Outline

● Modules
● Programming Environments
● Example Code Compilation
● Job Scripts & Affinity
Modules: Loading Preinstalled Software
module avail spider
## Modules Loaded at Login

### Modules Loaded by Default:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>craype-x86-milan</td>
<td>7)</td>
</tr>
<tr>
<td>2)</td>
<td>libfabric/1.15.0.0</td>
<td>8)</td>
</tr>
<tr>
<td>3)</td>
<td>craype-network-ofi</td>
<td>9)</td>
</tr>
<tr>
<td>4)</td>
<td>perftools-base/22.06.0</td>
<td>10)</td>
</tr>
<tr>
<td>5)</td>
<td>xpmem/2.4.4-2.3_12.2__gff0e1d9.shasta</td>
<td>11)</td>
</tr>
<tr>
<td>6)</td>
<td>gcc/11.2.0</td>
<td>12)</td>
</tr>
</tbody>
</table>

### Additional Modules:

- 13) darshan/3.4.0
- 14) Nsight-Compute/2022.1.1
- 15) Nsight-Systems/2022.2.1
- 16) cudatoolkit/11.7
- 17) craype-accel-nvidia80
- 18) gpu/1.0

- **CPU Architecture**
- **Default Programming Environment and Compiler**
- **GPU Architecture, CUDA-Aware MPI, GPU Profilers**

CUDA-Aware MPI by Default!
For CPU-only code we recommend:

```
module load cpu
```
Modules with Lmod

Most Common

• module list
• module load/unload
• module swap
• module show
• module spider

Cool Tricks

• module --redirect -r spider . | grep <string>
• ml -t
• module reset

More information: man module or https://docs.nersc.gov/environment/lmod/
New Hierarchical Structure of Modules on Perlmutter

- Searches all modules without regard for hierarchy
- Displays only modules that can currently be loaded
Modules with Lmod

No Longer Recommended

- module avail

*Only shows packages that can be loaded into the current module environment (due to hierarchy) – use module spider instead

More information: man module or https://docs.nersc.gov/environment/lmod/
Modules with Lmod

No Longer Recommended

• module avail
  *Only shows packages that can be loaded into the current module environment (due to hierarchy) – use module spider instead

More information: man module or https://docs.nersc.gov/environment/lmod/
New Hierarchical Structure of Modules on Perlmutter

- Searches for modules without regard for hierarchy
- Displays only modules that can currently be loaded
Loading Modules Modifies Your Environment

epalmer@perlmutter:login25:~/Training> $ module show cray-hdf5

/path/to/modulefile

family("hdf5")
conflict("PrgEnv-pathscale")
help({"Release info: /path/to/Release_info")
whatis("The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format.")
prepend_path("PATH", "/path/to/bin")
prepend_path("PKG_CONFIG_PATH", "/path/to/lib/pkgconfig")
prepend_path("PE_PKGCONFIG_LIBS", "hdf5_hi:hfdf5")
setenv("PE_HDF5_PKGCONFIG_LIBS", "hdf5_hi:hfdf5")
prepend_path("PE_FORTRAN_PKGCONFIG_LIBS", "hdf5hi_fortran:hfdf5_fortran")
setenv("PE_HDF5_FORTRAN_PKGCONFIG_LIBS", "hdf5hi_fortran:hfdf5_fortran")
prepend_path("PE_CXX_PKGCONFIG_LIBS", "hdf5_hi_cpp:hfdf5_cpp")
setenv("PE_HDF5_CXX_PKGCONFIG_LIBS", "hdf5_hi_cpp:hfdf5_cpp")
setenv("CRAY_HDF5_DIR", "/path/to/pe/hdf5/1.12.1.5")
setenv("PE_HDF5_DIR", "/path/to/pe/hdf5/1.12.1.5")
setenv("CRAY_HDF5_VERSION", "1.12.1.5")
setenv("CRAY_HDF5_PREFIX", "/path/to/pe/hdf5/1.12.1.5/gnu/9.1")
setenv("HDF5_DIR", "/path/to/pe/hdf5/1.12.1.5/gnu/9.1")
setenv("HDF5_ROOT", "/path/to/pe/hdf5/1.12.1.5/gnu/9.1")
prepend_path("CRAY_LD_LIBRARY_PATH", "/path/to/pe/hdf5/1.12.1.5/gnu/9.1/lib")
prepend_path("MODULEPATH", "/path/to/pe/lmod/modulefiles/hdf5/gnu/8.0/cray-hdf5/1.12.1")

Path Changes
Environment Variables
Other Info
Programming Environments: Configuring Compilers and Libraries
# Programming Environments

## Control Compilers and Libraries on Perlmutter

<table>
<thead>
<tr>
<th>Language</th>
<th>Wrapper</th>
<th>PrgEnv-gnu (default)</th>
<th>PrgEnv-nvidia</th>
<th>PrgEnv-cray</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>CC</td>
<td>g++</td>
<td>nvc++</td>
<td>crayCC (Clang)</td>
</tr>
<tr>
<td>C</td>
<td>cc</td>
<td>gcc</td>
<td>nvc</td>
<td>craycc (Clang)</td>
</tr>
<tr>
<td>Fortran</td>
<td>ftn</td>
<td>gfortran</td>
<td>nvfortran</td>
<td>crayftn</td>
</tr>
<tr>
<td>MPI</td>
<td>-</td>
<td>cray-mpich</td>
<td>cray-mpich</td>
<td>cray-mpich</td>
</tr>
</tbody>
</table>

More info:
- module show PrgEnv-gnu
- module show nvidia
- module show cce
Switching Programming Environments

To switch programming environments (PrgEnv) use:

```
module load PrgEnv-`
```

For example,

if I am in PrgEnv-gnu and want to switch to PrgEnv-cray, type:

```
module load PrgEnv-cray
```

Suggested Practice: Compiler Wrappers

Cray provides wrappers (CC, cc, and ftn) to the corresponding compilers for each PrgEnv. These wrappers incorporate many flags and features, including cray-mpich, the recommended MPI.

```bash
epalmer@nid005015:~/Training> gcc helloworld_openmp.c -fopenmp -o hello
epalmer@nid005015:~/Training> cc -craype-verbose helloworld_openmp.c -fopenmp -o hello
```

```bash
gcc -march=znver3 -D__CRAY_X86_MILAN -D__CRAY_NVIDIA80 -D__CRAYXT_COMPUTE_LINUX_TARGET -D__TARGET_LINUX__ helloworld_openmp.c -fopenmp -o hello -Wl,-rpath=/opt/cray/pe/gcc-libs -Wl,-Bdynamic -I/opt/cray/pe/mpich/8.1.17/ofi/gnu/9.1/include -I/opt/cray/pe/libsci/21.08.1.2/GNU/9.1/x86_64/include -I/opt/cray/pe/dsmml/0.2.2/dsmml//include -I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/nvvm/include -I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/extras/CUPTI/include ...(and more)
```
Automatic Links Provided By The Wrappers

• Depending on modules loaded, compiler wrappers link:

  MPI, LAPACK, Blas, ScaLAPACK, and more, automatically.

• Cray modules, such as cray-hdf5, cray-fftw, etc. are also linked automatically by the compiler wrappers when loaded into the user environment.

Note: Several scientific libraries such as, LAPACK, ScaLAPACK, Blas and QDWH, are included in cray-libsci. For more information use: man libsci.
Manually Specify Cray Compile Wrappers

Some build systems such as CMake or Makefiles may be coded to search for CC, CXX and FC environment variables.

In these cases, it is possible to specify the Cray compile wrappers by setting the environment variables in the following way:

```
CC=$(which cc) CXX=$(which CC) FC=$(which ftn)
```

Or at the configure step,

```
./configure CC=cc CXX=CC FC=ftn F77=ftn
```

More info: [https://docs.nersc.gov/development/build-tools/autoconf-make/](https://docs.nersc.gov/development/build-tools/autoconf-make/)
[https://docs.nersc.gov/development/build-tools/cmake/](https://docs.nersc.gov/development/build-tools/cmake/)
Modules Link Dynamically by Default

- Many modules prepend the LD_LIBRARY_PATH or CRAY_LD_LIBRARY_PATH, and have their shared libraries dynamically linked.
  
  e.g.
  ```
  module load gsl
  CC gsl_test.cpp -lgsl -lgslcblas -o gsl_test
  ```

- If you are compiling your own shared libraries, consider using the option, `-Wl,-rpath=<library_path>`. Cray wrappers build dynamically linked executables by default.

- On Perlmutter static compilation with `-static` or `CRAYPE_LINK_TYPE=static` can fail and is not supported.

More info: [https://docs.nersc.gov/development/compilers/wrappers/](https://docs.nersc.gov/development/compilers/wrappers/)
### Other Good-to-Know Compiler Settings for CPU

<table>
<thead>
<tr>
<th>GNU</th>
<th>Cray</th>
<th>Nvidia</th>
<th>Description/ Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O0</td>
<td>-O0</td>
<td>-O1</td>
<td>Default Optimization Level</td>
</tr>
<tr>
<td>-Ofast</td>
<td>-Ofast, -flto</td>
<td>-O4, -fast</td>
<td>Aggressive Optimization (some may cause non-bit-identical output)</td>
</tr>
<tr>
<td>-fopenmp</td>
<td>-fopenmp</td>
<td>-mp</td>
<td>Enable OpenMP (not default)</td>
</tr>
<tr>
<td>-g, -O0</td>
<td>-g, -O0</td>
<td>-g (-O0 by default)</td>
<td>Debug</td>
</tr>
<tr>
<td>-v</td>
<td>-v</td>
<td>-v</td>
<td>Verbose</td>
</tr>
</tbody>
</table>

For more information, when in the corresponding PrgEnv type:

- `man gcc/gfortran`
- `man craycc/crayftn`
- `man nvc/nvfortran`

https://docs.nersc.gov/development/compilers/
Three Quick Tips for Compiling Older Codes

**Fortran:**

1. For older codes, try the `-std=legacy` flag which includes some flags that reduce strictness, such as `-fallow-argument-mismatch`

**C/C++:**

2. Look for flags that reduce strictness, such as `-fpermissive`

3. `-Wpedantic` can warn you about lines that break code standards
# NERSC Programming Environment (Experimental)

<table>
<thead>
<tr>
<th>Language</th>
<th>Suggested Compile Commands</th>
<th>PrgEnv-Ilvm</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>mpic++</td>
<td>clang (Intel)</td>
</tr>
<tr>
<td>C</td>
<td>mpicc</td>
<td>clang (Intel)</td>
</tr>
<tr>
<td>Fortran</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>MPI</td>
<td>-</td>
<td>cray-mpich</td>
</tr>
</tbody>
</table>

**Usage:**

```bash
module use /global/cfs/cdirs/nstaff/cookbg/pe/modulefiles
dmodule load npe
dmodule load PrgEnv-Ilvm
```

**More Info:** [https://docs.nersc.gov/development/compilers/npe/](https://docs.nersc.gov/development/compilers/npe/)
OpenMPI

Usage:

```
module use /global/common/software/m3169/perlmutter/modulefiles
module load openmpi
```

Compile:

```
mpicc -o my_c_exec my_c_prog.c
mpif90 -o my_f90_exec my_f90_prog.f90
```

Run:

```
mpirun -np 64 ./my_c_exec
mpirun -np 64 ./my_f90_exec
```

More Info: [https://docs.nersc.gov/development/programming-models/mpi/openmpi/](https://docs.nersc.gov/development/programming-models/mpi/openmpi/)
Building Applications: Cori codes should be recompiled for Perlmutter
Example CPU Code Compile with MPI and OpenMP

Hellohybrid.c – contains both MPI and OpenMP commands.

```c
int main(int argc, char *argv[]) {
    int numprocs, rank, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int iam = 0, np = 1;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processor_name, &namelen);

    #pragma omp parallel default(shared) private(iam, np)
    {
        np = omp_get_num_threads();
        iam = omp_get_thread_num();
        printf("Hello from thread %d out of %d from process %d out of %d on %s\n",
            iam, np, rank, numprocs, processor_name);
    }
    MPI_Finalize();
}
```

Example from:
https://rcc.uchicago.edu/docs/running-jobs/hybrid/index.html
Compile with MPI and OpenMP (CPU only)

Compile line: `cc hellohybrid.c -fopenmp -o hellohybrid`
Compile with MPI and OpenMP (CPU only)

Compile line:

```
cc hellohybrid.c -fopenmp -o hellohybrid
```
Compile with MPI and OpenMP (CPU only)

Compile line: `cc hellohybrid.c -fopenmp -o hellohybrid`
Manually Specify Include, Library Location and Links

```bash
epalmer> # In this example, we will show how to manually include and link libraries during the compile step. The example code I will use, requires
```
Manually Specify Include, Library Location and Links
Understanding Job Parameters
Guiding Example Batch Job Script

```bash
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:
- Node
- MPI Task
- Logical CPU
- Thread
- Physical Core
- Processor

Advanced Terms:
- NUMA Domain
Understanding Job Parameters:
Hardware: Node, Processor, Physical Core, Logical CPU
Perlmutter CPU Node Terms

<table>
<thead>
<tr>
<th>From Perlmutter system architecture</th>
<th>This Talk</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x AMD EPYC 7763 (Milan) CPUs</td>
<td>2x AMD EPYC 7763 (Milan) Processors</td>
</tr>
<tr>
<td>64 Cores per CPU</td>
<td>64 Physical Cores per processor</td>
</tr>
<tr>
<td>2 Hyperthreads per core</td>
<td>2 Logical CPUs per physical core</td>
</tr>
<tr>
<td>4 NUMA domains per socket</td>
<td>4 NUMA domains per processor</td>
</tr>
</tbody>
</table>

Diagram of Perlmutter CPU Node
Perlmutter CPU Compute Node

- Node
- Processor
- Physical Core
- *Logical* CPU
Office Building Analogy for Node Architecture

- **Processor**
- **Compute Node(s)**
- **Physical Cores**
- **Logical CPUs / Hardware Threads**
Sample Batch Job Script

```bash
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:
- Node
- MPI Task
- Logical CPU
- Thread
- Physical Core
- Processor

Advanced Terms:
- NUMA Domain
Understanding Job Parameters:
Splitting Up Work: MPI Tasks, (OpenMP) Threads
Cargo Analogy for MPI Tasks & OMP Threads

Simulation Code

MPI Task

(OpenMP) Thread
Guiding Example Batch Job Script

```bash
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:
- Node
- MPI Task
- Logical CPU
- Thread
- Physical Core
- Processor

Advanced Terms:
- NUMA Domain
Understanding Job Parameters:
Memory Affinity: NUMA Domains
What is a NUMA Domain?

NUMA - Non-uniform Memory Access

Takeaway:
Accessing memory from remote NUMA domains is slower than accessing memory from a local NUMA domain.
NUMA Domains on Perlmutter CPU

- NUMA Domain
Perlmutter CPU Node NUMA Info

**numactl -H:** provides NUMA info of CPUs

<table>
<thead>
<tr>
<th>node</th>
<th>cpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>node 0</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143</td>
</tr>
<tr>
<td>node 1</td>
<td>16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159</td>
</tr>
<tr>
<td>node 2</td>
<td>32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175</td>
</tr>
</tbody>
</table>

... node 7 cpus: 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255

**node distances:**

<table>
<thead>
<tr>
<th>node</th>
<th>0 1 2 3 4 5 6 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>10 12 12 12 32 32 32 32</td>
</tr>
<tr>
<td>1:</td>
<td>12 10 12 12 32 32 32 32</td>
</tr>
<tr>
<td>2:</td>
<td>12 12 10 12 32 32 32 32</td>
</tr>
<tr>
<td>3:</td>
<td>12 12 12 10 32 32 32 32</td>
</tr>
<tr>
<td>4:</td>
<td>32 32 32 32 10 12 12 12</td>
</tr>
<tr>
<td>5:</td>
<td>32 32 32 32 12 10 12 12</td>
</tr>
<tr>
<td>6:</td>
<td>32 32 32 32 12 12 10 12</td>
</tr>
<tr>
<td>7:</td>
<td>32 32 32 32 12 12 12 10</td>
</tr>
</tbody>
</table>

Shows relative cost of memory bandwidth
There is a factor of 3 accessing local vs remote NUMA domains in this example
Affinity Verification Methods

- NERSC has provided pre-built binaries from an HPE code (xthi.c) to display process thread affinity: `check-mpi.gnu.pm`, `check-mpi.nvidia.pm`, `check-hybrid.nvidia.pm`, etc.

  ```bash
  % srun -n 32 -c 8 --cpu-bind=cores check-mpi.gnu.pm | sort -nk 4
  Hello from rank 0, on nid006340. (core affinity = 0-3,128-131)
  Hello from rank 1, on nid006340. (core affinity = 64-67,192-195) ...
  ```

- OpenMP 5.0 introduced `OMP_DISPLAY_AFFINITY` and `OMP_AFFINITY_FORMAT`

  ```bash
  % export OMP_DISPLAY_AFFINITY=true
  % export OMP_AFFINITY_FORMAT="host=%H, pid=%P, thread_num=%n, thread affinity=%A"
  % srun -n 2 -c 128 --cpu-bind=cores check-hybrid.gnu.pm | sort -nk 4
  host=nid006340, pid=127001, thread_num=0, thread affinity=64
  host=nid006340, pid=127001, thread_num=1, thread affinity=192 ...
  ```
Settings to Address NUMA Performance

- Use `--cpu_bind=cores` when the # of MPI tasks ≤ the # of physical cores
- Use `--cpu_bind=threads` when the # of MPI tasks > the # of physical cores
- In hybrid MPI/OpenMP code, use at least 8 MPI tasks to avoid NUMA penalties when using OpenMP threads
- The value of `-c` should be ≥ the value of `OMP_NUM_THREADS`
- For thread affinity set:
  
  `OMP_PROC_BIND=spread`
  `OMP_PLACES=threads`
Sample Batch Job Script

```bash
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:
- Node
- MPI Task
- Logical CPU
- Thread
- Physical Core
- Processor

Advanced Terms:
- NUMA Domain
### Compute Nodes Comparison for CPU Affinity

<table>
<thead>
<tr>
<th></th>
<th>Cori Haswell</th>
<th>Cori KNL</th>
<th>Perlmutter CPU</th>
<th>CPU on Perlmutter GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical cores</td>
<td>32</td>
<td>68</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>Logical CPUs per physical core</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Logical CPUs per node</td>
<td>64</td>
<td>272</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>NUMA domains</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>-c value for srun</td>
<td>floor(64/tpn)</td>
<td>floor(272/tpn), usually do floor(256/tpn)</td>
<td>floor(256/tpn)</td>
<td>floor(128/tpn)</td>
</tr>
</tbody>
</table>

*tpn = Number of MPI tasks per node*
Job Scripts
Sample Batch Job Script – MPI Only

**Cori Haswell**

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \ 
--cpu_bind=cores ./demo_code
```

**Perlmutter CPU**

```bash
#!/bin/bash

#SBATCH -N 10
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \ 
--cpu_bind=cores ./demo_code
```

**Tip**: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
Sample Batch Job Script – MPI Only

**Cori Haswell**

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \--cpu_bind=cores ./demo_code
```

**Perlmutter CPU**

```bash
#!/bin/bash

#SBATCH -N 10
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \--cpu_bind=cores ./demo_code
```

**Tip:** As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
Sample Batch Job Script – MPI Only

Cori Haswell

```bash
#!/bin/bash
#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00
export OMP_NUM_THREADS=1
srun -n 1280 -c 2 \--cpu_bind=cores ./demo_code
```

Perlmutter CPU

```bash
#!/bin/bash
#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00
export OMP_NUM_THREADS=1
srun -n 1280 -c 8 \--cpu_bind=cores ./demo_code
```

**Tip**: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
### Sample Batch Job Script – MPI Only

<table>
<thead>
<tr>
<th>Cori Haswell</th>
<th>Perlmutter CPU</th>
</tr>
</thead>
</table>
| ```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \ --cpu_bind=cores ./demo_code
``` | ```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \ --cpu_bind=cores ./demo_code
``` |

---

**Tip:** As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 320 -c ? --cpu_bind=cores ./demo_code
Sample Batch Job Script – MPI & OpenMP

Question

#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 320 -c ? \
   --cpu_bind=cores ./*demo_code

Hint

$320 / 40 = 8$

$256 / 8 = 32$

$32 \geq 8$

Considered because using OpenMP
Sample Batch Job Script – MPI & OpenMP

Question

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 320 -c ? --cpu_bind=cores ./demo_code
```

Process

1. MPI tasks / Nodes
   \[320 \div 40 = 8\]

2. Logical CPUs / MPI tasks
   \[256 \div 8 = 32\]

3. Check logical CPUs greater than threads
   \[32 > 8\]

   Considered because using OpenMP
### Question

```bash
#!/bin/bash

#SBATCH -N 40  
#SBATCH -C cpu  
#SBATCH -q regular  
#SBATCH -J job_name  
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 320 -c ? \  
--cpu_bind=cores ./demo_code
```

### Answer

```bash
#!/bin/bash

#SBATCH -N 40  
#SBATCH -C cpu  
#SBATCH -q regular  
#SBATCH -J job_name  
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 320 -c 32 \  
--cpu_bind=cores ./demo_code
```
# Sample Batch Job Script – MPI & OpenMP

## MPI Only

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 320 -c 32 \  
  --cpu_bind=cores ./demo_code
```

## Hybrid MPI & OpenMP

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 320 -c 32 \  
  --cpu_bind=cores ./demo_code
```
NERSC Job Script Generator

https://my.nersc.gov/script_generator.php or https://iris.nersc.gov/jobscript
Job Script Generator

- Job scripts are largely similar to Cori
- Job script generator now available within Iris to help you get correct process affinity, etc.
  - [https://iris.nersc.gov/jobscrip](https://iris.nersc.gov/jobscrip)
Spack: Even More Software!
Spack: A Package Manager for Supercomputers

Spack currently contains 6617 mainline packages

NERSC has two instances of Spack available on Perlmutter:

<table>
<thead>
<tr>
<th>Instance</th>
<th>Version</th>
<th>Command</th>
<th># of Packages</th>
<th>Can I Install More?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spack</td>
<td>0.17.1</td>
<td>module load spack</td>
<td>130</td>
<td>Yes</td>
</tr>
<tr>
<td>Extreme-scale Scientific Software Stack (E4S)</td>
<td>0.18.0.dev0</td>
<td>module load e4s/22.05</td>
<td>319</td>
<td>Yes</td>
</tr>
</tbody>
</table>

More info: Spack – https://docs.nersc.gov/development/build-tools/spack/
E4S – https://docs.nersc.gov/applications/e4s/
Spack Installed Packages

```
jp/2.0
```

E4S

Spack

62
## Install or Load a Spack Package

<table>
<thead>
<tr>
<th>Steps</th>
<th>Already Installed in Spack/E4S</th>
<th>Available via Spack/E4S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>spack find -v</code></td>
<td><code>spack list</code></td>
</tr>
<tr>
<td></td>
<td>Match package config. and compiler</td>
<td>Search list of ~6200 avail. packages</td>
</tr>
<tr>
<td>2</td>
<td><code>spack load &lt;package&gt;</code></td>
<td><code>spack info &lt;package&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Load desired package/package variant into environment</td>
<td>Get information about package options</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td><code>spack install &lt;package&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Installs the package into Spack</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td><code>spack load &lt;package&gt;</code></td>
</tr>
</tbody>
</table>

**Extra Step:** `spack load --sh <package>`

Shows the modifications made to your environment when a package is loaded.
Example: Install a Dependency with Spack

Suppose my example code, `gsl_test.cpp`, requires the GNU Scientific Library version 2.7.1. This example will show how to load this dependency from the E4S software stack using Spack.
Example: Install a Dependency with Spack

Suppose my example code, gsl_test.cpp, requires the GNU Scientific Library version 2.7.1. This example will show how to load this dependency from the E4S software stack using Spack.
Key Suggestions:

- Use `module spider` for comprehensive module search
- Recompile Cori code with `PrgEnv-gnu`, `PrgEnv-cray`, or another `PrgEnv`
- Use the compiler wrappers
- Recalculate job script parameters for optimal performance
CPU Hands-on Exercises

- Feel free to use some NERSC prepared CPU examples at
  - [https://github.com/NERSC/Migrate-to-Perlmutter/tree/main/CPU](https://github.com/NERSC/Migrate-to-Perlmutter/tree/main/CPU)
  - or bring your own applications codes today.

- Follow README.first and README for each example
  - hello-example: serial and MPI
  - matrix-example (C) or jacobi-example (Fortran): hybrid MPI/OpenMP
  - xthi-example: affinity
  - gsl_test: using package available from E4S stack

- Perlmutter Compute node reservations, 10:00 - 15:00:
  - CPU: `#SBATCH --reservation=pm_cpu_dec1 -A ntrain2 -C cpu`
  - Existing NERSC users are added to the ntrain2 project to access node reservations
Thanks for your attention!

More questions? Need help? ...
http://help.nersc.gov/