shifter)**
  - Docker images can be pulled to the NERSC Shifter Image Gateway and used as an alternate, portable runtime environment for a job
  - Performance benefits for some dynamically-linked executables (especially Python)
• Cori has 1.8PB of SSD-based “Burst Buffer” to support I/O intensive workloads
  – Multiple read/write
  – Time-critical read/write (eg writing checkpoint files)
• Stage-in/stage-out: data is moved before/after the job
  – Can move individual files, or all files in a directory

• More on BB tomorrow
• **Job-temporary filesystem:**
  - Note “#DW” not “#SBATCH”
  - Also note environment variable in #DW directive - special case!
    (SLURM normally cannot expand environment variables in directives)

```bash
#DW jobdw capacity=100GB access_mode=striped type=scratch pool=sm_pool
#DW stage_in source=/global/cscratch1/sd/username/path/to/dirname
destination=$DW_JOB_STRIPED type=directory
#DW stage_out source=$DW_JOB_STRIPED/filename
destination=/global/cscratch1/sd/username/path/to/filename type=file
```

• **Access BB directory in job script:**

```bash
cd $DW_JOB_STRIPED
```
Burst Buffer

• **Persistent reservation:**
  – Good for multi-job workflow
  – Not reliable storage!
  – Note creation and deletion use “#BB” not “#DW” or “#SBATCH”

• **Create:**
  
  #BB create_persistent name=myBBname capacity=100GB access=striped
type=scratch

• **Delete:**
  
  #BB destroy_persistent name=myBBname

• **Use existing:**
  
  #DW persistentdw name=myBBname

• **Access BB directory in job script:**
  
  cd $DW_JOB_PERSISTENT
What else can I request?

- **Quick vs cheap (QOS)**
  - Spend more to jump the queue?
    Or wait longer to spend less?

- **Faster I/O (Burst Buffer - Cori only)**
  - Cori has 1.8PB of SSD-based “Burst Buffer” to support I/O intensive workloads
  - Jobs can request a job-temporary BB filesystem, or a persistent (up to a few weeks) reservation

- **Containerized runtime environment (Shifter)**
  - Docker images can be pulled to the NERSC Shifter Image Gateway and used as an alternate, portable runtime environment for a job
  - Performance benefits for some dynamically-linked executables (especially Python)
Shifter

- Docker-like container environment for HPC
- Works with MPI, with $SCRATCH, with Burst Buffer, etc
- Solves performance issues relating to dynamically loaded libraries

- More on Shifter tomorrow
1. Create Docker image
2. Push it to DockerHub
3. Pull it to NERSC Shifter ImageGateway
   (Not currently available on Cori, use Edison for this step)
   
   ```
   module load shifter
   shifterimg -v pull docker:image_name:image_version_tag
   shifterimg images  # list available images
   ```

4. Use it in a job
   ```bash
   #!/bin/bash -l
   #SBATCH -N 2
   #SBATCH -t 30
   #SBATCH -L project,cscratch1
   #SBATCH --image=docker:image_name:image_version_tag
   #SBATCH --volume="/global/project/projectdirs/mpccc:/input;/global/cscratch1/sd/username/path/to/output:/output"
   
   cd /output
   srun -n64 shifter python ./my_python_app.py < /input/input_file.txt
   ```
Today’s Agenda

- How jobs work
- What must I request?
- My first job
- What else can I request?
- Where is my job?
- Working interactively
- Getting through the queue faster
- How usage is charged
- Which cores are running what? (advanced)
- Workflows - job arrays and dependencies
elvis@nersc:~> sqs

<table>
<thead>
<tr>
<th>JOBID</th>
<th>ST</th>
<th>REASON</th>
<th>USER</th>
<th>NAME</th>
<th>NODES USED</th>
<th>REQUESTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>2774102</td>
<td>R</td>
<td>Prolog</td>
<td>elvis</td>
<td>myscript.q</td>
<td>2</td>
<td>0:00</td>
</tr>
</tbody>
</table>

... SUBMIT PARTITION RANK_P RANK_BF
2016-11-18T11:24:20 debug N/A N/A

elvis@nersc:~> ls -lt

total 11280
-drw-r---- 1 elvis elvis 132 Nov 18 11:24 slurm-2774102.out
-drw-r---- 1 elvis elvis 208 Nov 18 11:24 myscript.q
Where is my job?

<table>
<thead>
<tr>
<th>Job ID</th>
<th>User</th>
<th>Nodes</th>
<th>Status</th>
<th>Queue</th>
<th>Submit Time</th>
<th>Time Required</th>
<th>Time Used</th>
<th>Priority Rank</th>
<th>Backfill Rank</th>
<th>QOS</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>3632807</td>
<td>azamat</td>
<td>2048</td>
<td>Queued</td>
<td>knl</td>
<td>2017-02-21T19:35:37</td>
<td>3:00:00</td>
<td>0:00</td>
<td>0001</td>
<td>0001</td>
<td>normal</td>
<td>Resources</td>
</tr>
<tr>
<td>3632873</td>
<td>plevine</td>
<td>30</td>
<td>Queued</td>
<td>regular</td>
<td>2017-02-22T13:34:12</td>
<td>36:00:00</td>
<td>0:00</td>
<td>0002</td>
<td>0002</td>
<td>premium</td>
<td>Resources</td>
</tr>
<tr>
<td>3633023</td>
<td>dli</td>
<td>18</td>
<td>Queued</td>
<td>debug</td>
<td>2017-02-22T14:04:15</td>
<td>0:00:00</td>
<td>0:00</td>
<td>0003</td>
<td>0003</td>
<td>debug</td>
<td>Resources</td>
</tr>
<tr>
<td>3633077</td>
<td>eer492</td>
<td>56</td>
<td>Queued</td>
<td>debug</td>
<td>2017-02-22T14:08:10</td>
<td>0:00:00</td>
<td>0:00</td>
<td>0004</td>
<td>0004</td>
<td>debug</td>
<td>Resources</td>
</tr>
<tr>
<td>3633080</td>
<td>igerherd</td>
<td>20</td>
<td>Queued</td>
<td>debug</td>
<td>2017-02-22T14:09:05</td>
<td>0:00:00</td>
<td>0:00</td>
<td>0005</td>
<td>0005</td>
<td>debug</td>
<td>Resources</td>
</tr>
<tr>
<td>3633104</td>
<td>jianli</td>
<td>24</td>
<td>Queued</td>
<td>debug</td>
<td>2017-02-22T14:12:34</td>
<td>10:00:00</td>
<td>0:00</td>
<td>0006</td>
<td>0006</td>
<td>debug</td>
<td>Resources</td>
</tr>
<tr>
<td>3633171</td>
<td>bonisto</td>
<td>16</td>
<td>Queued</td>
<td>debug</td>
<td>2017-02-22T14:15:46</td>
<td>20:00:00</td>
<td>0:00</td>
<td>0007</td>
<td>0007</td>
<td>debug</td>
<td>Resources</td>
</tr>
<tr>
<td>3633173</td>
<td>miler86</td>
<td>1</td>
<td>Queued</td>
<td>debug</td>
<td>2017-02-22T14:15:56</td>
<td>30:00:00</td>
<td>0:00</td>
<td>0008</td>
<td>0008</td>
<td>debug</td>
<td>Resources</td>
</tr>
<tr>
<td>3633046</td>
<td>tkurnh</td>
<td>1024</td>
<td>Queued</td>
<td>knl_reboot</td>
<td>2017-02-22T11:41:02</td>
<td>8:00:00</td>
<td>0:00</td>
<td>0009</td>
<td>0009</td>
<td>normal</td>
<td>Resources</td>
</tr>
<tr>
<td>36338177</td>
<td>tkurnh</td>
<td>256</td>
<td>Queued</td>
<td>knl_reboot</td>
<td>2017-02-22T14:16:56</td>
<td>1:00:00</td>
<td>0:00</td>
<td>0010</td>
<td>0010</td>
<td>normal</td>
<td>Resources</td>
</tr>
</tbody>
</table>

Showing 1 to 10 of 3,477 entries

Toggle Columns: Job Name, Repo, Nodes, Queue, Submit Time, Time Required, Time Used, Priority Rank, Backfill Rank

Click here for the Classic Queue Look
Where is my job?

Job states you might see:

- **nPENDING (PD)**: Job submitted, waiting in queue
- **nCONFIGURING (CF)**: Nodes are being prepared to run the job
- **nRUNNING (R)**: Processes are being killed, nodes being returned to pristine state
- **nCOMPLETING (CG)**: Job is completed

Note: completed jobs are not visible from sqs
Use sacct
Key Points

• **HPC work is via batch system**
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• **Key commands:**
  – `sbatch / salloc` - submit a job
  – `srun` - start an (optionally MPI) application within a job
  – `sqs` - check the queue for my job status

• **Queues are long!**
  – Work *with* the system to get better turnaround time

• **Watch your budget! NERSC-hours and charge factors**

• **Help!** [consult@nersc.gov](mailto:consult@nersc.gov)

[www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs/]
Today’s Agenda

- How jobs work
- What *must* I request?
- My first job
- What else *can* I request?
- Where is my job?

- **Working interactively**
- Getting through the queue faster
- How usage is charged
- Which cores are running what? (advanced)
- Workflows - job arrays and dependencies
• “salloc” to start an interactive session on compute nodes
• Takes the same arguments as sbatch
• Still a batch job!
  – Use “-p debug” to minimize waiting time
• Inherits environment (eg modules you have loaded)
• Not a login shell!
  – Run “source ~/.bash_profile” to get aliases etc
• “srun” to run a command across all my nodes
Today’s Agenda

• How jobs work
• What *must* I request?
• My first job
• What else *can* I request?
• Where is my job?
• Working interactively
• **Getting through the queue faster**
• How usage is charged
• Which cores are running what? (advanced)
• Workflows - job arrays and dependencies
How scheduling works

- **Jobs scheduled in two passes:**
  - Priority pass: highest-priority jobs are scheduled ASAP
  - Backfill pass: remaining jobs are scanned to find jobs that can start now in a gap left by priority pass

- **Short jobs are better backfill candidates than long jobs**
  - *Even if they require many nodes*

- **Jobs that can’t run in backfill take several days to accumulate sufficient priority for the priority pass**
## Queue wait time example

### Edison - 2017 so far

| Hours Requested | 0 | 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 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |
| Modes           | <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 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |
QOS Reminder

- For jobs in “-p regular”
- Affects where in queue your job starts
  - And the cost! (NERSC-hours)

`#SBATCH --qos=premium`

`#SBATCH --qos=low`

- Resources are reserved for these jobs
- Priority increases with time
- These jobs are started if there is a big enough gap in resource allocation

- Resources are reserved for these jobs
- Priority increases with time
- These jobs are started if there is a big enough gap in resource allocation
Key Points

- **HPC work is via batch system**
  - Dedicated subset of compute resources
  - Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

- **Key commands:**
  - sbatch / salloc - submit a job
  - srun - start an (optionally MPI) application within a job
  - sqs - check the queue for my job status

- **Queues are long!**
  - Work *with* the system to get better turnaround time

- **Watch your budget! NERSC-hours and charge factors**

- **Help!** consult@nersc.gov

[Links]
www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs/
Today’s Agenda

• How jobs work
• What must I request?
• My first job
• What else can I request?
• Where is my job?
• Working interactively
• Getting through the queue faster
• **How usage is charged**
• Which cores are running what? (advanced)
• Workflows - job arrays and dependencies
How usage is charged

“NERSC-hours” - based on performance/node relative to 1 core of Hopper (past NERSC system)

1 wallclock-hour is many NERSC-hours!

<table>
<thead>
<tr>
<th>System</th>
<th>Node Architecture</th>
<th>Base Charge per Node Hour (NERSC Hours)</th>
<th>System Size (Nodes)</th>
<th>Cores per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cori</td>
<td>Intel Xeon Phi (KNL)</td>
<td>96*</td>
<td>9,303</td>
<td>68</td>
</tr>
<tr>
<td>Cori</td>
<td>Intel Xeon (Haswell)</td>
<td>80</td>
<td>2,004</td>
<td>32</td>
</tr>
<tr>
<td>Edison</td>
<td>Intel Xeon (Ivy Bridge)</td>
<td>48</td>
<td>5,576</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Charge Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premium Queue Priority</td>
<td>Cori and Edison</td>
</tr>
<tr>
<td>Low Queue Priority</td>
<td>Cori and Edison</td>
</tr>
<tr>
<td>Big Job Discount*</td>
<td>Edison jobs that use &gt;683 nodes</td>
</tr>
<tr>
<td>Shared Node</td>
<td>Cori Haswell nodes</td>
</tr>
<tr>
<td>Scavenger</td>
<td>Cori and Edison</td>
</tr>
<tr>
<td>Cori Xeon Phi Pre-Production</td>
<td>Cori Xeon Phi (KNL) nodes</td>
</tr>
</tbody>
</table>

- 2 x base charge
- 0.5 x base charge
- 0.6 x base charge
- 2.5 NERSC hours per core hour
- No charge (“free”) until production computing begins on July 1, 2017

*NERSC-hours* - based on performance/node relative to 1 core of Hopper (past NERSC system)
Your repo is charged for each node your job was allocated, for the entire duration of the job:

```bash
#SBATCH -N 64
#SBATCH -t 1:00:00
srun -N 32 -t 30 ./my_long_app.x
```

- App only ran on 32 nodes, but 64 nodes were requested (and allocated), so you are charged for 64 nodes
- 1 hour was requested, but the job ended after 30 minutes, so you are charged for 30 minutes

If you have access to more than one repo, specify which to charge in your batch script:

```bash
#SBATCH -A repo_name
```
Key Points

• HPC work is via batch system
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• Key commands:
  – sbatch / salloc - submit a job
  – srun - start an (optionally MPI) application within a job
  – sqs - check the queue for my job status

• Queues are long!
  – Work with the system to get better turnaround time

• Watch your budget! NERSC-hours and charge factors

• Help! consult@nersc.gov

www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs/
Today’s Agenda

• How jobs work
• What *must* I request?
• My first job
• What else *can* I request?
• Where is my job?
• Working interactively
• Getting through the queue faster
• How usage is charged
• **Which cores are running what?** (advanced)
• Workflows - job arrays and dependencies
Modern compute nodes have multiple sockets, cores, hyperthreads and Non-uniform memory access (NUMA)

- Two tasks (OpenMP thread or MPI ranks) using hyperthreads of same core are contending for arithmetic, cache resources
- A task using memory from opposite socket has reduced memory bandwidth
- Two OpenMP threads on opposite sockets might cause cache thrashing between the two L3 caches
Process and memory affinity

 KNL node is even more complex

- Tiles as well as cores and hyperthreads
- Up to 8 NUMA nodes for tasks and memory to land on

Thread placement and memory affinity are increasingly important for good performance
srun -n 64 -c 4 ./my_exec

- The “-c” sbatch/srun option controls **number of CPUs reserved per task**, *not* task placement or binding!
- Linux will place threads wherever it sees fit, eg “cram them into the fewest possible number of cores, leaving other cores empty” (not ideal!)
- If (“-n” * “-c” != total_available_cpus) then SLURM+Linux can get confused => pathologically bad placement

- Solution: use --cpu_bind:

```
srun -n 64 -c 4 --cpu_bind=verbose,cores ./my_exec
srun -n 128 -c 2 --cpu_bind=verbose,threads ./my_exec
```
Solution: use --cpu_bind:

```
srun -n 64 -c 4 --cpu_bind=verbose,cores ./my_exec
srun -n 128 -c 2 --cpu_bind=verbose,threads ./my_exec
```

- **Controls what a task (MPI rank) is bound to**
  - If no more than 1 MPI rank per core:  `--cpu_bind=cores`

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>68</td>
<td>1</td>
<td>69</td>
</tr>
<tr>
<td>138</td>
<td>204</td>
<td>137</td>
<td>203</td>
</tr>
<tr>
<td></td>
<td></td>
<td>task 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>task 1</td>
<td></td>
</tr>
</tbody>
</table>

- If more than 1 MPI rank per core:  `--cpu_bind=threads`

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>70</td>
<td>3</td>
<td>71</td>
</tr>
<tr>
<td>138</td>
<td>206</td>
<td>139</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td></td>
<td>task 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>task 3</td>
<td></td>
</tr>
</tbody>
</table>

```
export OMP_NUM_THREADS=2
export OMP_PROC_BIND=spread    # or close
export OMP_PLACES=cores        # or threads, or sockets
srn -n 32 -c 8 --cpu_bind=verbose,cores   ./my_exec

...If using hyperthreads, use OMP_PLACES=threads
Memory affinity

Linux default behavior is to allocate to closest NUMA-node, if possible

Not always optimal:

- KNL nodes: DDR is “closer” than MCDRAM

#SBATCH -C knl,quad,flat
export OMP_NUM_THREADS=4
srun -n16 -c16 --cpu_bind=cores --mem_bind=map_mem:1 ./a.out

- NUMA node 1 is MCDRAM in quad,flat mode
- “Mandatory” mapping: if using >16GB, malloc will fail
“Preferred” affinity: if preferred NUMA node is full, allocate to another NUMA node

- Not yet supported by SLURM
- Use numactl instead

```bash
module load numactl
srun -n16 -c16 --cpu_bind=cores numactl -p 1 ./a.out
```
Today’s Agenda

• How jobs work
• What *must* I request?
• My first job
• What else *can* I request?
• Where is my job?
• Working interactively
• Getting through the queue faster
• How usage is charged
• Which cores are running what? (advanced)
• **Workflows - job arrays and dependencies**
Job arrays

Need to run a set of almost-identical jobs?

```bash
#!/bin/bash -l
#SBATCH -N 64
#SBATCH -t 6:00
#SBATCH -L SCRATCH
#SBATCH -C knl,quat,cache

runid=19 # update before each job
mkdir $SCRATCH/runs-$runid
cd $SCRATCH/runs-$runid
export OMP_NUM_THREADS=16
export OMP_PLACES=cores
export OMP_PROC_BIND=spread
srun -n 1024 -c 64
    --cpu_bind=cores ./a.out $runid
```

```bash
#!/bin/bash -l
#SBATCH -N 64
#SBATCH -t 6:00
#SBATCH -L SCRATCH
#SBATCH -C knl,quat,cache
#SBATCH --array=1-100

runid=$SLURM_ARRAY_JOB_ID
mkdir $SCRATCH/runs-$runid
cd $SCRATCH/runs-$runid
export OMP_NUM_THREADS=16
export OMP_PLACES=cores
export OMP_PROC_BIND=spread
srun -n 1024 -c 64
    --cpu_bind=cores ./a.out $runid
```
Job arrays

- Convenient way to manage sets of near-identical jobs
- The SLURM directives describe resources for a single job in the array
- Appears in queue as, eg 1234567_1, 1234567_2, ...
- To cancel an individual member:
  `scancel 1234567_7    # cancel array member number 7`
- To cancel the whole array:
  `scancel 1234567`
Job array gotchas

• The SLURM directives describe resources for a *single job in the array*
  – Common help ticket: “I wanted to run 1000 copies of my 1-node job for 1 hour, so I submitted this:”
    
    ```
    #SBATCH -N 1000
    #SBATCH -t 1000:00
    #SBATCH --array=1-1000
    ```
  – Desired effect: uses 1000 node-hours (80000 NERSC-hours)
  – Actual effect: uses 1000x1000x1000 node-hours (80 billion NERSC-hours = whole year allocation for repo
    • … And P.I. is *furious*

• Lesson: test your script on a small job array first!
Some workflows require a job to run only after another job has completed (or perhaps, only if another job fails)

elvis@nersc:~> sbatch job1.q
Submitted job 5436
elvis@nersc:~> sbatch -d afterok:5436 job2.q
Key Points

• **HPC work is via batch system**
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• **Key commands:**
  – `sbatch` / `salloc` - submit a job
  – `srun` - start an (optionally MPI) application within a job
  – `sqs` - check the queue for my job status

• **Queues are long!**
  – Work *with* the system to get better turnaround time

• **Watch your budget! NERSC-hours and charge factors**

• **Help!** [consult@nersc.gov](mailto:consult@nersc.gov)

[www.nersc.gov/users/computational-systems/{cori,edison}/running-jobs/]