Outline

● Running Jobs Introductions
● Batch Script Examples
● Advanced Running Jobs Options
● KNL Process/Thread/Memory Affinity
● Monitoring Jobs
● Running Jobs Best Practices
Running Jobs Introductions
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
- Debug jobs are supported for up to 30 min
- Batch interactive jobs are supported for up to 4 hrs
- Typical run times are a few to 10s of hours
  - Limits are necessary because of MTBF and the need to accommodate 7,000 users’ jobs
Login Nodes and Compute Nodes

- **Login nodes (external)**
  - Edit files, compile codes, submit batch jobs, etc.
  - Run short, serial utilities and applications
  - Cori has Haswell login nodes

- **Compute nodes**
  - Execute your application
  - Dedicated resources for your job
  - Cori has Haswell and KNL compute nodes
  - Binaries built for Haswell can run on KNL nodes, but not vice versa
Each Cori Haswell node has 2 Intel Xeon 16-core Haswell processors
- 2 NUMA domains (sockets) per node, 16 cores per NUMA domain. 2 hardware threads per 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
A Cori KNL node has 68 cores/272 CPUs, 96 GB DDR memory, 16 GB high bandwidth on package memory (MCDRAM)

Default mode is: quad, cache

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
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
- A batch job will request resources about which qos, which type of compute nodes, how many nodes, and for how long, etc.
- 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

**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

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

**Other Compute Nodes allocated to the job**:}

```bash
sbatch
or
salloc
```

```bash
srun
```

![Diagram showing login node, head compute node, and other compute nodes with arrows indicating `sbatch`, `srun`, and `srun` commands.

NERSC

Berkeley Lab

U.S. Department of Energy

Office of Science
Batch Script Examples
My First “Hello World” Program

my_batch_script:

#!/bin/bash
#SBATCH -q debug
#SBATCH -N 2
#SBATCH -t 10:00
#SBATCH -C haswell
#SBATCH -L SCRATCH
#SBATCH -J myjob
srun -n 64 ./helloWorld

To run via batch queue
% sbatch my_batch_script

To run via interactive batch
% salloc -N 2 -q interactive -C haswell -t 10:00
<wait_for_session_prompt. Land on a compute node>
% srun -n 64 ./helloWorld
Sample Cori Haswell Batch Script

```bash
#!/bin/bash
#SBATCH --qos=regular
#SBATCH --nodes=4
#SBATCH --time=1:00:00
#SBATCH --constraint=haswell
#SBATCH --license=SCRATCH
#SBATCH --jobname=myjob

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
- Environment is automatically imported
Sample Cori Haswell Batch Script

```bash
#!/bin/bash
#SBATCH --qos=regular
#SBATCH --nodes=4
#SBATCH --time=1:00:00
#SBATCH --constraint=haswell
#SBATCH --license=SCRATCH
#SBATCH --jobname=myjob

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

Job directives: instructions for the batch system

- Can use long name or short name (see next slide) to request resources
- 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
- More optional SBATCH keywords
Sample Cori Haswell Batch Script - MPI

```
#!/bin/bash
#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
#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 ./mycode.exe” is OK
Sample Cori Haswell Batch Script - MPI

```bash
#!/bin/bash
#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
```

- 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

```bash
#!/bin/bash
#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
- …
Use “shared” QOS to Run Serial Jobs

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

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

- Only available on Cori Haswell
- Small parallel job that use less than a full node can also run in the “shared” partition
- https://docs.nersc.gov/jobs/best-practices/#serial-jobs

- Do not specify #SBATCH -N"
- Default “#SBATCH -n” is 1
- Default memory is 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 on dedicated nodes.

- **Debug**
  - Max 512 nodes, up to 30 min, run limit 2, submit limit 5
  
  ```bash
  % salloc -N 20 -q debug -C haswell -t 30:00
  ```

- **Interactive** (highly recommend to use this!!)
  - Instant allocation (get nodes in 5 min or reject), run limit 2, submit limit 2
  - Max walltime 4 hrs, up to 64 nodes on Cori (Haswell and KNL combined) per project

  ```bash
  % salloc -N 2 -q interactive -C knl -t 2:00:00
  ```

- More information (such as find out who in your project is using)

  - [https://docs.nersc.gov/jobs/examples/#interactive](https://docs.nersc.gov/jobs/examples/#interactive)
  - [https://docs.nersc.gov/jobs/interactive/](https://docs.nersc.gov/jobs/interactive/)
Advanced Running Jobs Options
Advanced Running Jobs Options

- Bundle jobs (multiple “srun”s in one script, sequentially or simultaneously)
- Use Job Arrays to manage collections of similar jobs
- Use job dependency features to chain jobs
- Run variable-time jobs to run longer jobs
- Use workflow tools to manage jobs
- Use Burst Buffer for faster IO
- Use Shifter for jobs with custom user environment
- Use “xfer” for transferring to/from HPSS
- Use “bigmem” for large memory jobs
Bundle Jobs

Multiple Jobs Sequentially:
```
#!/bin/bash
#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
```

Multiple Jobs Simultaneously:
```
#!/bin/bash
#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 largest 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)
- [https://docs.nersc.gov/jobs/examples/#multiple-parallel-jobs-sequentially](https://docs.nersc.gov/jobs/examples/#multiple-parallel-jobs-sequentially)
- [https://docs.nersc.gov/jobs/examples/#multiple-parallel-jobs-simultaneously](https://docs.nersc.gov/jobs/examples/#multiple-parallel-jobs-simultaneously)
Job Arrays

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

https://docs.nersc.gov/jobs/examples/#job-arrays
Dependency Jobs

cori% sbatch job1
Submitted batch job 1655447

cori06% sbatch --dependency=afterok:165547 job2
or
cori06% sbatch --dependency=afterany:165547 job2

https://docs.nersc.gov/jobs/examples/#dependencies

cori06% sbatch job1
submitted batch job 1655447

cori06% cat job2
#!/bin/bash
#SBATCH -q regular
#SBATCH -N 1
#SBATCH -t 1:30:00
#SBATCH -d afterok:1655447
#SBATCH -C haswell
srun -n 16 -c 4 ./a.out

cori06% sbatch job2
Variable Time Jobs

```
#!/bin/bash
#SBATCH -q regular
#SBATCH -C haswell
#SBATCH -N 2
#SBATCH --comment=96:00:00
#SBATCH --time-min=2:00:00
#SBATCH --time=48:00:00
#SBATCH --signal=B:USR1@60
#SBATCH --requeue
ckpt_command=my_ckpt_script    (# or empty)
. /usr/common/software/variable-time-job/setup.sh
requeue_job func_trap USR1
srun -n 8 -c 16 --cpu-bind=cores ../test.exe &
wait
```

- Allows to run multiple jobs with accumulated run time longer than max allowed wall time
- You may get run time longer than 2 hrs but shorter than 48 hrs at a time in this example
- Job needs to have checkpoint/restart capability
- Individual jobs will be terminated with signal USR1 before time limit is reached
- Pre-terminated jobs will be requeued

[https://docs.nersc.gov/jobs/examples/#variable-time-jobs](https://docs.nersc.gov/jobs/examples/#variable-time-jobs)
Use “flex” QOS to Run Variable Time Jobs

- For user jobs that can produce useful work with a relatively short amount of run time before terminating, such as jobs capable of checkpointing and restarting where left off.
- Helps to improve throughput by submitting jobs that can fit into “backfill holes” in Slurm job scheduling.
- Requires to use “--time-min” of <= 2hrs, max “--time” is 48 hrs.
- 75% charging discount as of June 2020 (subject to change)
  - Available for KNL only. More info at https://docs.nersc.gov/jobs/examples/#using-the-flex-qos-for-charging-discount-for-variable-time-jobs-on-knl
  - https://docs.nersc.gov/jobs/policy/#flex
Use “overrun” QOS When Project is Out of Allocation

- When a project has zero or negative balance, a user can submit to the overrun qos (or “overrun_shared”) qos explicitly.
- Lowest priority
- Zero charge
- Requires to use “--time-min” of <= 4hrs
  - sbatch -q overrun --time-min=01:30:00 my_batch_script.sl
- More info at
  - [https://docs.nersc.gov/jobs/policy/#overrun](https://docs.nersc.gov/jobs/policy/#overrun)
Use Workflow Management Tools

- These tools can help data-centric science to automate moving data, multi-step processing, and visualization at scales. Can manage to run large number of similar jobs.
- Please do not do below!
  
  ```
  for i = 1, 10000
      srun -n 1 ./a.out
  ```

  which is inefficient and also overwhelms the scheduler.
- Available workflow tools include: GNU parallel, Taskfarmer, Fireworks, etc.
- See this afternoon’s Workflow talk for usage examples
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
  - More info at http://www.nersc.gov/users/computational-systems/cori/burst-buffer/
  - https://docs.nersc.gov/jobs/examples/#burst-buffer
- See this afternoon’s Burst Buffer talk for usage examples
Use Shifter for Custom Environment

- 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
- More info at
  - [https://docs.nersc.gov/development/shifter/how-to-use/](https://docs.nersc.gov/development/shifter/how-to-use/)
- See this afternoon’s Shifter talk for usage examples
xfer Jobs

#!/bin/bash
#SBATCH -M escori
#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 wasting NERSC hours if done within large runs
- Runs on external login nodes, via Slurm Server "escori".
- Can submit jobs to the xfer QOS from inside another batch script:
  - Add to the end of batch script: “sbatch -M escori -q xfer myarchive.sl”
- [https://docs.nersc.gov/jobs/examples/#xfer-queue](https://docs.nersc.gov/jobs/examples/#xfer-queue)
bigmem Jobs

```bash
#!/bin/bash
#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_exe
```

- Runs on external login nodes, via Slurm Server “escori”
- Node is shared among multiple users by default
- Can request exclusive node if needed to run with multiple threads
  - add `#SBATCH --exclusive`, and use `srun -N 1 -c 32 ./my_big_exe`
- [https://docs.nersc.gov/jobs/examples/#large-memory](https://docs.nersc.gov/jobs/examples/#large-memory)
KNL Process / Thread / Memory Affinity
Process / Thread / Memory Affinity

- Correct process, thread and memory affinity is the basis for getting optimal performance on Cori Haswell and 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 OpenMP standard settings for portability.
  - OMP_PROC_BIND and OMP_PLACES are preferred to Intel specific KMP_AFFINITY and KMP_PLACE_THREADS settings.

- [https://docs.nersc.gov/jobs/affinity/](https://docs.nersc.gov/jobs/affinity/)
Can We Just Do a Naive 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  (other choice are “close”, ”master”, ”true”, ”false”)
% export OMP_PLACES=threads     (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)
Hello from rank 0, thread 1, on nid02304. (core affinity = 144)  (on physical core 8)
Hello from rank 0, thread 2, on nid02304. (core affinity = 17)
Hello from rank 0, thread 3, on nid02304. (core affinity = 161)  (on physical core 25)
Hello from rank 0, thread 4, on nid02304. (core affinity = 34)
Hello from rank 0, thread 5, on nid02304. (core affinity = 178)  (on physical core 42)
Hello from rank 0, thread 6, on nid02304. (core affinity = 51)
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)
Hello from rank 1, thread 1, on nid02304. (core affinity = 144)

It is a mess!  thread 0 for rank 0, and thread 1 for rank 1 are on same physical core 0
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
- Set number of logical cores per MPI task (-c) manually by wasting extra 4 cores on purpose: $256/#\text{MPI\_tasks\_per\_node}$.
  - 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) (on physical core 0)
    Hello from rank 0, thread 2, on nid09244. (core affinity = 1)
    Hello from rank 0, thread 3, on nid09244. (core affinity = 137) (on physical core 1)
Now It Looks Good!

**Process/thread affinity are good! (Marked first 6 and last MPI tasks only)**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI rank 0</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
</tr>
<tr>
<td>MPI rank 1</td>
<td>138</td>
<td>137</td>
<td>138</td>
<td>139</td>
<td>140</td>
<td>141</td>
<td>142</td>
<td>143</td>
<td>144</td>
<td>145</td>
<td>146</td>
<td>147</td>
<td>148</td>
<td>149</td>
<td>150</td>
<td>151</td>
<td>152</td>
<td>153</td>
</tr>
<tr>
<td>MPI rank 2</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>204</td>
</tr>
<tr>
<td>MPI rank 3</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>MPI rank 4</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>MPI rank 5</td>
<td>154</td>
<td>155</td>
<td>156</td>
<td>157</td>
<td>158</td>
<td>159</td>
<td>160</td>
<td>161</td>
<td>162</td>
<td>163</td>
<td>164</td>
<td>165</td>
<td>166</td>
<td>167</td>
<td>168</td>
<td>169</td>
<td>170</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
<td>188</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</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 `n` CPUs per MPI task (process).
  - It helps to evenly spread MPI tasks, can use up to `n` OpenMP threads per MPI task.
  - Use `--cpu-bind=cores` (no hyperthreads) or `--cpu-bind=threads` (if hyperthreads are used)

- Use OpenMP envs: `OMP_PROC_BIND`, `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` # Specify threads may not be moved between CPUs
  - `OMP_PLACES=threads` # Specif a thread should be placed on a single CPU
Sample Job Script to Run on KNL Nodes

Sample Job script (MPI+OpenMP)

```bash
#!/bin/bash -l
#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
```

With the above two OpenMP envs, each thread is now pinned to a single CPU within each core.

Illustration Courtesy of Zhengji Zhao, NERSC
Affinity Verification Methods

- 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)
```

- OpenMP 5.0 has OMP_DISPLAY_AFFINITY and OMP_AFFINITY_FORMAT
  - Available in Intel compiler >= 18.0.5, gcc >= 9.0, and CCE >= 9.0.0
  ```bash
  % export OMP_DISPLAY_AFFINITY=true
  % export OMP_AFFINITY_FORMAT="host=%H, pid=%P, thread_num=%n, thread affinity=%A"
  host=nid02496, pid=150147, thread_num=0, thread affinity=0
  host=nid02496, pid=150147, thread_num=1, thread affinity=4
  ```
NERSC Job Script Generator

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

- 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 with:
  - `sqs`
  - `squeue`
  - `sacct`
- On the web:
  - [https://my.nersc.gov](https://my.nersc.gov) — Cori Queues, Queue backlogs, Queue Wait Times (statistics data)
  - [https://www.nersc.gov/users/live-status/](https://www.nersc.gov/users/live-status/) — Queue Look
  - [https://iris.nersc.gov](https://iris.nersc.gov) — the “Jobs” tab
By default, “squeue” displays all users jobs.

- Use “squeue -u” to display your own jobs.
- See “squeue --help” or “man squeue” for more details.
By default, “sqs” displays your own jobs. Use “sqs -a” to display all users jobs.

See “sqs --help” for more details.

sqs2 is a simplified NERSC wrapper for the Slurm "squeue" command with a chosen default format. It is more flexible, takes all allowed flags in “squeue”.

“sqs2” will be renamed to “sqs” in July.
scontrol: Show Job Details

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

```
JobId=31610730 JobName=mpi4py-import-cori-haswell-scratch-003
UserId=fbench(42034) GroupId=fbench(42034) MCS_label=N/A
Priority=66295 Nice=0 Account=nstaff QOS=regular_1
JobState=PENDING Reason=Nodes_required_for_job_are_DOWN,_DRAINED_or_reserved_for_jobs_in_higher_priority_partitions Dependency=(null)
RunTime=00:00:00 TimeLimit=00:30:00 TimeMin=N/A
```

```
NumNodes=3-3 NumCPUs=96 NumTasks=96 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
TRES=cpu=96,node=3,billing=96
Socks/Node=* NtasksPerN:B:S:C=32:0:*:* CoreSpec=* 
MinCPUsNode=32 MinMemoryNode=0 MinTmpDiskNode=0
Features=haswell DelayBoot=2-00:00:00
OverSubscribe=NO Contiguous=0 Licenses=ccscratch1:1 Network=(null)
Command=/global/cscratch1/sd/fbench/nersc-python-bench/scripts/mpi4py-import-cori-haswell-scratch-003.sh
WorkDir=/global/cscratch1/sd/fbench/nersc-python-bench/scripts
StdErr=/global/cscratch1/sd/fbench/nersc-python-bench/scripts/logs/mpi4py-import-cori-haswell-scratch-003-31610730.out
StdOut=/global/cscratch1/sd/fbench/nersc-python-bench/scripts/logs/mpi4py-import-cori-haswell-scratch-003-31610730.out
```
### sacct: Query Completed and Pending Jobs

<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>31413414</td>
<td>2020-06-09T02:20:35</td>
<td>2020-06-09T02:24:41</td>
<td>00:04:06</td>
<td>00:30:00</td>
<td>150</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>31438497</td>
<td>Unknown</td>
<td>Unknown</td>
<td>00:00:00</td>
<td>00:30:00</td>
<td>150</td>
<td>0:0</td>
<td>PENDING</td>
</tr>
<tr>
<td>fbench</td>
<td>31438498</td>
<td>Unknown</td>
<td>Unknown</td>
<td>00:00:00</td>
<td>00:30:00</td>
<td>150</td>
<td>0:0</td>
<td>PENDING</td>
</tr>
<tr>
<td>fbench</td>
<td>31541061</td>
<td>2020-06-09T01:51:34</td>
<td>2020-06-09T02:06:46</td>
<td>00:15:12</td>
<td>00:45:00</td>
<td>769</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>31541062</td>
<td>2020-06-09T02:41:30</td>
<td>2020-06-09T03:38:08</td>
<td>00:56:38</td>
<td>02:00:00</td>
<td>150</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>31541063</td>
<td>2020-06-09T03:14:48</td>
<td>2020-06-09T03:20:51</td>
<td>00:06:03</td>
<td>00:30:00</td>
<td>768</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>31541064</td>
<td>2020-06-09T00:15:04</td>
<td>2020-06-09T00:45:28</td>
<td>00:30:24</td>
<td>01:00:00</td>
<td>47</td>
<td>1:0</td>
<td>FAILED</td>
</tr>
<tr>
<td>fbench</td>
<td>31541065</td>
<td>2020-06-09T03:29:53</td>
<td>2020-06-09T03:36:10</td>
<td>00:06:17</td>
<td>00:15:00</td>
<td>768</td>
<td>0:0</td>
<td>COMPLETED</td>
</tr>
<tr>
<td>fbench</td>
<td>31541066</td>
<td>2020-06-09T03:40:06</td>
<td>2020-06-09T03:41:10</td>
<td>00:01:04</td>
<td>00:10:00</td>
<td>768</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 “sacct --help” or “man sacct” for more details.
Running Jobs Best Practices
Where 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 (much shorter on KNL than on Haswell), throughput, charging, code readiness on KNL
  - Cori Haswell (also known as the "Cori Data Partition") system is designed to accelerate data-intensive applications; 2388 total compute nodes
  - Cori KNL: Large capability and performance; 9688 total compute nodes
  - Smaller KNL charging factor (80) than Haswell (140)
  - Jobs use 1024+ nodes on Cori KNL get 20% charging discount
  - "flex" and "low" qos with discounts only available on Cori KNL
  - "shared" and "realtime" available on Cori Haswell only
  - "interactive" nodes available on Cori Haswell and KNL
  - "bigmem" and "xfer" available on Cori (run on external login nodes)
## Cori Haswell Queue Policy (as of June 2020)

<table>
<thead>
<tr>
<th>QOS</th>
<th>Max nodes</th>
<th>Max time (hrs)</th>
<th>Submit limit</th>
<th>Run limit</th>
<th>Priority</th>
<th>QOS Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>1932</td>
<td>48</td>
<td>5000</td>
<td>-</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>shared(^1)</td>
<td>0.5</td>
<td>48</td>
<td>10000</td>
<td>-</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>interactive(^4)</td>
<td>64</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>debug</td>
<td>64</td>
<td>0.5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>premium</td>
<td>1772</td>
<td>48</td>
<td>5</td>
<td>-</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>overrun(^2)</td>
<td>1772</td>
<td>48</td>
<td>5000</td>
<td>-</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>xfer</td>
<td>1 (login)</td>
<td>48</td>
<td>100</td>
<td>15</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bigmem</td>
<td>1 (login)</td>
<td>72</td>
<td>100</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>realtime(^3)</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>1</td>
<td>custom</td>
</tr>
<tr>
<td>special(^5)</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>-</td>
<td>custom</td>
</tr>
</tbody>
</table>
# Cori KNL Queue Policy (as of June 2020)

<table>
<thead>
<tr>
<th>QOS</th>
<th>Max nodes</th>
<th>Max time (hrs)</th>
<th>Submit limit</th>
<th>Run limit</th>
<th>Priority</th>
<th>QOS Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>9489</td>
<td>48</td>
<td>5000</td>
<td>-</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>64</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>debug</td>
<td>512</td>
<td>0.5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>premium</td>
<td>9489</td>
<td>48</td>
<td>5</td>
<td>-</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>low</td>
<td>9489</td>
<td>48</td>
<td>5000</td>
<td>-</td>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td>flex</td>
<td>256</td>
<td>48</td>
<td>5000</td>
<td>-</td>
<td>6</td>
<td>0.25</td>
</tr>
<tr>
<td>overrun</td>
<td>9489</td>
<td>48</td>
<td>5000</td>
<td>-</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>special</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>-</td>
<td>custom</td>
</tr>
</tbody>
</table>
Charging (1)

- **Unit:** NERSC Hours
- **Each architecture has a base charge per node hour used:**
  - Cori Haswell: 140
  - Cori KNL: 80
- **Modification to base charge by QOS used:**
  - premium: 2.0
  - regular: 1.0 (default)
  - low: 0.5
  - flex: 0.25
  - overrun: 0
  - shared: fraction of the node used
- **On Cori KNL**
  - Jobs requesting 1024 or more nodes get a 20% discount
Charging (2)

- Your project is charged for each node your job was allocated for the entire duration, (i.e. used time, not wall request time), 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 * 140 * 2 = 1120
  - “shared” jobs are charged with # of physical cores used instead of the entire node.

- If you have access to multiple projects, pick which one to charge in your batch script
  
  `#SBATCH -A project_name`
Jobs Scheduling

- Each job has its priority value, composed of qos, job age, and a small value of fairshare.
- There are two Slurm schedulers: main and backfill.
- Every few minutes, the main scheduler schedules jobs in the order of priority list for a few days into the future.
  - Jobs are only eligible to be scheduled if they've reached a priority threshold.
  - Currently only 2 jobs per qos per user are considered for scheduling.
- The backfill scheduler then schedules small and short jobs to run if they will not affect the start time of those jobs that are already scheduled by the main scheduler.
Tips for Getting Better Throughput

● Line jumping is allowed, but it may cost more (with “premium” QOS)

● Submit shorter jobs, they are easier to schedule
  ○ Checkpoint to break up long jobs, use variable time
  ○ Short jobs can take advantage of “backfill” opportunities
  ○ Run short jobs just before maintenance
  ○ Run variable-time jobs; use “flex” QOS

● Make sure the wall clock time you request is accurate
  ○ Larger shorter jobs are easier to schedule than long smaller jobs
  ○ Many users unnecessarily request the largest wall clock time possible as default

● Check queue backlogs and queue wait times
  ○ [https://my.nersc.gov/backlog.php](https://my.nersc.gov/backlog.php)
  ○ [https://my.nersc.gov/queuewaittimes.php](https://my.nersc.gov/queuewaittimes.php)
Large Jobs Considerations

- sbcast your executable to compute nodes before srun:
  
  ```sh
sbcast --compress=lz4 /path/to/exe /tmp/exe
  srun /tmp/exe
  ```

  [https://docs.nersc.gov/jobs/best-practices/#large-jobs](https://docs.nersc.gov/jobs/best-practices/#large-jobs)

- Consider to build statically to run large jobs.
  - There may be considerable startup delays for running large jobs of dynamic executables.

- Consider to use shifter for large jobs using shared libraries.

- Consider to use burst buffer for jobs doing large IO.
Other Running Jobs Considerations

- Remember to compile separately for each type of compute nodes
- **Running jobs from global homes is strongly discouraged**
  - IO is not optimized
  - The global homes file system access on compute nodes is much slower than from $SCRATCH
  - It may also cause negative impact for other users interactive response on the system
- Consider to put your project’s shared software in /global/common/software/<project>
  - It is mounted read-only on compute nodes, so has less impact than other GPFS file systems (global homes or community file system)
- Consider to adopt workflow tools for better managing your jobs
More Information

● NERSC Web pages:
  ○ https://docs.nersc.gov/jobs/
  ○ https://docs-dev.nersc.gov
    ■ Using Cori GPU nodes, not covered in this talk

● Contact NERSC Consulting:
  ○ File a service ticket via Help Portal
    https://help.nersc.gov
Thank You and Welcome to NERSC!