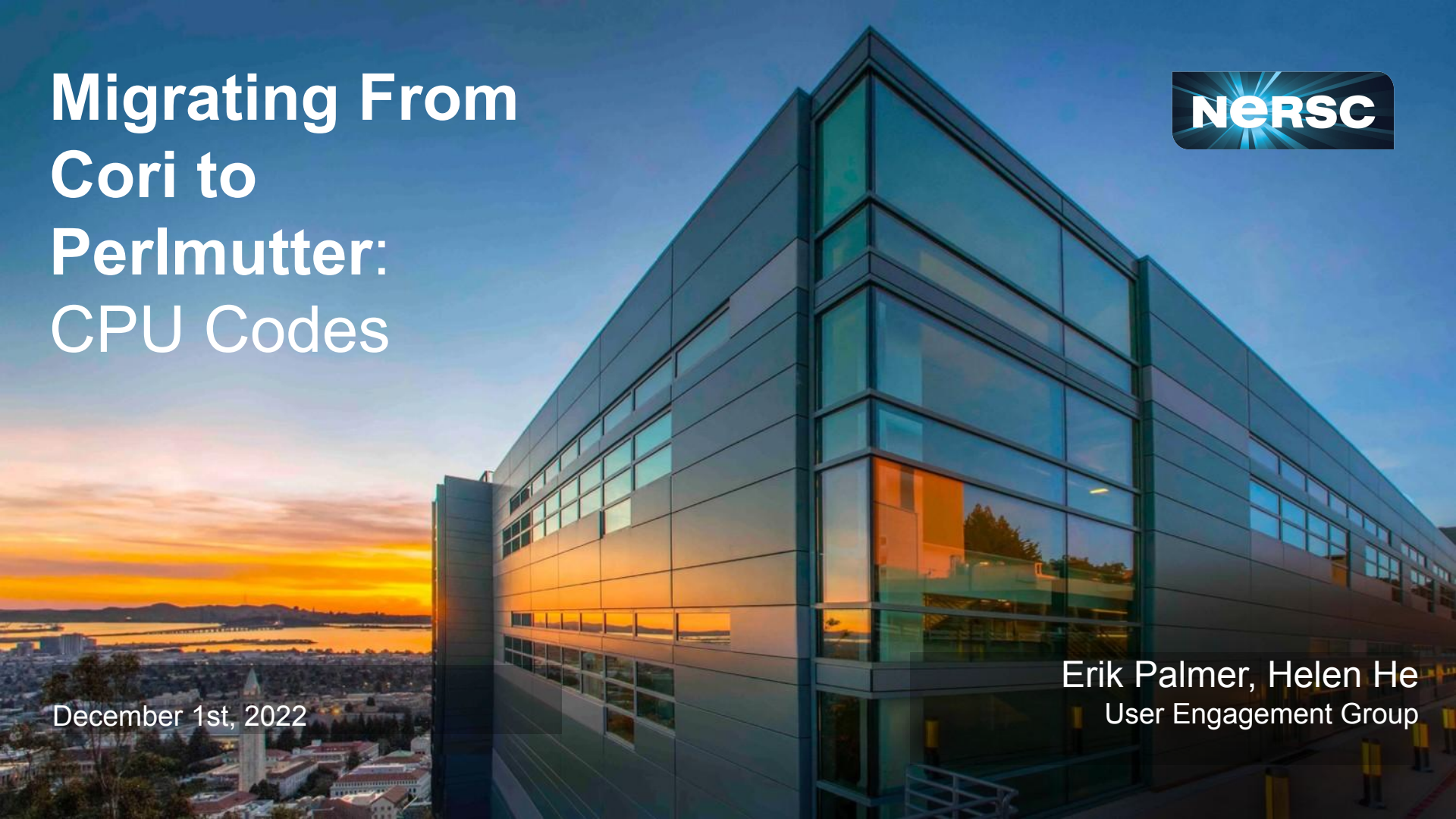
Migrating From Cori to Perlmutter: CPU Codes

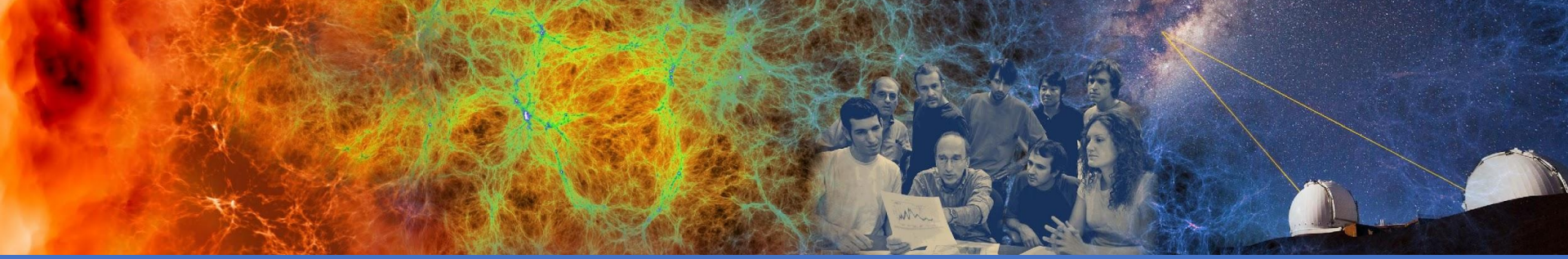


December 1st, 2022

Erik Palmer, Helen He
User Engagement Group

Outline

- Modules
- Programing Environments
- Example Code Compilation
- Job Scripts & Affinity



Modules: Loading Preinstalled Software

`module ~~avail~~ spider`

Modules Loaded at Login

Modules Loaded by Default:

1) craype-x86-milan
2) libfabric/1.15.0.0
3) craype-network-ofi
4) perftools-base/22.06.0
5) xpmem/2.4.4-2.3_12.2__gff0e1d9.shasta
6) gcc/11.2.0
7) craype/2.7.16
8) cray-dsmml/0.2.2
9) cray-mpich/8.1.17
10) cray-libsci/21.08.1.2
11) PrgEnv-gnu/8.3.3
12) xalt/2.10.2
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!

Default Modules for CPU-only Code

For CPU-only code we recommend:
`module load cpu`

1) craype-x86-milan
2) libfabric/1.15.0.0
3) craype-network-ofi
4) perftools-base/22.06.0
5) xpmem/2.4.4-2.3_12.2__gff0e1d9.shasta
6) gcc/11.2.0
7) craype/2.7.16
8) cray-dsmml/0.2.2
9) cray-mpich/8.1.17
10) cray-libsci/21.08.1.2
11) PrgEnv-gnu/8.3.3
12) xalt/2.10.2
13) darshan/3.4.0
14) cpu/1.0

- CPU Architecture
- Default Programming Environment and Compiler
- Configured for CPU-only MPI

CUDA-Aware MPI turned off!

Modules with Lmod

Most Common

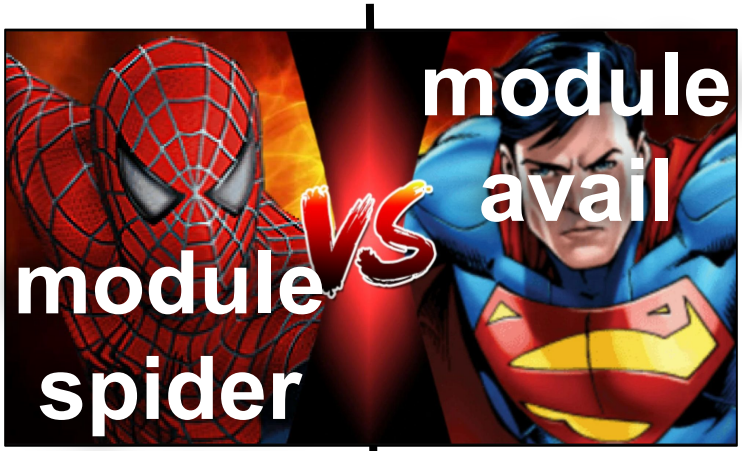
- `module list`
- `module load/unload`
- `module swap`
- `module show`
- `module spider`

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



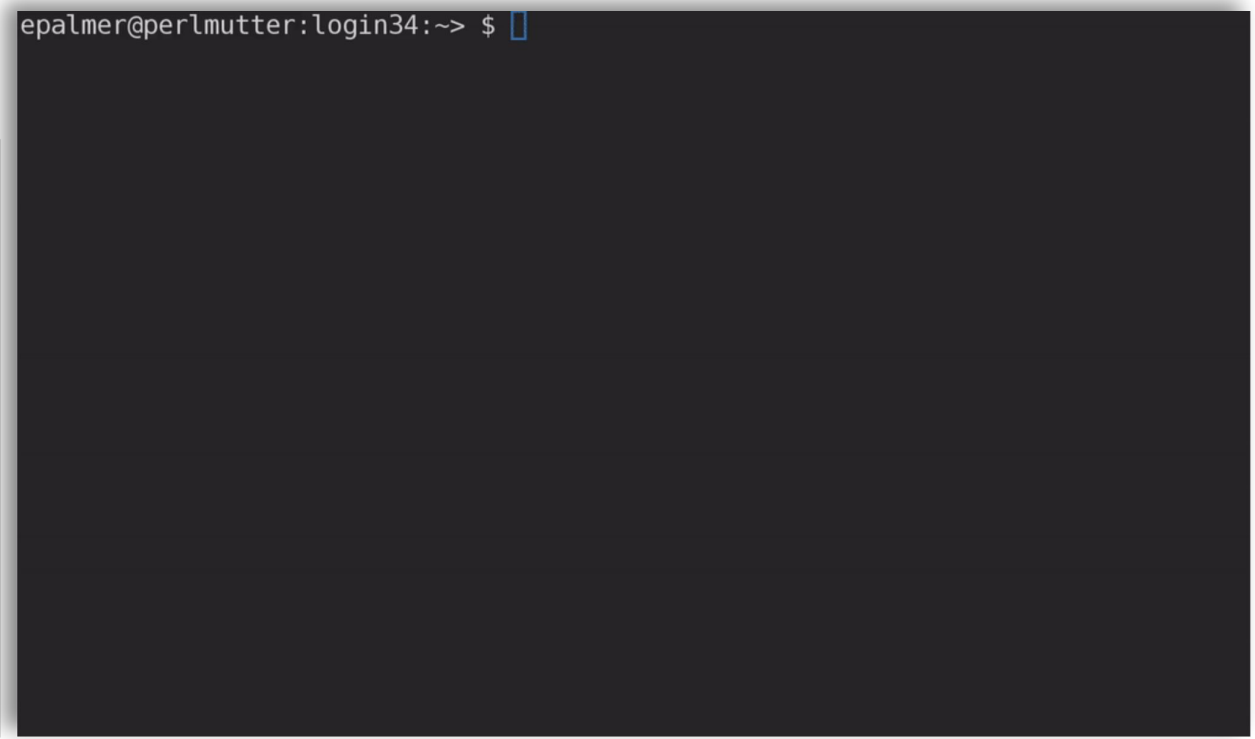
- Searches all modules without regard for hierarchy
- Displays only modules that can currently be loaded

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**



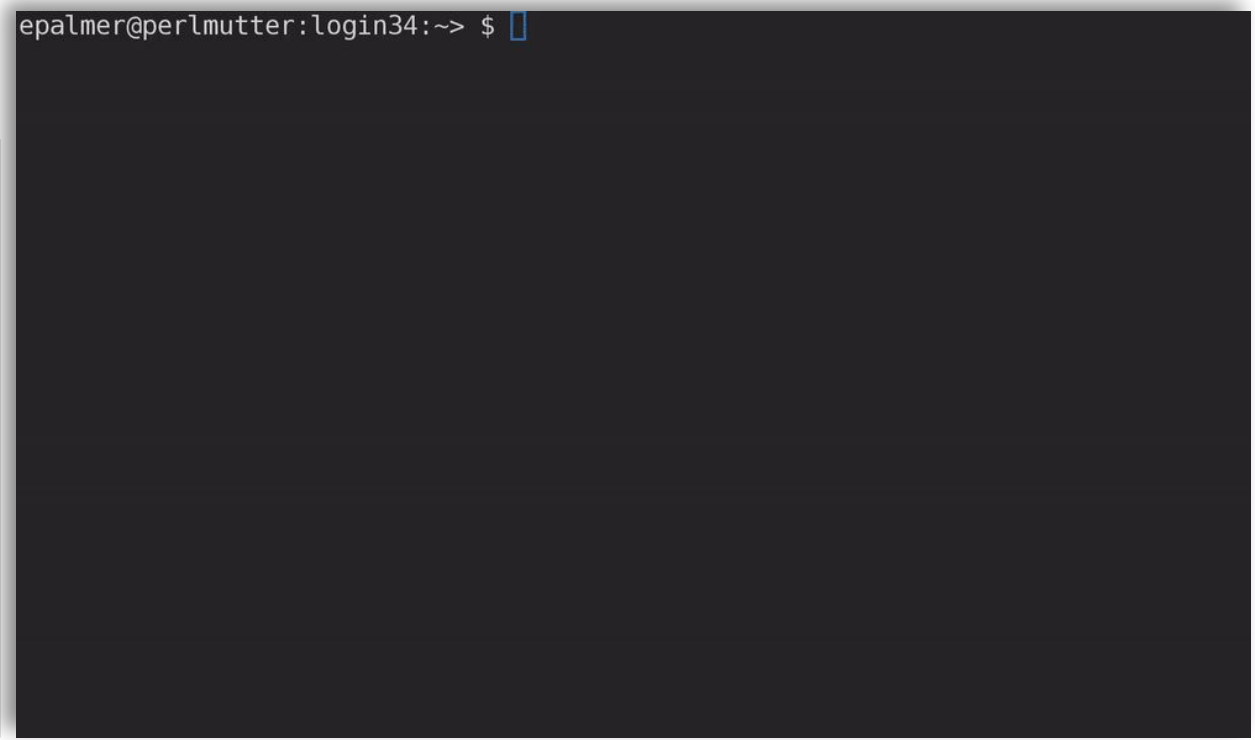
More information: `man module` or https://docs.nersc.gov/environment/lmod/

Modules with Lmod

No Longer Recommended

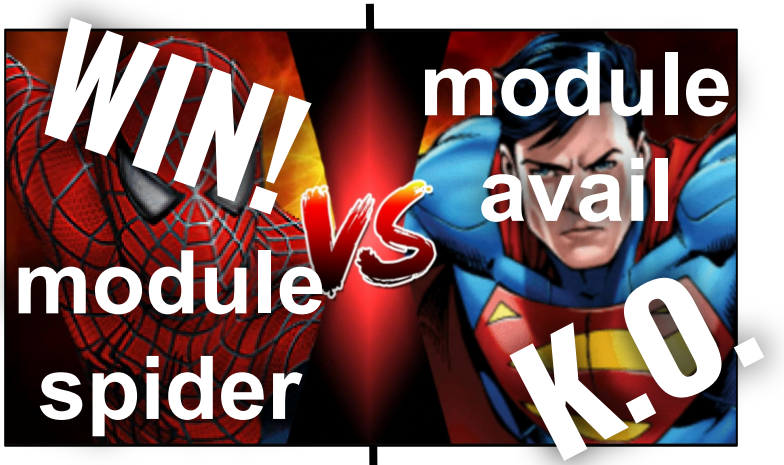
- `module avail`

  *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

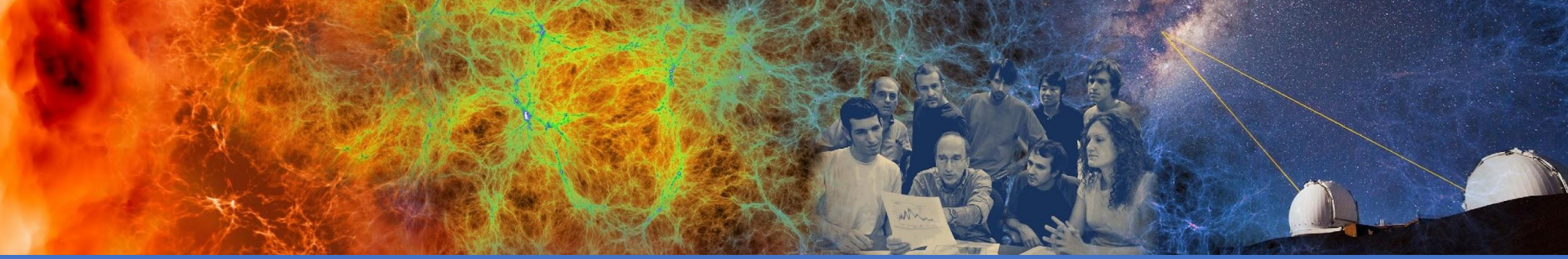


- Searches for modules without regard for hierarchy
- Displays only modules that can currently be loaded

Loading Modules Modifies Your Environment

```
epalmer@perlmutter:login25:~/Training> $ module show cray-hdf5
-------------------------------------------------------------------------------------------------------------------------------
   /opt/cray/pe/lmod/modulefiles/compiler/gnu/8.0/cray-hdf5/1.12.1.5.lua:
-------------------------------------------------------------------------------------------------------------------------------
family("hdf5")
conflict("PrgEnv-pathscale")
help([[Release info:  /opt/cray/pe/hdf5/1.12.1.5/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","/opt/cray/pe/hdf5/1.12.1.5/bin")
prepend_path("PKG_CONFIG_PATH","/opt/cray/pe/hdf5/1.12.1.5/gnu/9.1/lib/pkgconfig")
prepend_path("PE_PKGCONFIG_LIBS","hdf5_hl:hdf5")
setenv("PE_HDF5_PKGCONFIG_LIBS","hdf5_hl:hdf5")
prepend_path("PE_FORTRAN_PKGCONFIG_LIBS","hdf5hl_fortran:hdf5_fortran")
setenv("PE_HDF5_FORTRAN_PKGCONFIG_LIBS","hdf5hl_fortran:hdf5_fortran")
prepend_path("PE_CXX_PKGCONFIG_LIBS","hdf5_hl_cpp:hdf5_cpp")
setenv("PE_HDF5_CXX_PKGCONFIG_LIBS","hdf5_hl_cpp:hdf5_cpp")
setenv("CRAY_HDF5_DIR","/opt/cray/pe/hdf5/1.12.1.5")
setenv("PE_HDF5_DIR","/opt/cray/pe/hdf5/1.12.1.5")
setenv("CRAY_HDF5_VERSION","1.12.1.5")
setenv("CRAY_HDF5_PREFIX","/opt/cray/pe/hdf5/1.12.1.5/gnu/9.1")
setenv("HDF5_DIR","/opt/cray/pe/hdf5/1.12.1.5/gnu/9.1")
setenv("HDF5_ROOT","/opt/cray/pe/hdf5/1.12.1.5/gnu/9.1")
prepend_path("CRAY_LD_LIBRARY_PATH","/opt/cray/pe/hdf5/1.12.1.5/gnu/9.1/lib")
prepend_path("MODULEPATH","/opt/cray/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

Language	Wrapper	PrgEnv-gnu (default)	PrgEnv-nvidia	PrgEnv-cray
C++	CC	g++	nvc++	crayCC (Clang)
C	cc	gcc	nvc	craycc (Clang)
Fortran	ftn	gfortran	nvfortran	crayftn
MPI	-	cray-mpich	cray-mpich	cray-mpich
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.

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

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

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/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/

Other Good-to-Know Compiler Settings for CPU

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

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)

Language	Suggested Compile Commands	PrgEnv-llvm
C++	mpic++	clang (Intel)
C	mpicc	clang (Intel)
Fortran	N/A	N/A
MPI	-	cray-mpich

Usage:

```
module use /global/cfs/cdirs/nstaff/cookbg/pe/modulefiles
module load npe
module load PrgEnv-llvm
```

More Info: https://docs.nersc.gov/development/compilers/npe/

OpenMPI

Usage:

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

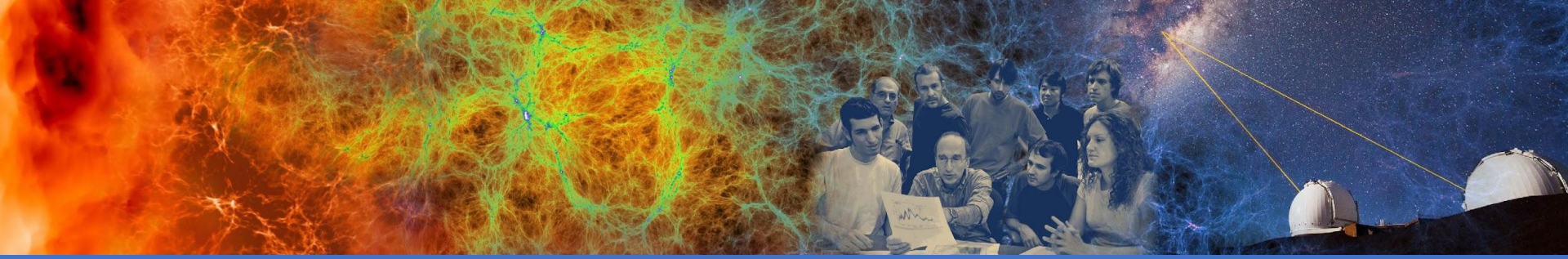
Compile:

```
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/



Building Applications: Cori codes should be recompiled for Perlmutter

Example CPU Code Compile with MPI and OpenMP

```
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();
 }
```

Hellohybrid.c – contains both MPI and OpenMP commands.

Example from: https://rcc.uchicago.edu/docs/running-jobs/hybrid/index.html

Compile with MPI and OpenMP (CPU only)

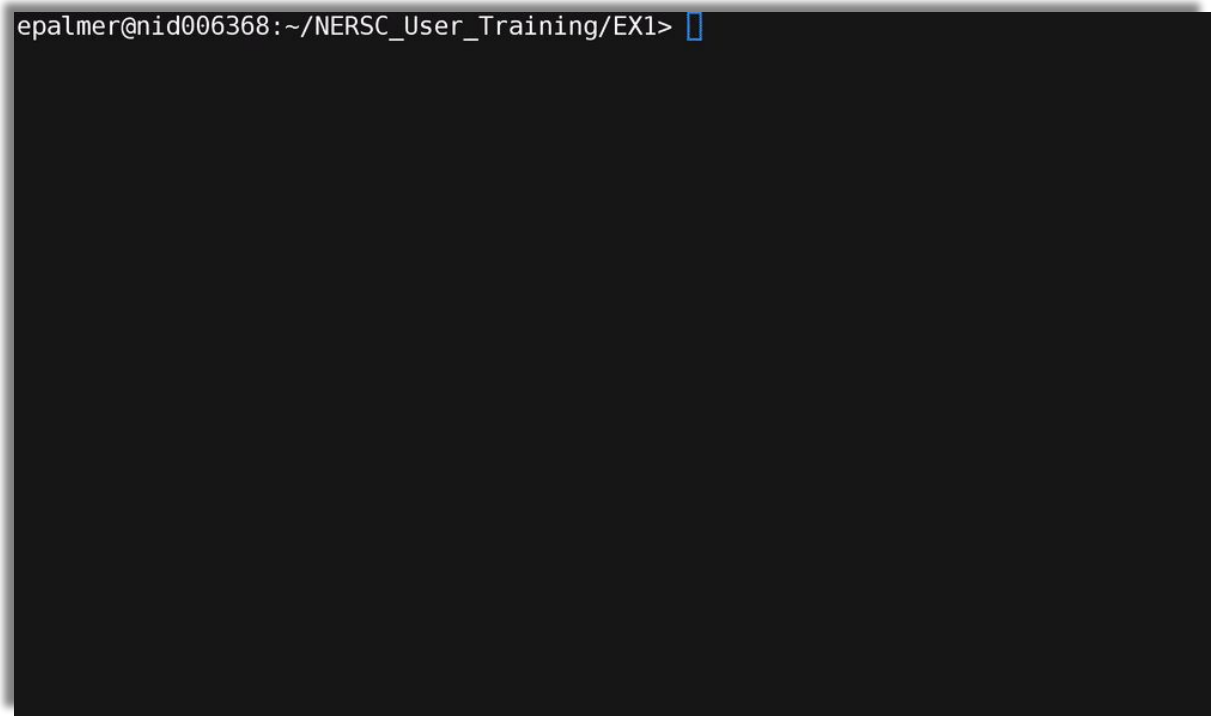
```
epalmer@nid006368:~/NERSC_User_Training/EX1> #In this example, we will compile a
 code with MPI and OpenMP
```

Modules Loaded:
PrgEnv-gnu

Compile line:

```
cc hellohybrid.c -fopenmp -o hellohybrid
```

Compile with MPI and OpenMP (CPU only)



Modules Loaded:
PrgEnv-gnu

Compile line:

```
cc hellohybrid.c -fopenmp -o hellohybrid
```

Compile with MPI and OpenMP (CPU only)

```
epalmer@nid006368:~/NERSC_User_Training/EX1>
```

Compilation with wrappers is very similar.* `-fopenmp`

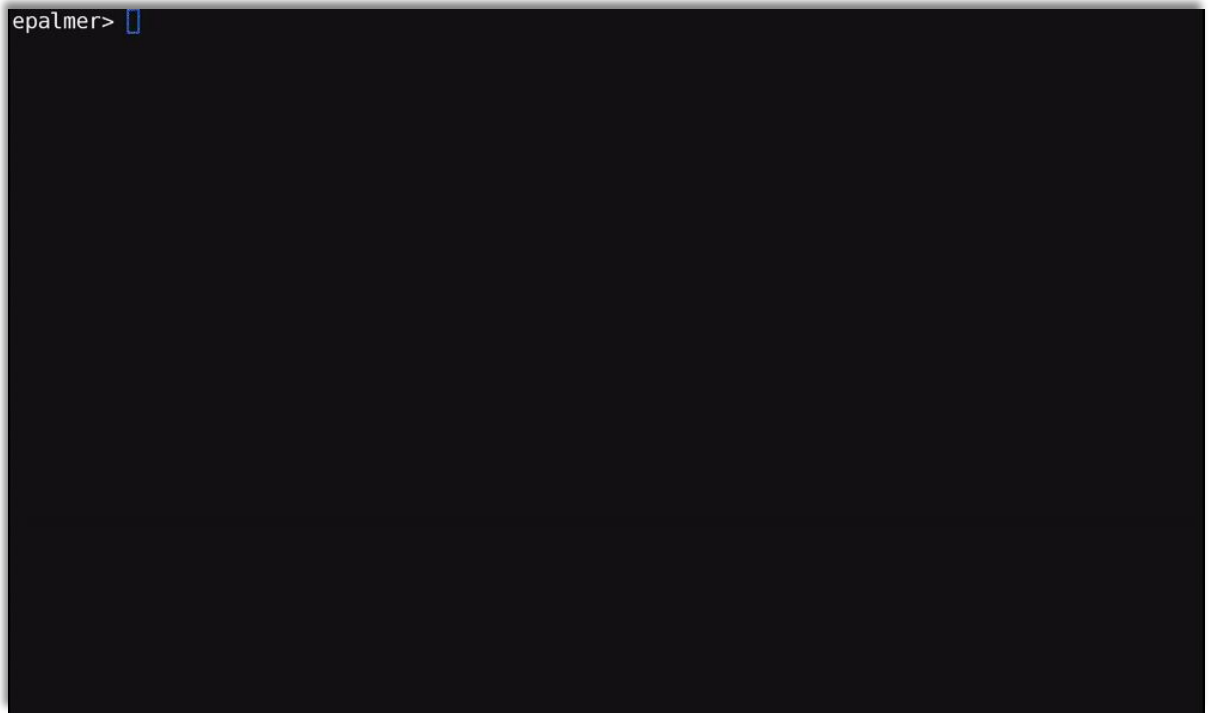
Compile line:

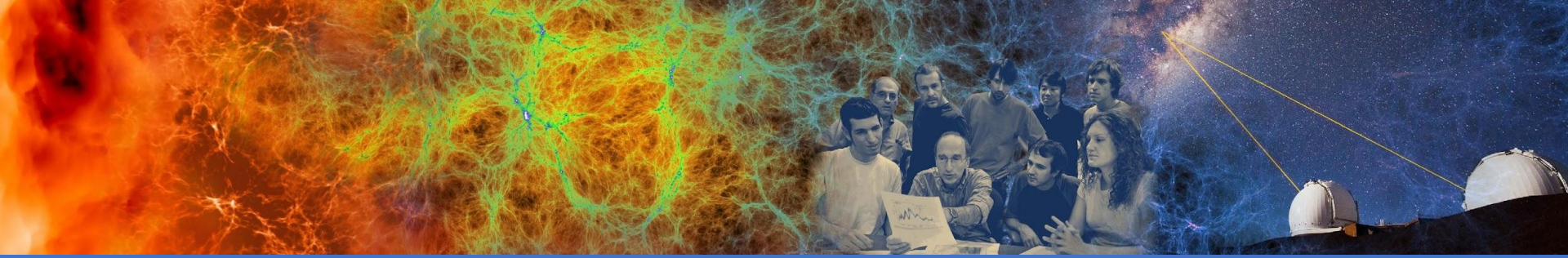
```
cc hellohybrid.c -fopenmp -o hellohybrid
```

Manually Specify Include, Library Location and Links

```
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

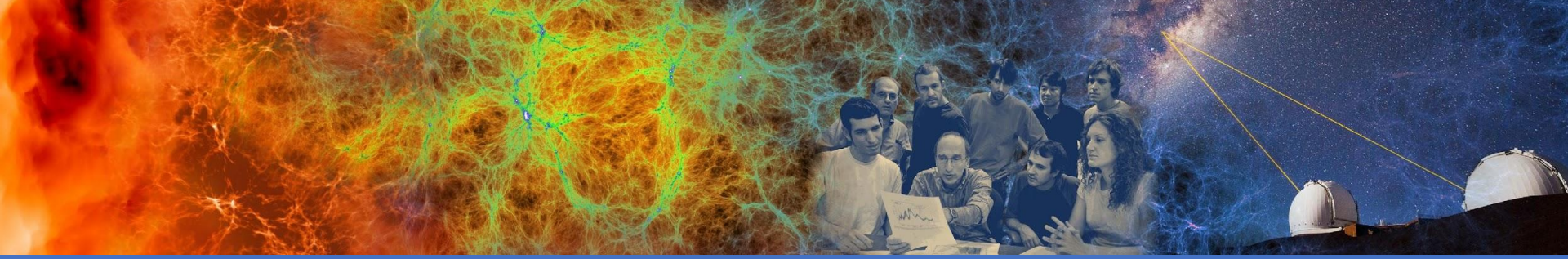
```
#!/bin/bash

#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

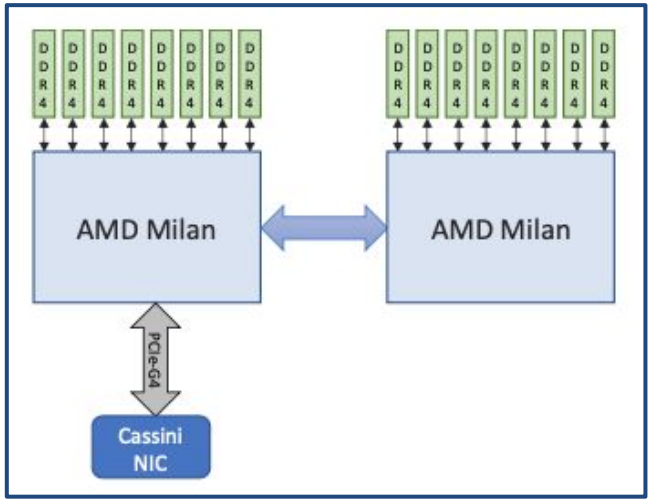


Understanding Job Parameters:
Hardware: Node, Processor, Physical Core, Logical CPU

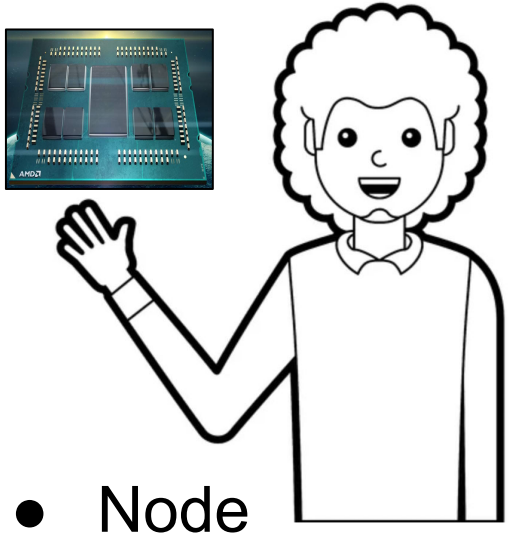
Perlmutter CPU Node Terms

From Perlmutter system architecture	This Talk
2x AMD EPYC 7763 (Milan) CPUs	2x AMD EPYC 7763 (Milan) Processors
64 Cores per CPU	64 Physical Cores per processor
2 Hyperthreads per core	2 Logical CPUs per physical core
4 NUMA domains per socket	4 NUMA domains per processor

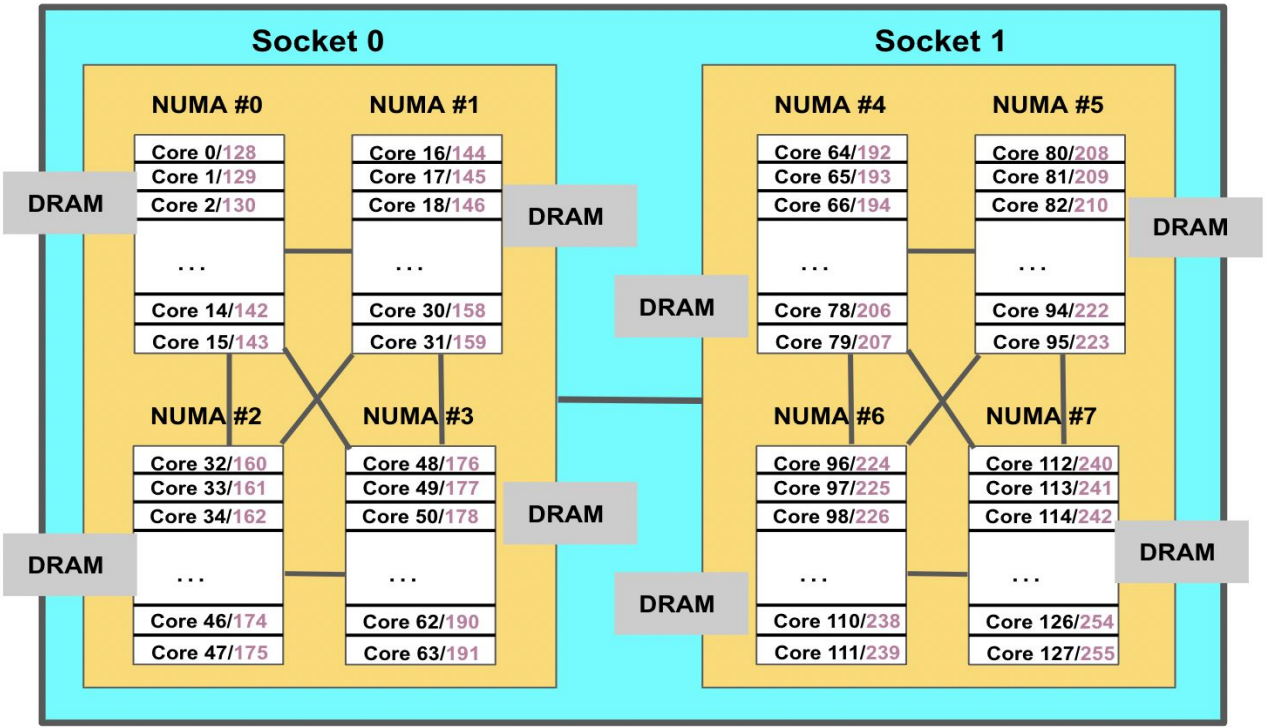
Diagram of Perlmutter CPU Node



Perlmutter CPU Compute Node



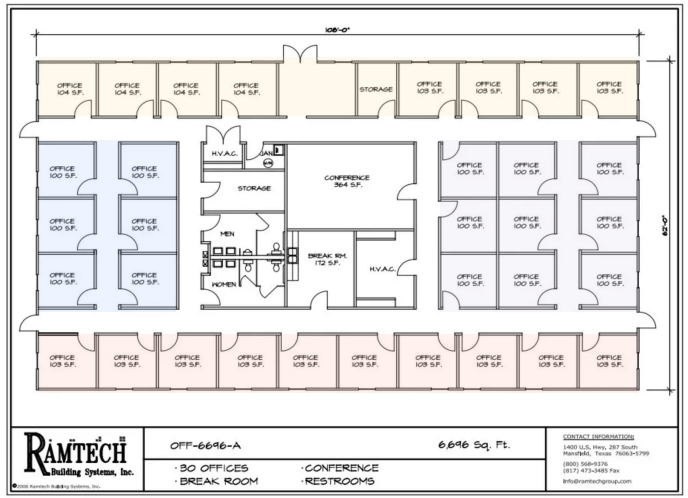
- Node
- Processor
- Physical Core
- *Logical* CPU



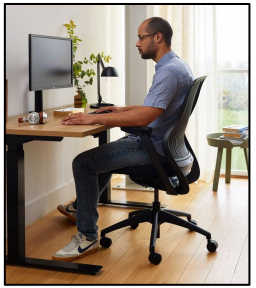
Office Building Analogy for Node Architecture

Compute Node(s)

Physical Cores



Processor



Logical CPUs / Hardware Threads

Sample Batch Job Script

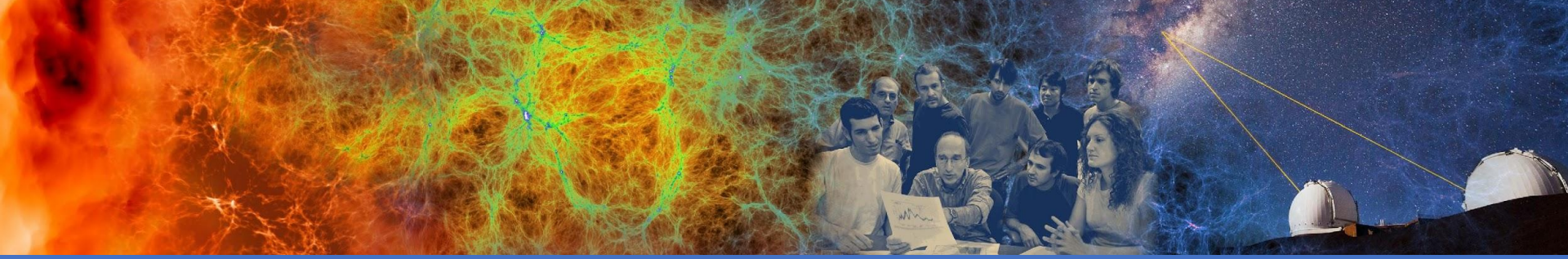
```
#!/bin/bash

#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

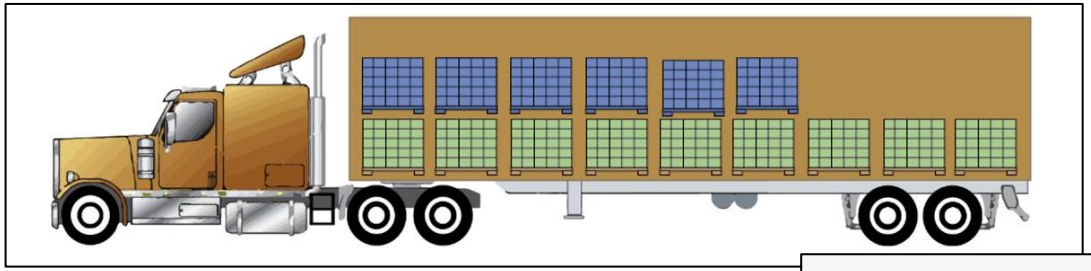
Advanced Terms:

- 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

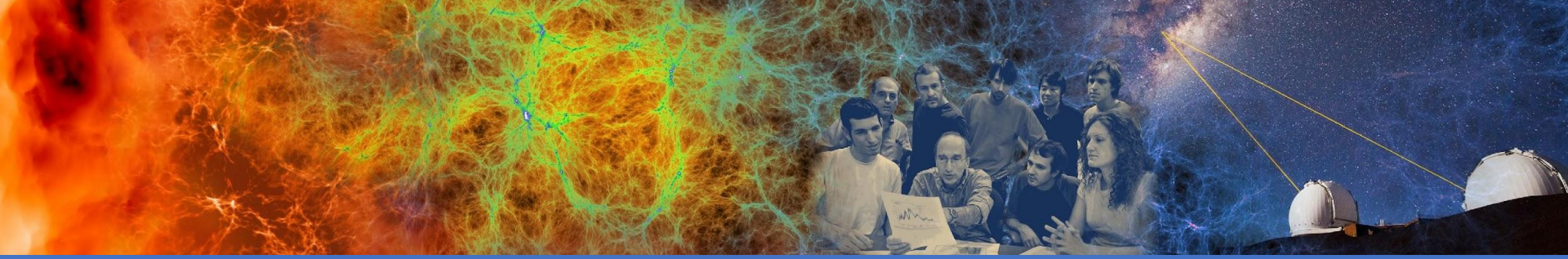
```
#!/bin/bash

#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:

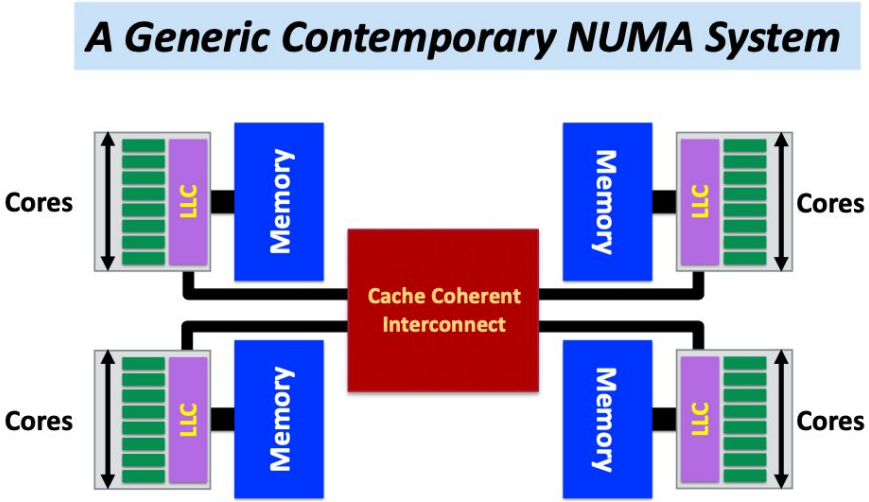
- NUMA Domain



Understanding Job Parameters:
Memory Affinity: NUMA Domains

What is a NUMA Domain?

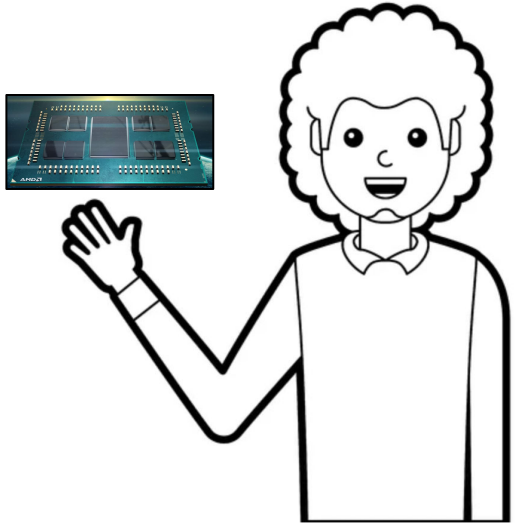
NUMA - Non-uniform Memory Access



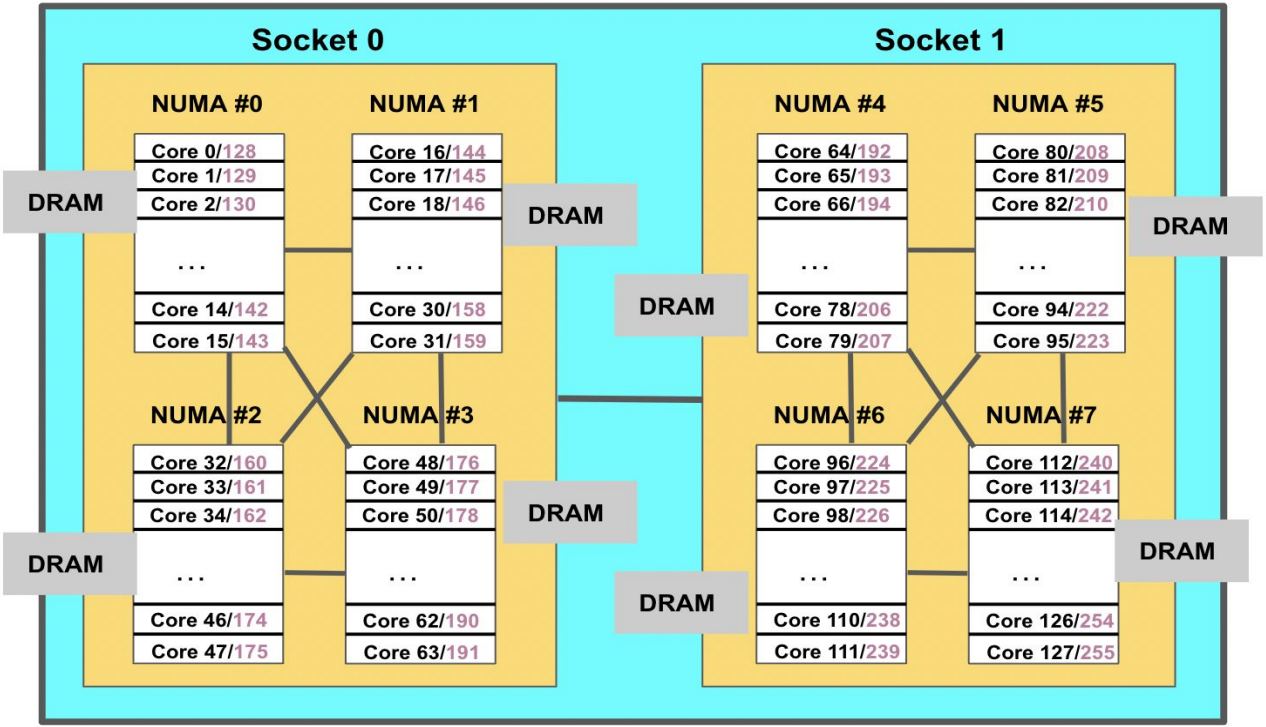
Takeaway:

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

128 cores and 2 processors total
64 cores and 4 NUMA domains per processor

```
yunhe@nid005620:~> numactl -H
available: 8 nodes (0-7)
node 0 cpus: 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
node 1 cpus: 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
node 2 cpus: 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
...
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:
node 0 1 2 3 4 5 6 7
0: 10 12 12 12 32 32 32 32
1: 12 10 12 12 32 32 32 32
2: 12 12 10 12 32 32 32 32
3: 12 12 12 10 32 32 32 32
4: 32 32 32 32 10 12 12 12
5: 32 32 32 32 12 10 12 12
6: 32 32 32 32 12 12 10 12
7: 32 32 32 32 12 12 12 10
```

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.

```
% 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

```
% 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

```
#!/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

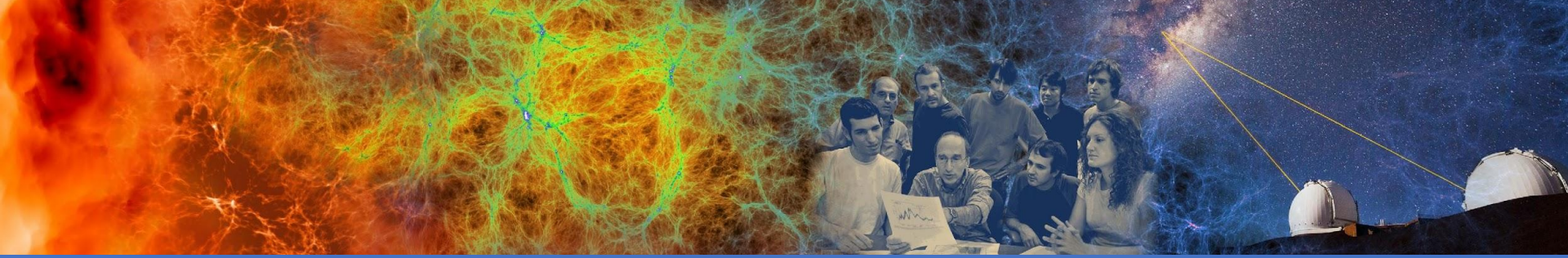
Advanced Terms:

- NUMA Domain

Compute Nodes Comparison for CPU Affinity

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

tpn = Number of MPI tasks per node



Job Scripts

Sample Batch Job Script – MPI Only

Cori Haswell

```
#!/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

```
#!/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

```
#!/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
```

1280/40=32

64/32=2

Perlmutter CPU

```
#!/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
```

1280/10=128

256/128=2

Tip: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`

Sample Batch Job Script – MPI Only

Cori Haswell

```
#!/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

```
#!/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

Cori Haswell

```
#!/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
```

1280/40=32

64/32=2

Perlmutter CPU

```
#!/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
```

1280/40=32

256/32=8

Tip: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`

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
```

What should the value of `-c` be?

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

```
#!/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 / 40 = 8

2. Logical CPUs / MPI tasks

   256 / 8 = 32

3. Check logical CPUs greater than threads

   32 > 8

Considered because using OpenMP

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
```

Answer

```
#!/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

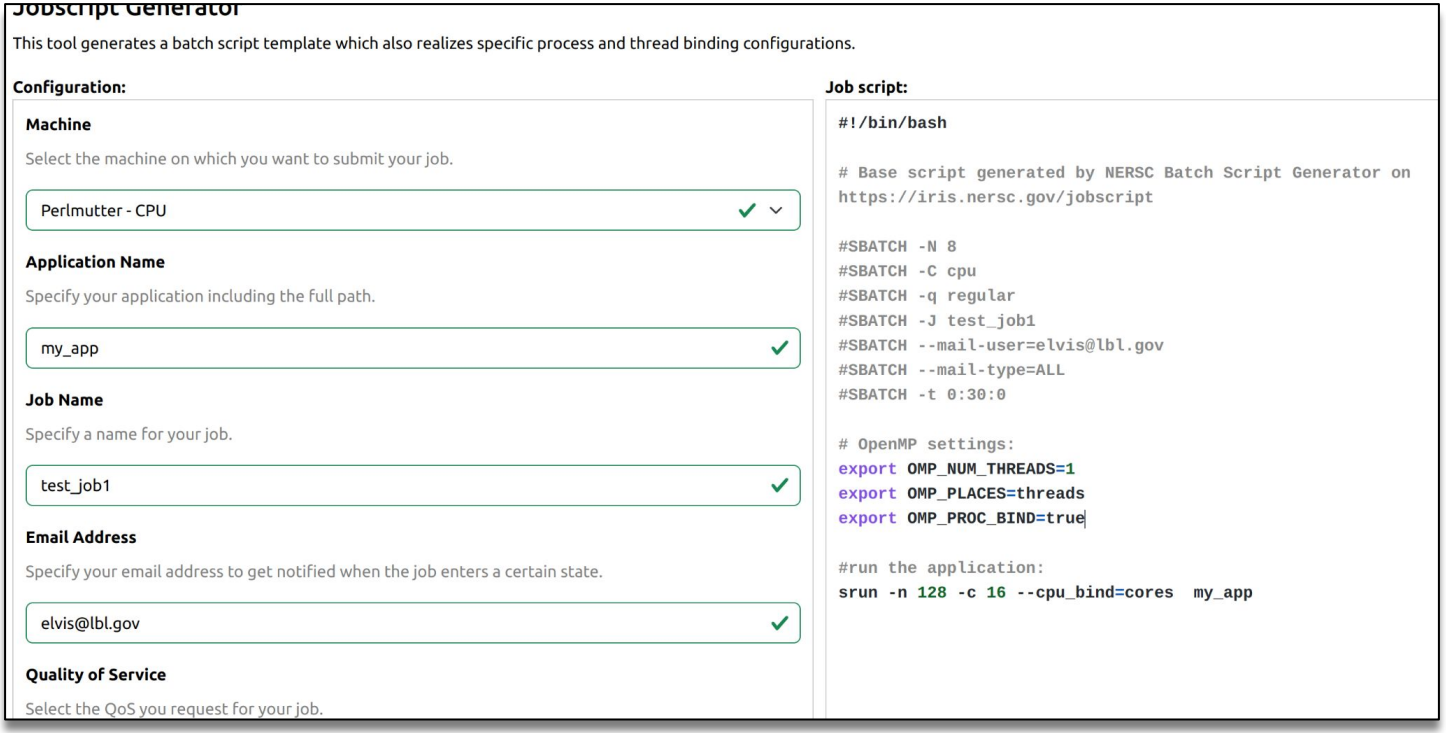
```
#!/bin/bash

#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
```

Hybrid MPI & OpenMP

```
#!/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
```

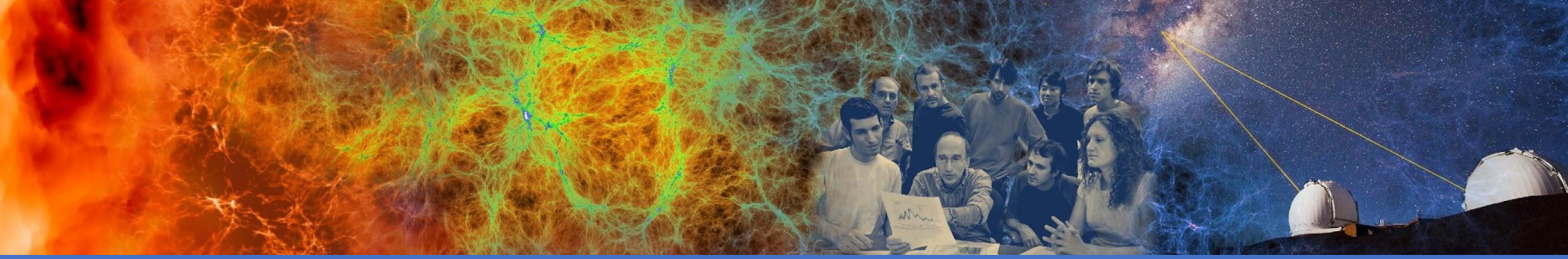
NERSC Job Script Generator

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/jobscript



Spack: Even More Software!

Spack: A Package Manager for Supercomputers

Spack currently contains 6617 mainline packages

NERSC has two instances of Spack available on Perlmuter:

Instance	Version	Command	# of Packages	Can I Install More?
Spack	0.17.1	module load spack	**130**	Yes
Extreme-scale Scientific Software Stack (E4S)	0.18.0.dev0	module load e4s/22.05	**319**	Yes

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

Installed Spack Packages

E4S

```
epalmer@perlmutter:login36:~> $ spack find
==> 130 installed packages
-- cray-cnl7-haswell / intel@19.1.2.254 --
cfitsio@4.0.0

-- cray-sles15-zen3 / gcc@11.2.0 ---------
ant@1.10.7                glproto@1.4.17
arpack-ng@3.8.0           gperf@3.1
bdftopcf@1.0.5            hypre@2.24.0
bison@3.0.4               icu4c@67.1
bzip2@1.0.6               inputproto@2.3.2
cfitsio@4.0.0             intltool@0.51.0
claw@2.0.3                jdk@11.0.1_13
cmake@3.17.0              kbproto@1.0.7
cmake@3.22.0              krb5@1.19.2
cmake@3.22.1              lcms@2.9
cmake@3.22.2              libbsd@0.11.3
cray-libsci@21.08.1.2     libffi@3.3
cuda@11.7.64              libfontenc@1.1.3
curl@7.66.0               libgit2@1.1.1
docbook-xml@4.4           libice@1.0.9
docbook-xsl@1.79.2        libiconv@1.16
expat@2.4.4               libjpeg@9d
fixesproto@5.0            libmd@1.0.3
flex@2.6.4                libpng@1.6.37
font-util@1.3.2           libpthread-stubs@0
fontconfig@2.13.94        libsm@1.2.3
fontsproto@2.1.3          libssh2@1.10.0
freetype@2.11.1           libtiff@4.3.0
fribidi@1.0.5             libx11@1.7.0
ghostscript-fonts@8.11    libxau@1.0.8

-- cray-sles15-zen3 / nvhpc@21.11 --------
cairo@1.16.0  gperf@3.1  intel-mkl@2020.4
```

Spack

```
==> 319 installed packages
-- cray-sles15-zen3 / cce@13.0.2 ----------------------------
adios2@2.8.0     cray-libsci@21.08.1.2  gettext@0.21          libarchive@3.5.2  libmd@1.0.4     mbedtls@2.28.0  python@3.8.13  sundials@6.2.0          xz@5.2.5
arpack-ng@3.8.0  cray-mpich@8.1.15       hdf5@1.10.7           libarchive@3.5.2  libpng@1.6.37   metis@5.1.0     readline@7.0   superlu@5.3.0           zfp@0.5.5
bzip2@1.0.6      curl@7.66.0            hdf5@1.12.1           libbsd@0.11.5     libuv@1.44.1    ncurses@6.1     rhash@1.4.2    superlu-dist@7.2.0      zlib@1.2.12
c-blosc@1.21.1   diffutils@3.6          hypre@2.24.0          libfabric@1.15.0.0  libxml2@2.9.13  parmetis@4.0.3  slepc@3.17.1   sz@2.1.12               zstd@1.5.2
cmake@3.23.1     expat@2.4.8            kokkos@3.6.00         libffi@3.4.2      lz4@1.9.3       petsc@3.17.1    snappy@1.1.8   tar@1.30
cmake@3.23.1     gdbm@1.23              kokkos-kernels@3.6.00 libiconv@1.16     lzo@2.10        pkgconf@1.8.0   sqlite@3.38.5  util-linux-uuid@2.33.1

-- cray-sles15-zen3 / gcc@11.2.0 ----------------------------
adios2@2.8.0           dyninst@12.1.0         hpctoolkit@2022.04.15  libiberty@2.37          metis@4.0.3          pkgconf@1.8.0          snappy@1.1.8
adios2@2.8.0           eccodes@2.25.0         hpcviewer@2022.03      libice@1.0.9            metis@5.1.0          plasma@21.8.29         sqlite@3.38.5
amrex@22.05            elfutils@0.186         hpx@1.7.1              libiconv@1.16           mfem@4.4.0           proj@8.2.1             strumpack@6.3.1
antlr@2.7.7            elfutils@0.186         hwloc@2.7.1            libjpeg@9e              mkfontdir@1.0.7      py-beniget@0.4.1       strumpack@6.3.1
arpack-ng@3.8.0        esmf@8.2.0             hypre@2.24.0           libjpeg-turbo@2.1.3     mkfontscale@1.1.2    py-cython@0.29.24      sundials@6.2.0
asio@1.21.0            expat@2.4.8            hypre@2.24.0           libjpeg-turbo@2.1.3     mpark-variant@1.4.0  py-gast@0.5.3          superlu@5.3.0
autoconf@2.69          ffmpeg@4.4.1           hypre@2.24.0           libmd@1.0.4             nasm@2.15.05         py-mpi4py@3.1.2        superlu-dist@7.2.0
automake@1.16.5        fftw@3.3.10            icu4c@67.1             libmonitor@2021.11.08   nccmp@1.9.0.1        py-numpy@1.22.3        sz@2.1.12
bdftopcf@1.0.5         fftw@3.3.10            inputproto@2.3.2       libpciaccess@0.16       ncl@6.6.2            py-periodictable@1.5.0 tar@1.30
berkeley-db@18.1.40    findutils@4.6.0        intel-tbb@2020.3       libpng@1.6.37           nco@5.0.1            py-picmistandard@0.0.19 tasmanian@7.7
binutils@2.38          flex@2.6.3             intel-tbb@2020.3       libpthread-stubs@0.4    ncurses@6.1          py-pip@21.3.1          tau@2.31.1
bison@3.8.2            font-util@1.3.2        intel-xed@2022.04.17   libquo@1.3.1            ncview@2.1.8         py-pip@21.3.1          tcsh@6.24.00
blaspp@2021.04.01      fontconfig@2.13.94     kbproto@1.0.7          libsm@1.2.3             netcdf-c@4.8.1       py-ply@3.11            trilinos@13.0.1
blaspp@2021.04.01      fontconfig@2.13.94     kim-api@2.2.1          libtiff@4.3.0           netcdf-c@4.8.1       py-pybind11@2.8.1      udunits@2.2.28
blaspp@2021.04.01      fontsproto@2.1.3       kokkos@3.6.00          libtool@2.4.7           netcdf-c@4.8.1       py-pyparsing@3.0.6     util-linux-uuid@2.33.1
blt@0.4.1              fortrilinos@2.0.0      kokkos@3.6.00          libunwind@1.6.2         netcdf-fortran@4.5.4 py-python@0.10.0       util-macros@1.19.3
boost@1.79.0           freetype@2.11.1        kokkos-kernels@3.6.00  libunwind@1.6.2         netlib-scalapack@2.2.0 py-scipy@1.8.0       vtk-m@1.7.1
boost@1.79.0           fribidi@1.0.12         kokkos-kernels@3.6.00  libuv@1.44.1            ninja@1.10.2         py-setuptools@59.4.0   warpx@22.05
boost@1.79.0           gasnet@2022.3.0        kokkos-nvcc-wrapper@3.2.00 libx11@1.7.0        ninja@1.10.2         py-setuptools@59.4.0   warpx@22.05
boost@1.79.0           gdbm@1.19              lammps@20220107        libxau@1.0.8            nlohmann-json@3.10.5 py-warpx@22.05         warpx@22.05
butterflypack@2.1.1    gdbm@1.23              lapackpp@2021.04.00    libxaw@1.0.13           numactl@2.0.14       py-warpx@22.05         xcb-proto@1.14.1
bzip2@1.0.6            gettext@0.21           lapackpp@2021.04.00    libxcb@1.14             openblas@0.3.20      py-warpx@22.05         xerces-c@3.2.3
c-blosc@1.21.1         gettext@0.21           lapackpp@2021.04.00    libxdmcp@1.1.2          openblas@0.3.20      py-wheel@0.37.0        xerces-c@3.2.3
cairo@1.16.0           git@2.26.2             libaec@1.0.5           libxext@1.3.3           openjdk@11.0.14.1_1  py-wheel@0.37.0        xextproto@7.3.0
cairo@1.16.0           glib@2.72.1            libaec@1.0.5           libxfont@1.5.2          openjpeg@2.3.1       python@3.8.13          xproto@7.0.31
camp@0.2.2             googletest@1.10.0      libarchive@3.5.2       libxml2@2.9.13          openpmd-api@0.14.4   python@3.8.13          xtrans@1.3.5
camp@0.2.2             gperf@3.1              libarchive@3.5.2       libxml2@2.9.13          otf2@2.3             python@3.8.13          xz@5.2.5
cdo@2.0.5              gperftools@2.9.1       libarchive@3.5.2       libxmu@1.1.2            papi@6.0.0.1         python@3.8.13          xz@5.2.5
cmake@3.23.1           grads@2.2.1            libbsd@0.11.5          libxpm@3.5.12           papi@6.0.0.1         qthreads@1.16          yasm@1.3.0
cmake@3.23.1           gromacs@2021.5         libcerf@1.3            libxrender@0.9.10       parallel@20210922    raja@0.14.0            zfp@0.5.5
cmake@3.23.1           gsl@2.7                libdwarf@20180129      libxt@1.1.5             parallel-netcdf@1.12.2 raja@0.14.0          zfp@0.5.5
conduit@0.8.3          gsl@2.7.1              libdwarf@20180129      lz4@1.9.3               parmetis@4.0.3       readline@7.0           zlib@1.2.12
cray-libsci@21.08.1.2  hdf5@1.8.22            libfabric@1.15.0.0     lzo@2.10                pcre@8.45            renderproto@0.11.1     zstd@1.5.2
cray-mpich@8.1.15      hdf5@1.10.7            libffi@3.3             m4@1.4.18               pdt@3.25.1           rhash@1.4.2
cub@1.13.1             hdf5@1.10.8            libffi@3.4.2           makedepend@1.0.5        perl@5.34.1          shapelib@1.5.0
cuda@11.5.0            hdf5@1.12.1            libfontenc@1.1.3       mbedtls@2.28.0          petsc@3.17.1         slate@2021.05.02
curl@7.66.0            hdf5@1.12.1            libgd@2.2.4            memkind@1.13.0          petsc@3.17.1         slate@2021.05.02
diffutils@3.6          hdf5@1.12.1            libgd@2.2.4            meson@0.62.1            petsc@3.17.1         slepc@3.17.1
dyninst@12.1.0         heffte@2.2.0           libgeotiff@1.6.0       meson@0.62.1            pixman@0.40.0        slepc@3.17.1
```

Install or Load a Spack Package

Steps	Already Installed in Spack/E4S	Available via Spack/E4S
1	`spack find -v`	`spack list`
	Match package config. and compiler	Search list of ~6200 avail. packages
2	`spack load <package>`	`spack info <package>`
	Load desired package/package variant into environment	Get information about package options
3		`spack install <package>`
		Installs the package into Spack
4		`spack load <package>`

Extra Step: `spack load --sh <package>`
Shows the modifications made to your environment when a package is loaded.

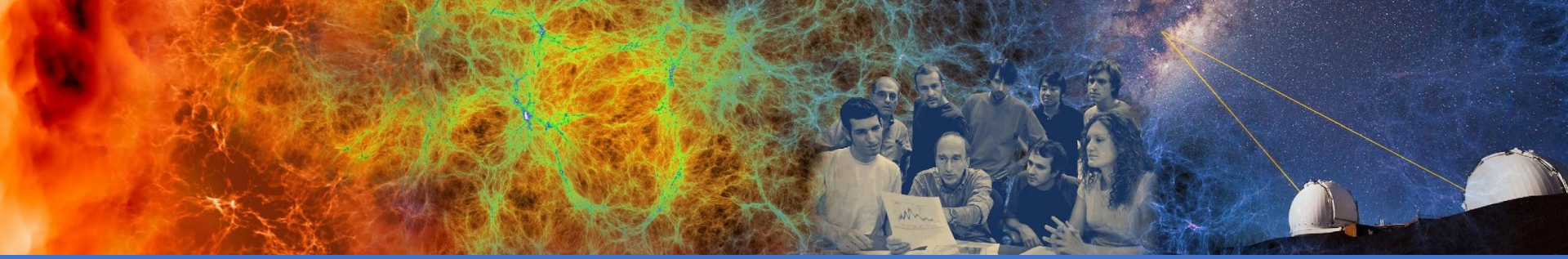
Example: Install a Dependency with Spack

```
epalmer> # In this example, suppose I am compiling a code that depends on a
 package or version of a package that is not currently installed as a module
 on Perlmutter. To demonstrate, I will compile a code that requires
```

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.

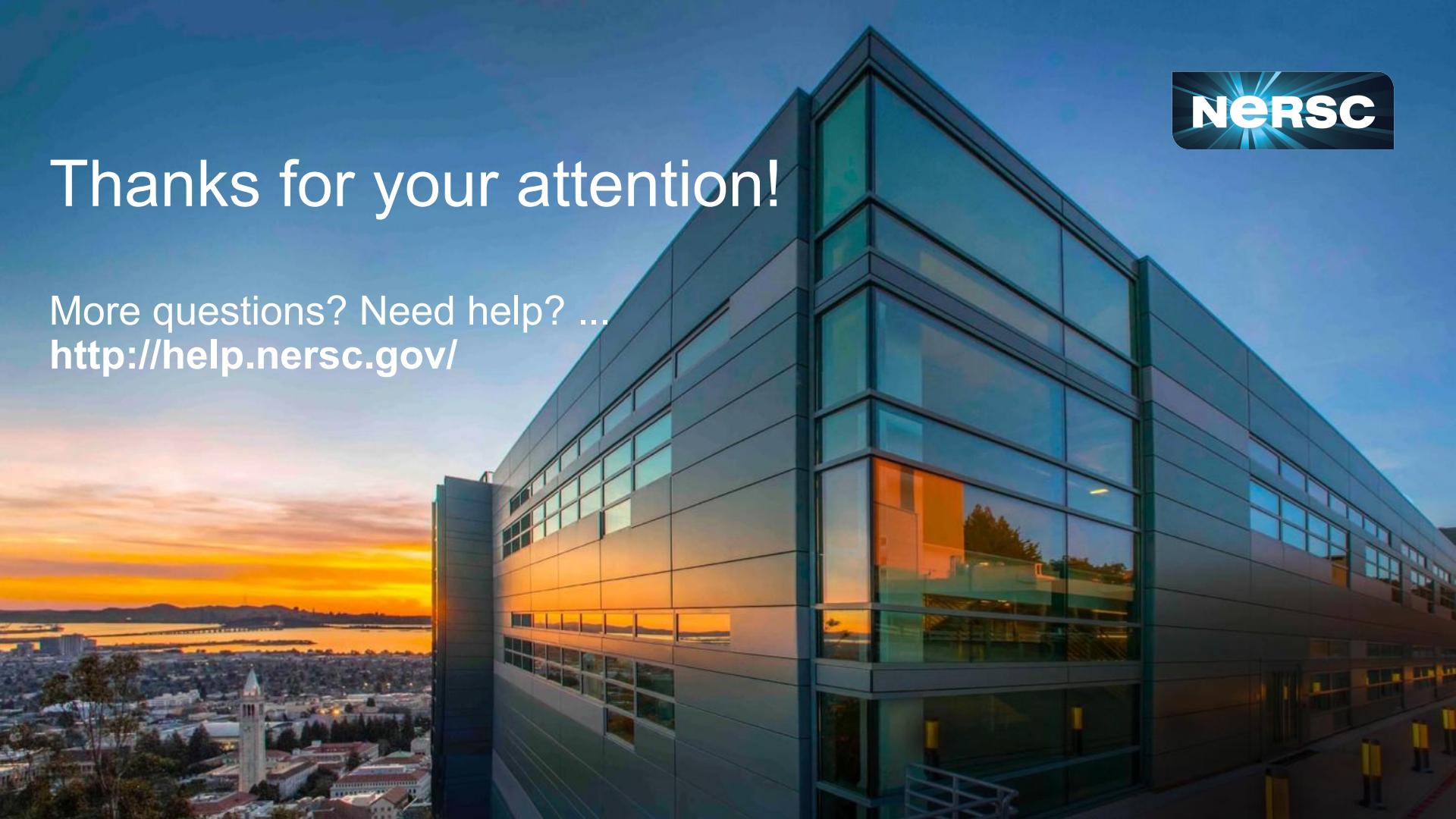


Key Suggestions:

- 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
  - 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/