Running Jobs on Edison and Cori

Helen He
NERSC User Engagement Group

New User Training
March 21, 2018
Outline

• Systems and running jobs introductions
• Batch script examples
• More on KNL with process/thread/memory affinity considerations
• Advanced workflow options
• Monitoring jobs
Edison - Cray XC30

- 134,064 cores, 5,586 nodes
- 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
- “Aries” interconnect
- 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
Cori - Cray XC40

• 2,388 Haswell nodes (76,416 cores)
  9,388 KNL nodes (658,784 cores)
• Haswell: 2 x 16-core Intel Haswell 2.3 GHz processors (32 processor cores per node, 64 with hyperthreading), 128 GB of memory per node, 298.5 TB of aggregate memory
• KNL: 68-core Intel KNL 1.4 GHz processors (68 processor cores per node, 272 with hyperthreading), 96 GB of DDR memory plus 16 GB of MCRAM per node, 1.06 PB of aggregate memory
• “Aries” interconnect
• 4 GB memory / core (Haswell) and 1.4 GB DDR + 0.23 GB MCDRAM /core (KNL) 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
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 and interactive jobs are supported for up to 30 min and 4 hrs respectively
- 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
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
  – Edison has SandyBridge login nodes
  – Cori has Haswell login nodes

• **Compute nodes**
  – Execute your application
  – Dedicated resources for your job
  – Edison has IvyBridge compute nodes
  – Cori has Haswell and KNL compute nodes. Binaries built for Haswell can run on KNL nodes, but not vice versa
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

• Run in `$SCRATCH` or `/project`, do not run in your `$HOME` which is tuned for large number of smaller files, usually not optimal IO for large application runs.

• Submit the job to the queuing system with the `sbatch` or `salloc` command

  ```
  % sbatch my_batch_script
  or % salloc <command line options>
  ```
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
Cori Haswell Compute Nodes

- Each node has 2 Intel Xeon 16-core Haswell processors
- 2 NUMA domains (sockets) per node, 16 cores per NUMA domain. 2 hardware threads per core. Physical core
- NUMA Domain 0: physical cores 0-15 (and logical cores 32-47);
  NUMA Domain 1: physical cores 16-31 (and logical cores 48-63)
- Memory bandwidth is non-homogeneous among NUMA domains.

To obtain processor info:

Get on a compute node: % salloc -N 1 -C …

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

Similar for Edison, except each socket has 12 physical cores
Cori KNL Example Compute Nodes

- A Cori KNL node has 68 cores/272 CPUs, 96 GB DDR memory, 16 GB high bandwidth on package memory (MCDRAM).
- Three cluster modes, all-to-all, quadrant, sub-NUMA clustering, and 3 memory modes (cache, flat, hybrid) are available at boot time to configure the KNL mesh interconnect.

![Arrangement of Hardware Threads for 68 Core KNL](image)

- A quad,cache node (default setting) has only 1 NUMA node with all CPUs on the NUMA node 0 (DDR memory). MCDRAM is hidden from the "numactl -H" result since it is a cache.

Other combinations are by reservation only:
- A quad,flat node has only 2 NUMA nodes with all CPUs on the NUMA node 0 (DDR memory). NUMA node 1 has MCDRAM only.
- A snc2,flat node has 4 NUMA domains with DDR memory and all CPUs on NUMA nodes 0 and 1.
Sample Cori Haswell Batch Script - MPI

### Command

```bash
#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 1:00:00
#SBATCH -C haswell
#SBATCH -L SCRATCH
#SBATCH -J myjob

echo "Environment is automatically imported"
export OMP_NUM_THREADS=1
srun -n 1280 -c 2 --cpu_bind=cores ./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 Haswell Batch Script
- MPI

#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 1:00:00
#SBATCH -C haswell
#SBATCH -L SCRATCH
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 -c 2 --cpu_bind=cores ./mycode.exe

Job directives: instructions for the batch system
- Submission QOS (default is “debug”)
- How many compute nodes to reserve for your job
- How long to reserve those nodes
- What type of compute nodes to use.
  - knl, quad, cache (mostly) for KNL, optional for Edison
  - More optional SBATCH keywords
#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 1:00:00
#SBATCH -C haswell
#SBATCH -L SCRATCH
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 -c 2 --cpu_bind=cores ./mycode.exe

SBATCH optional keywords:
• What file systems my job depends on (prevent to start when there are file system issues)
• what to name my job
• what to name STDOUT files
• what account to charge
• whether to notify you by email when your job finishes
• ...
Sample Cori Haswell Batch Script - MPI

```bash
#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 1:00:00
#SBATCH -C haswell
#SBATCH -L SCRATCH
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 -c 2 --cpu_bind=cores ./mycode.exe
```

- No need to set this if your application programming model is pure MPI.
- If your code is hybrid MPI/OpenMP, or to prevent from using threaded libraries, set OMP_NUM_THREADS to 1 to run in pure MPI mode.
Sample Cori Haswell Batch Script - MPI

#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -C haswell
#SBATCH -L SCRATCH
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 -c 2 --cpu_bind=cores ./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.
Sample Cori Haswell Batch Script
- MPI

```bash
#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 40
#SBATCH -t 1:00:00
#SBATCH -C haswell
#SBATCH -L SCRATCH
#SBATCH -J myjob

export OMP_NUM_THREADS=1
srun -n 1280 -c 2 --cpu_bind=cores ./mycode.exe
```

- There are 64 logical CPUs (the number Slurm sees) on each node
- "-c" specifies # logical CPUs to be allocated to each MPI task
- --cpu_bind is critical especially when nodes are not fully occupied
  - use "--cpu_bind=cores" when #MPI tasks <= # physical cores per node
  - use "--cpu_bind=threads" when #MPI tasks > # physical cores per node
- With 40 nodes, using hyperthreading, up to 40*64=2,560 MPI tasks can be launched: “srun -n 2560 -c 1 --cpu_bind=threads ./mycodee.exe” is OK
Sample Cori Haswell Batch Script
- Hybrid MPI/OpenMP

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

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=true
export OMP_PLACES=threads

srun -n 160 -c 16 --cpu_bind=cores ./mycode.exe
```

- Set `OMP_NUM_THREADS`
- Use OpenMP standard settings for process and thread affinity
- Again, “-c” specifies # logical CPUs to be allocated to each MPI task
  - with 4 MPI tasks per node, set 64 logical CPUs /4 =16 for ”-c”
  - “-c” value should be >= `OMP_NUM_THREADS`
Serial Jobs

• The “shared” QOS allows multiple executables from different users to share a node
• Each serial job run on a single core of a “shared” node
• Up to 24 (Edison) or 32 (Cori Haswell) jobs from different users depending on their memory requirements

```bash
#SBATCH -q shared
#SBATCH -t 1:00:00
#SBATCH --mem=4GB
#SBATCH -C haswell
#SBATCH -J my_job
./mycode.x
```

• Small parallel job that use less than a full node can also run in the “shared” partition
• https://www.nersc.gov/users/computational-systems/cori/running-jobs/example-batch-scripts/#toc-anchor-10

• Do not specify #SBATCH -N”
• Default “#SBATCH -n” is 1
• Default memory is 1,300 MB for Edison and 1,952 MB for Haswell
• Use -n or --mem to request more slots for larger memory
• Do not use “srun” for serial executable (reduces overhead)
How to Run Debug and Interactive Jobs

• You can run small parallel jobs interactively

• Debug
  – Max 512 nodes, 30 min, run limit 1, queue limit 5
    `% salloc -N 20 -q debug -C knl,quad,cache -t 30:00`

• Interactive
  – Instant allocation (or reject), run limit 1, max walltime 4 hrs, up to 64 nodes on Cori (Haswell and KNL) per repo
    `% salloc -N 2 -q interactive -C knl,quad,cache -t 2:00:00`

• https://www.nersc.gov/users/computational-systems/cori/running-jobs/interactive-jobs/
Advanced Workflow Management

- Bundle jobs (multiple “srun”s in one script, sequential or simultaneously)
- Use Job Arrays for submitting and managing collections of similar jobs
- Use job dependency features to chain jobs that have dependency
- Use Burst Buffer for faster IO
- Use Shifter for jobs with custom user environment
- Run “xfer” for transferring to/from HPSS
- Run “bigmem” for large memory jobs
Bundle Jobs

Multiple Jobs Sequentially:

```bash
#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 100
#SBATCH -t 12:00:00
#SBATCH -J my_job
#SBATCH -o my_job.o%j
#SBATCH -L project,SCRATCH
#SBATCH -C haswell

srun -n 3200 ./a.out
srun -n 3200 ./b.out
srun -n 3200 ./c.out
```

- need to request largest number of nodes needed

Multiple Jobs Simultaneously:

```bash
#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 9
#SBATCH -t 12:00:00
#SBATCH -J my_job
#SBATCH -o my_job.o%j
#SBATCH -L project
#SBATCH -C haswell

srun -n 44 -N 2 -c2 --cpu_bind=cores ./a.out &
srun -n 108 -N 5 -c2 --cpu_bind=cores ./b.out &
srun -n 40 -N 2 -c2 --cpu_bind=cores ./c.out &
wait
```

- need to request total number of nodes needed
- No applications are shared on the same nodes
- make sure to use “&” (otherwise run in sequential) and “wait” (otherwise job exit immediately)
Job Arrays

# Submit a job array with index values between 0 and 31:
$ sbatch --array=0-31 -N1

# Submit a job array with index values of 1, 3, 5 and 7:
$ sbatch --array=1,3,5,7 -N1 -n2

# Submit a job array with index values between 1 and 7 with a step size of 2 (i.e. 1, 3, 5, 7)
$ sbatch --array=1-7:2 -N1 -q regular

#!/bin/bash -l
#SBATCH -q regular
#SBATCH -N 1
#SBATCH -t 00:05:00
#SBATCH --array=1-10
#SBATCH -L SCRATCH
#SBATCH -C haswell

cd test_${SLURM_ARRAY_JOB_ID}
srun ./mycode.exe

• Better managing jobs, not necessary faster turnaround
• Each array task is considered a single job for scheduling
• Use $SLURM_ARRAY_JOB_ID for each individual array task
Dependency Jobs

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 -q regular
#SBATCH -N 1
#SBATCH -t 00:30:00
#SBATCH -d afterok:5547
#SBATCH -C haswell
srun -n 16 -c 4 ./a.out

cori06% sbatch job2
Use Burst Buffer for Faster IO

- 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
- See this afternoon’s Burst Buffer talk for usage examples
Use Shifter

• Shifter is an open-source software stack that enables users to run custom environments on HPC systems.
• Compatible with the popular Docker container format so users can easily run Docker containers on NERSC systems.
• See this afternoon’s Shifter talk for usage examples.
xfer Jobs

```
#!/bin/bash -l
#SBATCH -M esedison
#SBATCH -q xfer
#SBATCH -t 12:00:00
#SBATCH -J my_transfer

#Archive run01 to HPSS
htar -cvf run01.tar run01
```

- Configured for the purpose of staging data from HPSS before run or archive result to HPSS after run
- Avoid large NERSC hours if done within large runs
- Runs on external login nodes, via Slurm Server “esedison” or ”escori”.
- Can submit jobs to the xfer QOS from inside another batch script:
  - Add to the end of batch script: “sbatch -M esedison -q xfer myarchive.sl”
#!/bin/bash –l
#SBATCH -M escori
#SBATCH -q bigmem
#SBATCH -N 1
#SBATCH -t 01:00:00
#SBATCH -J my_big_job
#SBATCH -L SCRATCH
#SBATCH --mem=250GB
srun -N 1 -n 1 my_big_executable

- Only exists on Cori.
- Runs on external login nodes, via Slurm Server “escori”
Process / Thread / Memory Affinity

• Correct process, thread and memory affinity is the basis for getting optimal performance on KNL. It is also essential for guiding further performance optimizations.
  – Process Affinity: bind MPI tasks to CPUs
  – Thread Affinity: bind threads to CPUs allocated to its MPI process
  – Memory Affinity: allocate memory from specific NUMA domains

• Our goal is to promote OpenMP4 standard settings for portability. For example, OMP_PROC_BIND and OMP_PLACES are preferred to Intel specific KMP_AFFINITY and KMP_PLACE_THREADS settings.
“numactl -H” displays NUMA domain info

68-core Quad Cache node:
NUMA Domain 0: all 68 cores (272 logic cores)

yunhe@cori01:> salloc -N 1 --qos=interactive -C knl,quad,cache -t 30:00
salloc: Granted job allocation 5291739

yunhe@nid02305:> numactl -H
available: 1 nodes (0)
node 0 cpus: 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 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246
247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 0 size: 96762 MB
node 0 free: 93067 MB
node distances:
node 0
  0: 10

- The quad,cache mode has only 1 NUMA node with all CPUs on the NUMA node 0 (DDR memory)
- The MCDRAM is hidden from the numactl -H command (it is a cache).
Can We Just Do a Naïve Srun?

Example: 16 MPI tasks x 8 OpenMP threads per task on a single 68-core KNL quad, cache node:

% export OMP_NUM_THREADS=8
% export OMP_PROC_BIND=spread
% export OMP_PLACES=threads
(other choice are “close”, ”master”, ”true”, ”false”)
(other choices are: cores, sockets, and various

ways to specify explicit lists, etc.)

% srun -n 16 ./xthi |sort -k4n,6n
Hello from rank 0, thread 0, on nid02304. (core affinity = 0)  (on physical core 8)
Hello from rank 0, thread 1, on nid02304. (core affinity = 144) (on physical core 25)
Hello from rank 0, thread 2, on nid02304. (core affinity = 17) (on physical core 42)
Hello from rank 0, thread 3, on nid02304. (core affinity = 161) (on physical core 59)
Hello from rank 0, thread 4, on nid02304. (core affinity = 34)  (on physical core 8)
Hello from rank 0, thread 5, on nid02304. (core affinity = 178) (on physical core 25)
Hello from rank 0, thread 6, on nid02304. (core affinity = 51)  (on physical core 42)
Hello from rank 0, thread 7, on nid02304. (core affinity = 195) (on physical core 59)
Hello from rank 1, thread 0, on nid02304. (core affinity = 0)  (on physical core 8)
Hello from rank 1, thread 1, on nid02304. (core affinity = 144) (on physical core 25)

It is a mess!
Importance of -c and --cpu_bind Options

• The reason: 68 is not divisible by #MPI tasks!
  – Each MPI task is getting 68x4/#MPI tasks of logical cores as the domain size
  – MPI tasks are crossing tile boundaries

• Let’s set number of logical cores per MPI task (-c) manually by wasting extra 4 cores on purpose, which is 256/#MPI tasks.
  – Meaning to use 64 cores only on the 68-core KNL node, and spread the logical cores allocated to each MPI task evenly among these 64 cores.
  – Now it looks good!
  – % srun -n 16 -c 16 --cpu_bind=cores ./xthi
    Hello from rank 0, thread 0, on nid09244. (core affinity = 0)
    Hello from rank 0, thread 1, on nid09244. (core affinity = 136)
    Hello from rank 0, thread 2, on nid09244. (core affinity = 1)
    Hello from rank 0, thread 3, on nid09244. (core affinity = 137)
**Now It Looks Good!**

<table>
<thead>
<tr>
<th>Process/thread affinity are good! (Marked first 6 and last MPI tasks only)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MPI rank 0</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MPI rank 1</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MPI rank 2</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MPI rank 3</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MPI rank 4</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MPI rank 5</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MPI rank 15</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

And so on for other MPI tasks and threads ....
Essential Runtime Settings for Process/Thread Affinity

- Use `srun -c` and `--cpu_bind` flags to bind tasks to CPUs
  - `-c <n>` (or `--cpus-per-task=n`) allocates (reserves) `n` CPUs per task (process). It helps to evenly spread MPI tasks
  - Use `--cpu_bind=cores` (no hyperthreads) or `--cpu_bind=threads` (if hyperthreads are used)

- Use OpenMP envs, `OMP_PROC_BIND` and `OMP_PLACES` to fine pin each thread to a subset of CPUs allocated to the host task
  - Different compilers may have different implementations
  - The following provide compatible thread affinity among Intel, GNU and Cray compilers:

    - `OMP_PROC_BIND=true` # Specifying threads may not be moved between CPUs
    - `OMP_PLACES=threads` # Specifying a thread should be placed on a single CPU
Sample Job Script to Run on KNL quad, cache Nodes

Sample Job script (**MPI+OpenMP**)  
#!/bin/bash -l  
#SBATCH -N 2  
#SBATCH -q regular  
#SBATCH -t 1:00:00  
#SBATCH -L SCRATCH  
#SBATCH -C knl,quad,cache

export OMP_NUM_THREADS=4  
	srun -n 128 -c 4 --cpu_bind=cores ./a.out

This job script requests 2 KNL quad, cache nodes to run 128 MPI tasks, 64 MPI tasks per node, allocating 4 CPUs per task, and binds each task to the 4 CPUs allocated within each core. The 4 OpenMP threads per MPI task are bound to the 4 CPUs in the core.

Illustration Courtesy of Zhengji Zhao, NERSC
Sample Job Script to Run on KNL quad, cache Nodes

Sample Job script (MPI+OpenMP)

```
#!/bin/bash

#SBATCH -N 2
#SBATCH -q regular
#SBATCH -t 1:00:00
#SBATCH -L SCRATCH
#SBATCH -C knl_quad.cache

export OMP_PROC_BIND=true
export OMP_PLACES=threads
export OMP_NUM_THREADS=4

srun -n 128 -c 4 --cpu_bind=cores ./a.out
```

Process and thread affinity

Illustration Courtesy of Zhengji Zhao, NERSC

With the above two OpenMP envs, each thread is now pinned to a single CPU within each core
Essential Runtime Settings for MCDRAM
Memory Affinity

• In cache mode, no special setting is needed to use MCDRAM
• In flat mode, using quad,flat as an example: NUMA node 1 is MCDRAM *(need to request a reservation to use this mode)*
• `#SBATCH -C knl,quad,flat`
  – Enforced memory mapping to MCDRAM
  – If using >16 GB, malloc will fail
    `srun -n 16 -c 16 -cpu_bind=cores --mem_bind=map_mem:1 ./a.out`
    Or:
    `srun -n 16 -c 16 -cpu_bind=cores ... numactl -m 1 ./a.out`
  – Preferred memory mapping to MCDRAM:
  – If using >16 GB, malloc will spill to DDR
    `srun -n 16 -c 16 -cpu_bind=cores
     --mem_bind=preferred,map_mem:1 ./a.out`
    Or:
    `srun -n 16 -c 16 -cpu_bind=cores numactl -p 1 ./a.out`
Affinity Verification Methods (1)

- NERSC has provided pre-built binaries from a Cray code (xthi.c) to display process thread affinity: `check-mpi.intel.cori`, `check-mpi.cray.cori`, `check-hybrid.intel.cori`, etc.

```bash
% srun -n 32 -c 8 --cpu_bind=cores check-mpi.intel.cori|sort -nk 4
Hello from rank 0, on nid02305. (core affinity = 0,1,68,69,136,137,204,205)
Hello from rank 1, on nid02305. (core affinity = 2,3,70,71,138,139,206,207)
Hello from rank 2, on nid02305. (core affinity = 4,5,72,73,140,141,208,209)
Hello from rank 3, on nid02305. (core affinity = 6,7,74,75,142,143,210,211)
```

- Intel compiler has a run time environment variable `KMP_AFFINITY`, when set to "verbose”:

  ```
  OMP: Info #242: KMP_AFFINITY: pid 255705 thread 0 bound to OS proc set {55}
  OMP: Info #242: KMP_AFFINITY: pid 255660 thread 1 bound to OS proc set {10,78}
  OMP: Info #242: OMP_PROC_BIND: pid 255660 thread 1 bound to OS proc set {78} ...
  ```

- Cray compiler has a similar env `CRAY_OMP_CHECK_AFFINITY`, when set to "TRUE”:

  ```
  [CCE OMP: host=nid00033 pid=14506 tid=17606 id=1] thread 1 affinity: 90
  [CCE OMP: host=nid00033 pid=14510 tid=17597 id=1] thread 1 affinity: 94 ...
  ```
Affinity Verification Methods (2)

- Use `srun` flag `--cpu_bind=verbose` to check thread affinity
  - Need to read the cpu masks in hexadecimal format
- Use `srun` flag `--mem_bind=verbose,<type>` to check memory affinity
- Use the `numastat -p <PID>` command to confirm while a job is running
NERSC Job Script Generator (check new script)

https://my.nersc.gov/script_generator.php
Monitoring Your Job

• Once your job is submitted, it enters the queue and will start when resources are available.

• Overall job priorities are a combination of QOS, queue wait time, job size, wall time request (and fair share).

• You can monitor it with:
  – sqs
  – squeue

On the web:

https://my.nersc.gov
https://www.nersc.gov/users/live-status/ : “Queue Look”
https://www.nersc.gov/users/job-logs-and-analytics/completed-jobs/
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 Batch Queue Script

### sqs

```
yunhe@cori05:~$ sqs -u fxxxx |more
```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>ST</th>
<th>USER</th>
<th>NAME</th>
<th>NODES</th>
<th>REQUESTED</th>
<th>USED</th>
<th>SUBMIT</th>
<th>QOS</th>
<th>SCHEDULED_START</th>
<th>FEATURES</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>110901xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>mxxxx</td>
<td>1536</td>
<td>5:00</td>
<td>0:00</td>
<td>2018-03-20T10:49:23</td>
<td>regular_0</td>
<td>2018-03-22T06:30:00</td>
<td>haswell</td>
<td>Resources</td>
</tr>
<tr>
<td>110901xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>run.xxx*</td>
<td>1537</td>
<td>20:00</td>
<td>0:00</td>
<td>2018-03-20T10:51:03</td>
<td>regular_0</td>
<td>2018-03-22T06:30:00</td>
<td>haswell</td>
<td>Resources</td>
</tr>
<tr>
<td>110823xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>gxxxxx</td>
<td>300</td>
<td>30:00</td>
<td>0:00</td>
<td>2018-03-19T23:05:24</td>
<td>regular_1</td>
<td>avail_in_-1.6_days</td>
<td>haswell</td>
<td>Priority</td>
</tr>
<tr>
<td>110823xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>run-xx</td>
<td>768</td>
<td>20:00</td>
<td>0:00</td>
<td>2018-03-19T23:05:33</td>
<td>regular_1</td>
<td>avail_in_-1.6_days</td>
<td>haswell</td>
<td>Priority</td>
</tr>
<tr>
<td>110901xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>Jxxxxx</td>
<td>768</td>
<td>10:00</td>
<td>0:00</td>
<td>2018-03-20T10:51:33</td>
<td>regular_1</td>
<td>avail_in_-2.1_days</td>
<td>haswell</td>
<td>Priority</td>
</tr>
<tr>
<td>110921xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>pxxxxxx-xx*</td>
<td>150</td>
<td>30:00</td>
<td>0:00</td>
<td>2018-03-20T12:42:38</td>
<td>regular_1</td>
<td>avail_in_-2.1_days</td>
<td>haswell</td>
<td>Priority</td>
</tr>
<tr>
<td>111046xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>hxxxxxx</td>
<td>32</td>
<td>20:00</td>
<td>0:00</td>
<td>2018-03-21T01:20:09</td>
<td>regular_1</td>
<td>avail_in_-2.7_days</td>
<td>haswell</td>
<td>Priority</td>
</tr>
<tr>
<td>110823xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>xxxxx</td>
<td>1536</td>
<td>20:00</td>
<td>0:00</td>
<td>2018-03-19T23:05:04</td>
<td>regular_0</td>
<td>N/A</td>
<td>haswell</td>
<td>JobHeldUser</td>
</tr>
<tr>
<td>110823xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>axxxxxx*</td>
<td>1536</td>
<td>30:00</td>
<td>0:00</td>
<td>2018-03-19T23:05:16</td>
<td>regular_0</td>
<td>N/A</td>
<td>haswell</td>
<td>JobHeldUser</td>
</tr>
<tr>
<td>111152xx</td>
<td>PD</td>
<td>fxxxx</td>
<td>run.xxx</td>
<td>769</td>
<td>2:00:00</td>
<td>0:00</td>
<td>2018-03-21T09:39:29</td>
<td>regular_1</td>
<td>avail_in_-3.0_days</td>
<td>knl&amp;quad&amp;cache</td>
<td>None</td>
</tr>
</tbody>
</table>

---

Note: The SCHEDULED_START column shows either the estimated start time provided by Slurm, or that an estimated start time will be "available in xxx hrs/days", meaning your job won't be scheduled by then (due to priority) unless via backfill, and the actual job start time could still be some period of time after then.

Second Note: Upcoming system maintenance on 03-22-2018 07:00 AM, SCHEDULED_START will be updated after the maintenance is completed.

---

- sqs combines information from squeue and sinfo, provides formatted output with various options
- It has many options, such as no running jobs, yes or no for a specific qos, wider with more fields, compact format, custom fields, expand array jobs, csv and json output, etc. See “sqs --help” or “man sqs” for more details.
- The scheduled_start gives users an estimate for when their jobs will start (or when will be considered for scheduling based on distance to priority threshold).
sinfo: Show Available Nodes

• % sinfo --format="%F %b" for features of available nodes (or sinfo --format="%C %b" for CPUs).
  – A/I/O/T (allocated/idle/other/total)

```bash
yunhe@cori02:~> sinfo --format="%F %b"
NODES(A/I/O/T) ACTIVE_FEATURES
2024/293/71/2388 haswell
  0/0/2/2 knl
0/0/1/1 knl,flat,quad
9275/373/37/9685 knl,cache,quad
```

All these are quad,cache KNL nodes
scontrol: Show Job Details

- % scontrol show job <jobid> for details of a job

```bash
yunhe@cori01:~> scontrol show job 11082385
JobId=11082385 JobName=gtc19200
   UserId=fbench(42034) GroupId=fbench(42034) MCS_label=N/A
   Priority=66669 Nice=0 Account=mpcray QOS=regular_1
   JobState=PENDING Reason=Priority Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:30:00 TimeMin=N/A
   StartTime=Unknown EndTime=Unknown Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   LastSchedEval=2018-03-21T06:18:17
   Partition=regular AllocNode:Sid=cori01:159414
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null)
   NumNodes=300-300 NumCPUs=300 NumTasks=300 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=300,node=300
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=haswell DelayBoot=00:00:00
   Gres=craynetwork:1 Reservation=(null)
   OverSubscribe=NO Contiguous=0 Licenses=cscratch1:1 Network=(null)
   Command=/global/cscratch1/sd/fbench/GTC/run/run19200opt
   WorkDir=/global/cscratch1/sd/fbench/GTC/run
   StdErr=/global/cscratch1/sd/fbench/GTC/run/slurm-11082385.out
   StdIn=/dev/null
   StdOut=/global/cscratch1/sd/fbench/GTC/run/slurm-11082385.out
   Power=
```
### Sacct: Query Completed and Pending Jobs Information

```bash
yunhe@cori05:~> sacct -u fbench -S 2018-03-19 -E 2018-03-20 -o user,jobid,start,end,elapsed,timelimit,nnodes,exitcode,state -X
```

<table>
<thead>
<tr>
<th>User</th>
<th>JobID</th>
<th>Start</th>
<th>End</th>
<th>Elapsed</th>
<th>Timelimit</th>
<th>NNodes</th>
<th>ExitCode</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbench</td>
<td>10971468</td>
<td>2018-03-19T10:51:07</td>
<td>2018-03-19T10:56:30</td>
<td>00:05:23</td>
<td>00:30:00</td>
<td>32</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>11006329</td>
<td>2018-03-19T14:06:28</td>
<td>2018-03-19T14:06:56</td>
<td>00:00:28</td>
<td>00:30:00</td>
<td>150</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>11038987</td>
<td>2018-03-19T23:19:34</td>
<td>2018-03-19T23:24:54</td>
<td>00:05:20</td>
<td>00:30:00</td>
<td>32</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>11062282</td>
<td>2018-03-19T03:35:01</td>
<td>2018-03-19T03:37:56</td>
<td>00:02:55</td>
<td>00:30:00</td>
<td>1</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>11076856</td>
<td>2018-03-19T16:50:30</td>
<td>2018-03-19T16:53:10</td>
<td>00:02:40</td>
<td>00:30:00</td>
<td>3</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>11076857</td>
<td>2018-03-19T16:50:30</td>
<td>2018-03-19T17:04:48</td>
<td>00:14:18</td>
<td>00:30:00</td>
<td>3</td>
<td>1:0</td>
<td>FAILED</td>
</tr>
<tr>
<td>fbench</td>
<td>11082385</td>
<td>Unknown</td>
<td>Unknown</td>
<td>00:00:00</td>
<td>00:30:00</td>
<td>300</td>
<td>0:0</td>
<td>PENDING</td>
</tr>
<tr>
<td>fbench</td>
<td>11082394</td>
<td>Unknown</td>
<td>Unknown</td>
<td>00:00:00</td>
<td>00:20:00</td>
<td>768</td>
<td>0:0</td>
<td>PENDING</td>
</tr>
<tr>
<td>fbench</td>
<td>11082403</td>
<td>2018-03-20T00:19:32</td>
<td>2018-03-20T00:32:33</td>
<td>00:13:01</td>
<td>00:30:00</td>
<td>63</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
</tbody>
</table>

- Maximum query duration is one month (subject to change)
- Detailed job steps info will be displayed without “-X” flag
- Many more job fields can be queried. See “man sacct”
How Your Jobs Are Charged (1)

• Unit: NERSC Hours
• Each machine has a base charge per node hour used:
  – Edison: 48
  – Cori Haswell: 80
  – Cori KNL: 96
• Modification to base charge by QOS used:
  – premium: 2.0
  – normal: 1.0 (default)
  – low: 0.5
  – scavenger: 0
• On Cori KNL:
  – Jobs requesting 1024 or more nodes get a 20% discount
• 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” QOS*.
  
  – Example: 4 Cori Haswell nodes, run for 1 hour with “premium” QOS
    NERSC hours = 4 * 1 hour * 80 * 2 = 640
  
  – “shared” jobs are charged with # of 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
Which System to Run My Jobs

- Queue configuration and policies are subject to further tuning for max throughput and system utilization.
- Factors to consider: queue wait time, throughput, charging, code readiness
- Remember to compile separately for each type of compute nodes
- **Edison**: Large stable machine, low charge factor
- Cori Haswell (also known as the "Cori Data Partition") system is designed to accelerate data-intensive applications
- **Cori KNL**: Large capability and performance (Q: Are your codes optimized and ready for KNL?)
- Jobs use 1024+ nodes on Cori KNL get 20% charging discount
- “shared” and ”realtime” available on Edison and Cori Haswell
- “interactive” available on Cori Haswell and KNL
- “bigmem” available on Cori (run on external login nodes)
# Edison Queue Policy (as of March 2018)

<table>
<thead>
<tr>
<th>QOS(^1)</th>
<th>Nodes</th>
<th>Physical Cores</th>
<th>Max Wallclock</th>
<th>Run Limit</th>
<th>Submit Limit</th>
<th>Relative Priority (lower is higher priority)</th>
<th>NERSC Hours Charged per Node Per Hour(^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>1-512</td>
<td>1-12,288</td>
<td>30 mins</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>regular</td>
<td>1-5,586(^4)</td>
<td>1-134,064</td>
<td>48 hrs</td>
<td>-</td>
<td>5,000</td>
<td>3</td>
<td>48</td>
</tr>
<tr>
<td>premium</td>
<td>1-5,586</td>
<td>1-134,064</td>
<td>48 hrs</td>
<td>-</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>shared(^5)</td>
<td>1</td>
<td>1-12</td>
<td>48 hrs</td>
<td>-</td>
<td>10,000</td>
<td>3</td>
<td>2 x (no. of cores used)</td>
</tr>
<tr>
<td>realtime(^6)</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>1 (special permission)</td>
<td>--</td>
</tr>
<tr>
<td>scavenger(^2)</td>
<td>1-5,586</td>
<td>1-134,064</td>
<td>48 hrs</td>
<td>-</td>
<td>5,000</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>xfer(^7)</td>
<td>-</td>
<td>-</td>
<td>48 hrs</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Cori Queue Policy (as of March 2018) (1)

<table>
<thead>
<tr>
<th>QOS</th>
<th>Max Wall Time</th>
<th>Nodes</th>
<th>Submit Limit</th>
<th>Run Limit</th>
<th>Relative Priority</th>
<th>Charge Factor</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>interactive</td>
<td>4 hrs</td>
<td>1-64 nodes (max 64 total per repo, Haswell and KNL combined)</td>
<td>1</td>
<td>1 (max nodes per repo = 64)</td>
<td>3</td>
<td>Haswell: 80 KNL: 96</td>
<td>NERSC accounts</td>
</tr>
<tr>
<td>debug</td>
<td>30 min</td>
<td>Haswell: 1-64 KNL: 1-512</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>Haswell: 80 KNL: 96</td>
<td>NERSC accounts</td>
</tr>
<tr>
<td>regular</td>
<td>48 hrs</td>
<td>Haswell: 1-1,932 KNL: 1-8,991</td>
<td>--</td>
<td>--</td>
<td>4</td>
<td>Haswell: 80 KNL: 96 (&lt;1,024 nodes) KNL: 76.8 (≥1,024 nodes)</td>
<td>NERSC accounts</td>
</tr>
<tr>
<td>premium</td>
<td>48 hrs</td>
<td>Haswell: 1-1,932 KNL: 1-8,991</td>
<td>5</td>
<td>--</td>
<td>2</td>
<td>Haswell: 160 KNL: 192 (&lt;1,024 nodes) KNL: 153.6 (≥1,024 nodes)</td>
<td>NERSC accounts</td>
</tr>
<tr>
<td>shared</td>
<td>48 hrs</td>
<td>1 node, 1-16 cores (Haswell only)</td>
<td>10,000</td>
<td>--</td>
<td>3</td>
<td>2.5*num_cores_used</td>
<td>NERSC accounts</td>
</tr>
<tr>
<td>QoS</td>
<td>Max Wall Time</td>
<td>Nodes</td>
<td>Submit Limit</td>
<td>Run Limit</td>
<td>Relative Priority</td>
<td>Charge Factor</td>
<td>Access</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>-------------------------------------</td>
<td>--------------</td>
<td>-----------</td>
<td>------------------</td>
<td>-------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>genepool</td>
<td>72 hrs</td>
<td>1-192 nodes (Haswell only)</td>
<td>500</td>
<td>--</td>
<td>3</td>
<td>80</td>
<td>JGI accounts</td>
</tr>
<tr>
<td>genepool_shared</td>
<td>72 hrs</td>
<td>1 node, 1-16 cores (Haswell only)</td>
<td>500</td>
<td>--</td>
<td>3</td>
<td>2.5*num_cores_used</td>
<td>JGI accounts</td>
</tr>
<tr>
<td>realtime</td>
<td>custom</td>
<td>custom (Haswell only)</td>
<td>custom</td>
<td>custom</td>
<td>1</td>
<td>2.5*num_cores_used</td>
<td>special request</td>
</tr>
<tr>
<td>special</td>
<td>special</td>
<td>special</td>
<td>special</td>
<td>special</td>
<td>special</td>
<td>Haswell: 80</td>
<td>special permission</td>
</tr>
<tr>
<td>scavenger</td>
<td>48 hrs</td>
<td>Haswell: 1-1,932 KNL: 1-8,991</td>
<td>---</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>not selectable</td>
</tr>
<tr>
<td>xfer</td>
<td>12 hrs</td>
<td>1 (run on login nodes)</td>
<td>100</td>
<td>15</td>
<td>--</td>
<td>0</td>
<td>NERSC accounts</td>
</tr>
<tr>
<td>bigmem</td>
<td>72 hrs</td>
<td>1-2 (run on login nodes)</td>
<td>5</td>
<td>1</td>
<td>--</td>
<td>0</td>
<td>NERSC accounts</td>
</tr>
</tbody>
</table>
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
- Queue wait time statistics
  - https://www.nersc.gov/users/queues/queue-wait-times/
More Information

NERSC Web pages:

• Edison
  http://www.nersc.gov/users/computational-systems/edison/running-jobs/

• Cori
  http://www.nersc.gov/users/computational-systems/cori/running-jobs/

• Cori KNL:
  http://www.nersc.gov/users/computational-systems/cori/running-jobs/example-batch-scripts-for-knl/

Contact NERSC Consulting:

– Toll-free 800-666-3772
– 510-486-8611, option #3
– Email consult@nersc.gov
Thank you.