Introduction to NERSC Resources

LBNL CS Summer Program
June 8, 2023

Helen He
NERSC User Engagement Group
Some Logistics

- Users are muted upon joining Zoom (can unmute to speak)
- Please change your name in Zoom session
  - to: first_name last_name
  - Click “Participants”, then “More” next to your name to rename
- Click the CC button to toggle captions and View Full Transcript
- GDoc is used for Q&A (instead of Zoom chat)
  - https://tinyurl.com/mtva7dar
- Slides and videos will be available on the Training Event page and CSA Summer Program page
  - https://www.nersc.gov/users/training/events/introduction-to-nersc-resources-jun2023/
- Apply for a training account if no NERSC account or MFA not setup yet
  - https://iris.nersc.gov/train, and use the 4-letter code "aO7N"
Outline

- NERSC and Systems Overview
- NERSC Online Resources
- Connecting to NERSC
- File Systems and Data Management / Transfer
- Software Environment / Building Applications
- Running Jobs
- Data Analytics Software and Services
- Hands-on: Compiling and Running Jobs on Perlmutter
NERSC and Systems Overview
NERSC is the Mission HPC Computing Center for the DOE Office of Science

- NERSC deploys advanced HPC and data systems for the broad Office of Science community
- NERSC staff provide advanced application and system performance expertise to users
- Approximately 9,000 users and 900 projects
- Over 2,000 publications cite using NERSC resources per year
- Founded in 1974, focused on open science
- Division of Lawrence Berkeley National Laboratory

<table>
<thead>
<tr>
<th>ASCR</th>
<th>Advanced Scientific Computing Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>BER</td>
<td>Biological &amp; Environmental Research</td>
</tr>
<tr>
<td>BES</td>
<td>Basic Energy Sciences</td>
</tr>
<tr>
<td>FES</td>
<td>Fusion Energy Sciences</td>
</tr>
<tr>
<td>HEP</td>
<td>High Energy Physics</td>
</tr>
<tr>
<td>NP</td>
<td>Nuclear Physics</td>
</tr>
<tr>
<td>SBIR</td>
<td>Small Business Innovation Research</td>
</tr>
</tbody>
</table>
NERSC Systems Roadmap

NERSC-7: Edison
- Multicore CPU
- NESAP Launched: transition applications to advanced architectures

NERSC-8: Cori
- Manycore CPU
- NESAP Launched: transition applications to advanced architectures

NERSC-9: Perlmutter
- CPU and GPU nodes
- Continued transition of applications and support for complex workflows

NERSC-10: Exa system

NERSC-11: Beyond Moore

NERSC-7: Edison
- Multicore CPU

NERSC-8: Cori
- Manycore CPU

NERSC-9: Perlmutter
- CPU and GPU nodes

NERSC-10: Exa system

NERSC-11: Beyond Moore

Increasingly energy-efficient architectures

2013

2016

2021

2024

2028
NERSC Systems

Perlmutter
1,536 NVIDIA A100 accelerated nodes
4 A100 GPUs & 1 AMD ‘Milan’ CPU per node
384 TB (CPU) + 240 TB (GPU) memory
HPE Cray Slingshot high speed interconnect
World’s 7th most powerful supercomputer
140 PF Peak
Pre-production system

Cori (retired 5/31/2023)
9,600 Intel Xeon Phi “KNL” manycore nodes
2,000 Intel Xeon “Haswell” nodes
700,000 processor cores, 1.2 PB memory
Cray XC40 / Aries Dragonfly interconnect
30 PF Peak

DTNs, Spin, Gateways
2 x 100 Gb/s SDN

Ethernet & IB Fabric
Science Friendly Security
Production Monitoring
Power Efficiency
LAN

HPSS Tape Archive
~200 PB

1.5 TB/s
2 PB Burst Buffer
28 PB Scratch

120 PB Common File System
275 TB /home

NERSC Systems
HPSS Tape Archive
~200 PB
50 GB/s

100 GB/s
2 PB Burst Buffer
28 PB Scratch

120 PB Common File System
275 TB /home

NERSC Systems
HPSS Tape Archive
~200 PB
50 GB/s

100 GB/s
2 PB Burst Buffer
28 PB Scratch

120 PB Common File System
275 TB /home
NERSC Online Resources
• https://www.nersc.gov
• Science, News, Publications
• Contact Us
• Live Status (MOTD) https://www.nersc.gov/live-status/motd/
• NUG (and Slack)
• Training Events https://www.nersc.gov/users/training/events/
NERSC YouTube Channel

https://www.youtube.com/c/NERSC

Training sessions and other NERSC events presentations are archived on youtube, with professional captions
User Slack; User Appointments

https://www.nersc.gov/users/NUG/

https://docs.nersc.gov/getting-started/#appointments-with-nersc-user-support-staff
Technical Documentations
https://docs.nersc.gov

- Getting Started
  https://docs.nersc.gov/getting-started/

Home
Getting Started
Tutorials
Accounts
Iris
Systems
Storage Systems
Connecting
Environment
Policies
Development
Developer Tools
Running Jobs
Applications
Analytics
Machine Learning
Performance
Services
Science Partners
Acronyms
Contributed Tips and Tricks
Current Known Issues

NERSC Documentation
Getting Started
Tutorials
Accounts
Iris
Systems
Storage Systems
Connecting
Environment
Policies
Development
Developer Tools
Running Jobs
Applications
Analytics
Machine Learning
Performance
Services
Science Partners
Acronyms
Contributed Tips and Tricks
Current Known Issues

About this page
This document will guide you through the basics of using NERSC's supercomputers, storage systems, and services.

Welcome to the National Energy Research Scientific Computing Center (NERSC)!

- Talk NERSC Overview at New User Training event, September 28, 2022 - Slides, Video

Computing Resources
Perlmuter
Perlmuter is a HPE Cray EX supercomputer with over 1500 GPU-accelerated compute nodes.

- Perlmuter system information
  - NERSC Live status

Storage Resources
File systems are configured for different purposes. Perlmuter has access to at least three different file systems with different levels of performance, permanence and available space.
IRIS

- IRIS: NERSC Account Management and Reporting: https://iris.nersc.gov
  - Account info
  - Change password
  - Change contact info
  - SSH Keys, MFA
  - Check usage info
Help Portal

https://help.nersc.gov

- Submit tickets (ask questions)
- All my tickets
- All my projects tickets
- Request forms:
  - Quota Increase
  - Reservations, ...
- Book consulting appo
- NERSC user Slack
- Allocation (ERCAP) requests
- Iris
MyNERSC

https://my.nersc.gov

- Dashboard
- Jobs
- Center Status
- File Browser
- Service Tickets
- Data Dashboard
- Jupyter Hub
- Links to other useful pages
https://my.nersc.gov Leads You to All Sites

- help.nersc.gov
- jupyter.nersc.gov
- www.nersc.gov
- docs.nersc.gov
- iris.nersc.gov

- my disk quota
- is Perlmutter up?
- my jobs
Prefer for Perlmutter

Please check the Transitioning Applications to Perlmutter webpage for a wealth of useful information on how to transition your applications for Perlmutter.
Connecting to NERSC
Multi-Factor Authentication (MFA) and sshproxy

- NERSC password + OTP ("One-Time Password")
  - OTP obtained via the “Google Authenticator” app on your smartphone
  - Alternative/backup option: Authy on desktop https://authy.com/
- MFA is used in login to NERSC systems, web sites, and services
  - Setup MFA https://docs.nersc.gov/connect/mfa/
- sshproxy.sh creates a short-term certificate
  - Run sshproxy.sh once, then you can ssh to NERSC systems for the next 24 hours before being asked for password+OTP again
  - https://docs.nersc.gov/connect/mfa/#sshproxy
SSH and MFA Examples

$laptop>$ ssh elvis@perlmutter.nersc.gov
...
Password + OTP:
elvis@perlmutter:login32:~>

You will login to one of the login nodes (40 on Perlmutter).

To allow X-forwarding to access visualization programs, use the “-Y” flag:

```
localhost% ssh -Y elvis@perlmutter.nersc.gov
e/elvis> module load matlab
e/elvis> matlab
<MATLAB starts up>
```
Connecting to NERSC: NX

- NERSC recommends using NX instead of SSH X-forwarding since NX is faster and more reliable
- NX is a service for Accelerated X
- NX also has the benefit of long lasting terminal sessions that can survive between lost internet connections
  - Can reconnect later, even from a different location or computer
- Download and install the **Client** software: NoMachine
  - [https://docs.nersc.gov/connect/nx](https://docs.nersc.gov/connect/nx)
  - Works on Window/Mac/Linux
NoMachine

- Could also setup with sshproxy so only need to authenticate once per day
Terminal in Jupyter

You can access Perlmutter from any web browser, via https://jupyter.nersc.gov
File Systems and Data Management / Transfer
Simplified NERSC File Systems

- **Memory**
- **Scratch**
- **Community**
- **HPSS**

- **35 PB (Perlmutter) Flash Scratch**
  - Lustre >5 TB/s
  - temporarily (purge)

- **157 PB HDD Community**
  - Spectrum Scale (GPFS)
  - 150 GB/s, permanent

- **150 PB Tape Archive**
  - HPSS Forever

- **20 TB SSD Software**
  - Spectrum Scale
  - Permanent
  - Faster compiling / Source Code

- **Global Common**
- **Global Home**
### Global File Systems

#### Global Home
- Permanent, relatively small storage
- Mounted on all platforms
- NOT tuned to perform well for parallel jobs
- Quota cannot be changed
- Snapshot backups (7-day history)
- **Perfect for storing data such as source code, shell scripts**

#### Community File System (CFS)
- Permanent, larger storage
- Mounted on all platforms
- Medium performance for parallel jobs
- Quota can be changed
- Snapshot backups (7-day history)
- **Perfect for sharing data within research group**
Local File Systems

Scratch

- Large, temporary storage
- Optimized for read/write operations, NOT storage
- Not backed up
- Purge policy (8 weeks)
- Perfect for staging data and performing computations
HPSS: Long Term Storage System

- High-Performance Storage System
- Archival storage of infrequently accessed data
- Use `hsi` and `htar` to put/get files between NERSC computational systems and HPSS
- [https://docs.nersc.gov/filesystems/archive/](https://docs.nersc.gov/filesystems/archive/)
Software Environment and Building Applications
Software

- Cray supercomputers OS is a version of Linux
- Compilers are provided on machines
- Libraries: many libraries provided by vendor and by NERSC
- Applications: NERSC compiles and supports many software packages (such as chemistry and materials sciences packages) for our users
- DOE Extreme-scale Scientific Software Stack (E4S): open-source projects, including xSDK, dev-tools, math-libraries, compilers, and more
## Modules Environment

- LMod is used to manage the user environment
  - [https://docs.nersc.gov/environment/#nersc-modules-environment](https://docs.nersc.gov/environment/#nersc-modules-environment)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td></td>
</tr>
<tr>
<td>list</td>
<td>To list the modules in your environment</td>
</tr>
<tr>
<td>spider &lt;name&gt;</td>
<td>To list available modules with &lt;name&gt; as substring, and how to load</td>
</tr>
<tr>
<td>load/unload ..</td>
<td>To load or unload module</td>
</tr>
<tr>
<td>swap .. ..</td>
<td>To swap modules</td>
</tr>
<tr>
<td>show/display ..</td>
<td>To see what a module loads, what env a module sets</td>
</tr>
<tr>
<td>whatis ..</td>
<td>Display the module file information</td>
</tr>
<tr>
<td>help ..</td>
<td>General help: $module help</td>
</tr>
<tr>
<td></td>
<td>Information about a module: $ module help PrgEnv-cray</td>
</tr>
</tbody>
</table>
Default Modules Loaded at Login (GPU Environment)

Modules Loaded by Default:

1) craype-x86-milan                        7) craype/2.7.16                        13) darshan/3.4.0
2) libfabric/1.15.0.0                     8) cray-dsmml/0.2.2                    14) Nsight-Compute/2022.1.1
3) craype-network-ofi                     9) cray-mpich/8.1.17                   15) Nsight-Systems/2022.2.1
4) perftools-base/22.06.0                 10) cray-libsci/21.08.1.2              16) cudatoolkit/11.7
5) xpmem/2.4.4-2.3_12.2__gff0e1d9.shasta 11) PrgEnv-gnu/8.3.3                    17) craype-accel-nvidia80
6) gcc/11.2.0                            12) xalt/2.10.2                        18) gpu/1.0

- CPU Architecture
- Default Programming Environment, Compiler, MPI, Scientific Libraries
- GPU Architecture, CUDA-Aware MPI, GPU Profilers

- CUDA-aware MPI is enabled by default
- Modules cudatoolkit, craype-accel-nvidia80, and gpu are loaded by default.
- gpu module also sets MPICH_GPU_SUPPORT_ENABLED to 1.
# Default Modules for CPU-only Code

For CPU-only code we recommend:

```
module load cpu
```

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

- **CPU Architecture**
- Default Programming Environment, Compiler, MPI and Scientific Libraries
- Configured for CPU-only MPI
Software Environment

- Available compilers: GNU, Nvidia, CCE, (and Intel, in progress)
- It calls native compilers for each compiler (such as gfortran, gcc, g++, etc.) underneath.
  - Do not use native compilers directly
  - ftn for Fortran codes: `ftn my_code.f90`
  - cc for C codes: `cc my_code.c`
  - CC for C++ codes: `CC my_code.cc`
- Compiler wrappers add header files and link in MPI and other loaded Cray libraries by default
  - Builds applications dynamically by default.
Building Sample Program on CPU

- module load cpu
- Using default GNU compiler
  
  ftn -o mytest mytest.f90 (MPI code)
  cc -fopenmp -o mytest mytest_hybrid.c (hybrid MPI/OpenMP code)

- Using Nvidia compiler
  module load PrgEnv-nvidia
  cc -o mytest mytest_code.c (MPI code)
  cc -mp -o mytest_hybrid mytest_hybrid.c (MPI/OpenMP hybrid code)
## Perlmutter Supports Every GPU Programming Model

<table>
<thead>
<tr>
<th></th>
<th>Fortran/C/C++</th>
<th>CUDA</th>
<th>OpenACC 2.x</th>
<th>OpenMP 5.x</th>
<th>CUDA Fortran</th>
<th>Kokkos/Raja</th>
<th>MPI</th>
<th>HIP</th>
<th>DPC++/SYCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Vendor Supported**
- **NERSC Supported in progress**
Building CUDA Program on GPU

- module load gpu
- Using default GNU compiler
  CC -o mytest mytest.cpp
- using Nvidia compiler
  module load PrgEnv-nvidia
  CC -cuda -o mytest mytest.cpp
Building OpenMP Offload Program on GPU

- module load gpu
- using Nvidia compiler
  module load PrgEnv-nvidia
  ftn -mp=gpu -o mytest mytest.f90
  cc -mp=gpu -o mytest mytest.c
  CC -mp=gpu -o mytest mytest.cc

- Using CCE compiler
  module load PrgEnv-cray
  ftn -O3 -h omp -h noacc -o mytest mytest.f90
  cc -Ofast -fopenmp -o mytest mytest.c
  CC -Ofast -fopenmp -o mytest mytest.cc
Building Applications on Perlmutter

● More info on building for Perlmutter GPU
  ○ https://docs.nersc.gov/systems/perlmutter/#compilingbuilding-software

● More info on porting and optimizing for GPU on Perlmutter
  Readiness page
  ○ https://docs.nersc.gov/performance/readiness/
  ○ Basic GPU concepts and programming considerations, programming
    models, running jobs, machine learning applications, libraries, profiling
    tools, IO, case studies, …
Running Jobs
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**
- Typical run times are a few to 10s of hours
  - Limits are necessary because of MTBF and the need to accommodate 9,000 users’ jobs
Login Nodes and Compute Nodes

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

- **Compute nodes**
  - Execute your application
  - Dedicated resources for your job
  - Perlmutter has CPU and GPU compute nodes
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**

---

NERSC

[NERSC Logo]

[NERSC Website]

[NERSC Contact]

43
My First “Hello World” Program

my_batch_script:

#!/bin/bash
#SBATCH -q debug
#SBATCH -N 2
#SBATCH -t 10:00
#SBATCH -C cpu
##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 cpu -t 10:00
<wait_for_session_prompt. Land on a compute node>
% srun -n 64 ./helloWorld
To obtain processor info:

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

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

- 2 sockets 4 NUMA domains/socket (8/node)
- 128 physical cores
- 256 logical cores
- Memory access on remote NUMA domains are slower
Sample Perlmutter CPU Batch Script - MPI

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

srun -n 1280 -c 8 --cpu_bind=cores ./mycode.exe
```

- There are 256 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
Sample Perlmutter CPU Batch Script - Hybrid MPI/OpenMP

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

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

srun -n 320 -c 32 --cpu-bind=cores ./mycode.exe
```

- Set OMP_NUM_THREADS
- Use OpenMP standard settings for process and thread affinity
- Again, “-c” specifies #_logical_CPUs to be allocated to each MPI task
  - with 8 MPI tasks per node, set 256 logical CPUs / 8 =32 for ”-c”
  - “-c” value should be >= OMP_NUM_THREADS
# CPU and GPU Compute Nodes Affinity

<table>
<thead>
<tr>
<th></th>
<th>Perlmutter CPU</th>
<th>CPU on Perlmutter GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical cores</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>Logical CPUs per physical core</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Logical CPUs per node</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>NUMA domains</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td><code>-c</code> value for <code>srun</code></td>
<td>$2 \times \text{floor}(128/\text{tpn})$</td>
<td>$2 \times \text{floor}(64/\text{tpn})$</td>
</tr>
</tbody>
</table>

**tpn = Number of MPI tasks per node**
Process / Thread / Memory Affinity

- Correct process, thread and memory affinity is critical for getting optimal performance on Perlmutter CPU and GPU
  - 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

- Both -c xx and --cpu-bind=cores are essential, otherwise multiple processes may land on the same core, while other cores are idle, hurting performance badly

- [https://docs.nersc.gov/jobs/affinity/](https://docs.nersc.gov/jobs/affinity/)
Use salloc to Run Debug and Interactive Jobs

- You can run small parallel jobs interactively on dedicated nodes
- Debug
  - Max 8 nodes, up to 30 min
    % salloc -N 20 -q debug -C cpu -t 30:00
- Interactive (highly recommend to use this!!)
  - Instant allocation (get nodes in 6 min or reject)
  - Max 4 nodes, walltime 4 hrs
    % salloc -N 2 -q interactive -C cpu -t 2:00:00
  - More information
    - [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/)
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 128 (Perlmutter CPU) jobs from different users depending on their memory requirements

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

- Charged by a fraction of a node used
- [https://docs.nersc.gov/jobs/examples/#shared](https://docs.nersc.gov/jobs/examples/#shared)
- Also available on Perlmutter GPU
Bundle Jobs

Multiple Jobs Sequentially:
```bash
#!/bin/bash
#SBATCH --qos=debug
#SBATCH --nodes=4
#SBATCH --time=10:00
#SBATCH --licenses=cfs,scratch
#SBATCH --constraint=cpu

# each srun uses 4 nodes
srun -n 128 -c 8 --cpu_bind=cores ./a.out
srun -n 64 -c 16 --cpu_bind=cores ./b.out
srun -n 32 -c 32 --cpu_bind=cores ./c.out
```

Multiple Jobs Simultaneously:
```bash
#!/bin/bash
#SBATCH --qos=debug
#SBATCH --nodes=8
#SBATCH --time=30:00
#SBATCH --licenses=scratch
#SBATCH --constraint=cpu

# 3 sruns combined use 8 nodes
srun -N 2 -n 176 -c 2 --cpu_bind=cores ./a.out &
srun -N 4 -n 432 -c 2 --cpu_bind=cores ./b.out &
srun -N 2 -n 160 -c 2 --cpu_bind=cores ./c.out &
wait
```

- 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-simultaneously
Dependency Jobs

```bash
perlmutter% sbatch job1
Submitted batch job 1655447

perlmutter% sbatch --dependency=afterok:165547 job2
or
perlmutter% sbatch --dependency=afterany:165547 job2
```

```bash
perlmutter% sbatch job1
Submitted batch job 1655447

perlmutter% cat job2
#!/bin/bash
#SBATCH -q regular
#SBATCH -N 1
#SBATCH -t 1:30:00
#SBATCH -d afterok:1655447
#SBATCH -C cpu
srun -n 64 -c 4 –cpu-bind=cores ./a.out

perlmutter% sbatch job2
```

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

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
Use Workflow Management Tools

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

  It is inefficient and overwhelms Slurm scheduler

- Available workflow tools include: GNU parallel, Taskfarmer, Fireworks, Nextflow, Papermill, etc.
- One usage case is to pack large number of serial jobs into one script
- [https://docs.nersc.gov/jobs/workflow-tools/](https://docs.nersc.gov/jobs/workflow-tools/)
GNU Parallel Is Better Than Shared QOS

```
perlmutter% module load parallel

perlmutter% seq 1 5 | parallel -j 2 'echo "Hello world {}!"; sleep 10; date'
Hello world 1!
Wed 07 Jun 2023 10:22:11 PM PDT
Hello world 2!
Wed 07 Jun 2023 10:22:11 PM PDT
Hello world 3!
Wed 07 Jun 2023 10:22:21 PM PDT
Hello world 4!
Wed 07 Jun 2023 10:22:21 PM PDT
Hello world 5!
Wed 07 Jun 2023 10:22:31 PM PDT
```

- Packed jobs have massively reduced total queue wait
  - Can also pack single-node tasks into multiple node jobs
- No risk of Slurm overload
- Run combinations of tasks in parallel and sequence
- Easy input substitution
  - If you need it, much more power is available
- Superior to task arrays, too
- [https://docs.nersc.gov/jobs/workflow/gnuparallel/](https://docs.nersc.gov/jobs/workflow/gnuparallel/)
Sample GPU Job Script

```bash
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=gpu
#SBATCH --job-name=myjob
#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=2
#SBATCH --gpus-per-node=4

export OMP_NUM_THREADS=1
srun -n 128 --cpu-bind=cores --gpu-bind=closest <executable>
```

c = 2*floor(64/tpn)

Where:
  tpn = ntasks-per-node

- By default all processes will have access to all GPUs.
- A round robin assignment does not guarantee affinity.
- To guarantee that closest GPU is assigned: -gpus-bind=closest
- To bind ranks to individual cores: -cpu-bind=cores
1 Node, 4 Tasks, 4 GPUs

1 GPU visible to each task

#!/bin/bash
#SBATCH -A ntrain3
#SBATCH -C gpu
#SBATCH -q regular
#SBATCH -t 1:00:00
#SBATCH -N 1
#SBATCH --ntasks-per-node=4
#SBATCH -c 32
#SBATCH --gpus-per-task=1
export SLURM_CPU_BIND="cores"
srun ./gpus_for_tasks

# Default for --gpus-per-task=1 is 1 task only see 1 GPU

4 GPUs visible to each task

#!/bin/bash
#SBATCH -A ntrain3
#SBATCH -C gpu
#SBATCH -q debug
#SBATCH -t 10:00
#SBATCH -N 1
#SBATCH --ntasks-per-node=4
#SBATCH -c 32
#SBATCH --gpus-per-task=1
#SBATCH --gpu-bind=none
export SLURM_CPU_BIND="cores"
srun ./gpus_for_tasks

# Default for --gpus-per-task=1 and --gpu-bind=none is each task sees all GPU
## Perlmutter CPU Queue Policy (as of June 2023)

<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>-</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>high</td>
<td>1</td>
</tr>
<tr>
<td>jupyter</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>high</td>
<td>1</td>
</tr>
<tr>
<td>debug</td>
<td>8</td>
<td>0.5</td>
<td>5</td>
<td>2</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>shared²</td>
<td>0.5</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>preempt</td>
<td>128</td>
<td>24</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>0.5</td>
</tr>
<tr>
<td>overrun</td>
<td>-</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>very low</td>
<td>0</td>
</tr>
<tr>
<td>realtime</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>very high</td>
<td>1</td>
</tr>
<tr>
<td>QOS</td>
<td>Max nodes</td>
<td>Max time (hrs)</td>
<td>Submit limit</td>
<td>Run limit</td>
<td>Priority</td>
<td>QOS Factor</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>----------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>regular</td>
<td>-</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>high</td>
<td>1</td>
</tr>
<tr>
<td>jupyter</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>high</td>
<td>1</td>
</tr>
<tr>
<td>debug</td>
<td>8</td>
<td>0.5</td>
<td>5</td>
<td>2</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>shared</td>
<td>0.5</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>preempt</td>
<td>128</td>
<td>24 (preemptible after two hours)</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>0.25</td>
</tr>
<tr>
<td>overrun</td>
<td>-</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>very low</td>
<td>0</td>
</tr>
<tr>
<td>realtime</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>very high</td>
<td>1</td>
</tr>
</tbody>
</table>
NERSC Job Script Generator

[Image of the NERSC Job Script Generator interface]

The image shows the interface for the NERSC Job Script Generator, which allows users to generate batch script templates for submitting jobs to the NERSC supercomputer. The interface includes fields for specifying job information such as machine, application name, job name, email address, quality of service, and wallclock time. Example code snippets are displayed, including commands for bash scripts and OpenMP settings.

Example Code Snippet:

```bash
#!/bin/bash
#SBATCH -N 4
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -t 01:30:00

#OpenMP settings:
export OMP_NUM_THREADS=4
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

#run the application:
srun -n 32 -c 32 --cpu_bind=cores myapp.x
```
Monitoring Your Jobs

- Jobs are waiting in the queue until resources are available
- Overall job priorities are a combination of QOS, queue wait time, job size, wall time request, etc.
- You can monitor with
  - `squeue`: Slurm native command
  - `sqs`: NERSC custom wrapper script
  - `sacct`: Query Completed and Pending Jobs
  - [https://docs.nersc.gov/jobs/monitoring/](https://docs.nersc.gov/jobs/monitoring/)
- On the web
  - [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
Data Analytics Software and Services
# Production Data Software Stack

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Transfer + Access</td>
<td>globus online GridFTP jupyter newt python django</td>
</tr>
<tr>
<td>Workflows</td>
<td>Parsl GNUparallel papermill FireWorks TaskFarmer</td>
</tr>
<tr>
<td>Data Management</td>
<td>HDF netCDF ROOT mongoDB MySQL PostgreSQL</td>
</tr>
<tr>
<td>Data Analytics</td>
<td>python Spark julia TensorFlow PyTorch</td>
</tr>
<tr>
<td>Data Visualization</td>
<td>ParaView visit</td>
</tr>
</tbody>
</table>

---

[NERSC Logo] [NERSC Logo] [BERKELEY LAB Logo] [U.S. DEPARTMENT OF ENERGY Logo] [Office of Science Logo]
Data Analytic Software Services

- Globus Online
- Science Gateways
- Databases
- Shifter / Podman
- Python
- Jupyter
- Machine Learning / Deep Learning
- Workflows
- And more …
Globus Online: Move Data

- [https://www.globus.org](https://www.globus.org)  [https://docs.nersc.gov/services/globus/](https://docs.nersc.gov/services/globus/)
- The recommended tool for moving data in&out of NERSC
  - Reliable & easy-to-use web-based service:
    - Automatic retries
    - Email notification of success or failure
  - NERSC managed endpoints for optimized data transfers
    - NERSC DTN (dedicated data transfer system), NERSC Perlmutter, NERSC HPSS, etc.
  - Other Center has endpoints, such as OLCF DTN
  - Setup [Globus Connect Personal](https://www.globus.org/identity/globus-connect/) to ease transfer between local system (such as laptop) and NERSC systems
Globus File Transfer Example

Please authenticate to access OLCF DTN

When you press the "CONTINUE" button below you will be redirected to the collection’s login page. After logging in, you will be returned here.
Data Transfer General Tips

- Use Globus Online for large, automated or monitored transfers
- `cp`, `scp`, or `rsync` is fine for smaller, one-time transfers (<100 MB)
  - But note that Globus is also fine for small transfers
- Use **give-and-take** to share files between NERSC users
  - `% give -u <receiving_user> <file or directory>`
  - `% take -u <sending_user> <filename>`
Access for External Collaborators

- **Web Portals**
  - NERSC supports project-level public http access
    - Project specific area can be created:
      - `/global/cfs/cdirs/<your_project>/www`
    - These are available for public access under the URL:
      - `http://portal.nersc.gov/cfs/<your_project>`
      - Each repo has a /project space, can publish as above

- **Special Science Gateways** can be created.
  - Sophisticated ones can be made with SPIN
    - [https://docs.nersc.gov/services/spin/](https://docs.nersc.gov/services/spin/)
    - [https://www.nersc.gov/users/training/spin/](https://www.nersc.gov/users/training/spin/) (SPIN workshop required)
  - Details at: [https://docs.nersc.gov/services/science-gateways/](https://docs.nersc.gov/services/science-gateways/)
Shifter

- NERSC R&D effort, in collaboration with Cray, to support Docker Application images
- “Docker-like” functionality on the Cray and HPC Linux clusters. Enables users to run custom environments on HPC systems.
- Addresses security issues in a robust way
- Efficient job-start & Native application performance

https://docs.nersc.gov/development/shifter/how-to-use/
Shifter Accelerates Python Applications

- Shifter is especially helpful for Python applications
- A large number of shared libraries needed on compute nodes before execution
FROM ubuntu:14.04
MAINTAINER Shane Canon scanon@lbl.gov
# Update packages and install dependencies
RUN apt-update -y && \
    apt-get install -y build-essential

# Copy in the application
ADD . /myapp
# Build it
RUN cd /myapp && \
    make && make install

laptop> docker build -t scanon/myapp:1.1 .
laptop> docker push scanon/myapp:1.1
Use the Image with Shifter

#!/bin/bash
#SBATCH -N 16 -t 20
#SBATCH --image=scanon/myapp:1.1

module load shifter
export TMPDIR=/mnt
srun -n 16 shifter /myapp/app

cori> shifterimg pull scanon/myapp:1.1
cori> sbatch ./job.sl
Try this: Podman

• Podman (Pod manager) is an Open Container Initiative compliant container framework under active development by Red Hat
• Free and open source
• Usable anywhere (including your laptop), not just NERSC
• Can provide rootless containers, which give users the ability to run as root within their image while still maintaining security
• Will allow users to build images on Perlmutter login nodes
• Performance in most cases should be similar to what is currently possible with Shifter (i.e. it’s fast!)
• https://docs.nersc.gov/development/podman-hpc/overview/
Python

- Extremely popular interpreted language, continuing to grow
- Libraries like NumPy, SciPy, scikit-learn commonly used for scientific analysis
- Are used for ML/DL
- Python is fully supported at NERSC - we use Anaconda Python to provide pre-built environments and the ability for users to create their own environments
Python

- Avoid running “conda init” which will hardcode conda initialization in your shell startup file ($HOME/.bashrc)
- Do not use /usr/bin/python, instead:
  - module load python
  which already includes basic packages: numpy, scipy, mpi4py
- Guide to use Python on Perlmutter:
  - https://docs.nersc.gov/development/languages/python/using-python-perlmutter
Create a custom conda environment:

perlmutter> module load python
perlmutter> conda create --name myenv --yes python=3.10
perlmutter> conda activate myenv
(myenv) perlmutter> python
Python 3.10.4 (main, Mar 31 2022, 08:41:55) [GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

Use Python inside a Shifter container:

perlmutter> shifter --image=docker:library/python:latest python
Python 3.10.7 (main, Sep 13 2022, 14:31:33) [GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.

https://docs.nersc.gov/development/languages/python/nersc-python/
Building and using mpi4py

• mpi4py provides a Python interface to MPI
• mpi4py is available via `module load python`
• This mpi4py is CUDA-aware (can communicate GPU objects)
• To build your own CUDA-aware mpi4py, follow this recipe:

```
perlmutter> module load PrgEnv-gnu cudatoolkit python
perlmutter> conda create -n cudaaware python=3.9 -y
perlmutter> conda activate cudaaware
perlmutter> MPICC="cc -target-accel=nvidia80 -shared" pip install
--+force-reinstall --no-cache-dir --no-binary=mpi4py mpi4py
```

• Be aware that with any CUDA-aware mpi4py, you must have `cudatoolkit` loaded, even for code that does not use the GPU
Getting started with GPUs in Python

- NumPy and SciPy do not utilize GPUs out of the box

- There are many Python GPU frameworks out there:
  - “drop in” replacements for numpy, scipy, pandas, scikit-learn, etc
    - CuPy, RAPIDS
  - “machine learning” libraries that also support general GPU computing
    - PyTorch, TensorFlow, JAX
  - “I want to write my own GPU kernels”
    - Numba, PyOpenCL, PyCUDA, CUDA Python
  - multi-node / distributed memory:
    - mpi4py+X, dask, cuNumeric
Getting started with GPUs in Python (CuPy)

```bash
> module load python
> conda create -y --name cupy-demo python=3.9 numpy scipy
> conda activate cupy-demo
> pip install cupy-cuda11X
> python
>>> import cupy as cp
>>> print(cp.array([1, 2, 3]))
[1 2 3]
```

Note: cudatoolkit module is loaded by default
Current default version is cudatoolkit/11.7

Check your package documentation to see cudatoolkit compatibility requirements

See documentation at [https://docs.nersc.gov/development/languages/python/using-python-perlmutter/](https://docs.nersc.gov/development/languages/python/using-python-perlmutter/) or open a ticket at [https://help.nersc.gov/](https://help.nersc.gov/)
What is Jupyter?

Interactive open-source web application

Allows you to **create** and **share** documents, “notebooks,” containing:
- Live code
- Equations
- Visualizations
- Narrative text
- Interactive widgets

**Things you can use Jupyter notebooks for:**
- Data cleaning and data transformation
- Numerical simulation
- Statistical modeling
- Data visualization
- Machine learning
- Workflows and analytics frameworks
- Training and Tutorials

[https://docs.nersc.gov/services/jupyter/](https://docs.nersc.gov/services/jupyter/)
# Available Notebook Servers

<table>
<thead>
<tr>
<th></th>
<th>Shared CPU Node</th>
<th>Exclusive CPU Node</th>
<th>Exclusive GPU Node</th>
<th>Configurable Job</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Perlmutter</strong></td>
<td>start</td>
<td>start</td>
<td>start</td>
<td>start</td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td>Use a node shared with other users' notebooks but outside the batch queues.</td>
<td>Use your own node within a job allocation using defaults.</td>
<td>Use multiple compute nodes with specialized settings.</td>
<td></td>
</tr>
<tr>
<td><strong>Use Cases</strong></td>
<td>Visualization and analytics that are not memory intensive and can run on just a few cores.</td>
<td>Visualization, analytics, machine learning that is compute or memory intensive but can be done on a single node.</td>
<td>Multi-node analytics jobs, jobs in reservations, custom project charging, and more.</td>
<td></td>
</tr>
</tbody>
</table>
Available Jupyter Kernels

Your own custom kernels

And many NERSC provided kernels: Python, Julia, ML/DL packages etc.
Your Own Custom Jupyter Kernel

Most common Jupyter question:

“How do I take a conda environment and use it from Jupyter?”

Several ways to accomplish this, here’s the easy one.

$ module load python
$ conda create -n myenv python=3.9 ipykernel <more-packages-to-install>
$ conda activate myenv
(myenv) $ python -m ipykernel install --user --name myenv-jupyter

Point your browser to jupyter.nersc.gov.
(You may need to restart your notebook server via control panel).
Kernel “myenv-jupyter” should be present in the kernel list.
The helper script is the most flexible approach for NERSC users since it easily enables modules.

Meanwhile, in jupyter-helper.sh:
```bash
#!/bin/bash
export SOMETHING=123
module load texlive
exec python -m ipykernel "$@
```

edit: $HOME/.local/share/jupyter/kernels/myenv-jupyter/kernel.json

```json
{
  "argv": [
    "/global/homes/y/yunhe/jupyter-helper.sh",
    "python",
    "-m",
    "ipykernel_launcher",
    "-f",
    "{connection_file}"
  ],
  "display_name": "myenv-jupyter2",
  "language": "python",
}
```
NERSC Deep Learning Software Stack Overview

https://docs.nersc.gov/machinelearning/

Frameworks:
- TensorFlow
- Keras
- PyTorch

Distributed training libraries:
- Horovod
- PyTorch distributed
- Cray Plugin

Productive tools and services:
- Jupyter, Shifter
How to Use NERSC DL Software Stack

- We have modules you can load which contain python and DL libraries
  - module load tensorflow/<version>
  - module load pytorch/<version>
- You can install your own packages on top to customize
  - pip install --force-reinstall --no-cache-dir --user MY-PACKAGE
- Or you can create your conda environments from scratch
  - conda create -n my-env MY-PACKAGES
- We also have pre-installed Jupyter kernels
Containerized DL: using Shifter on Perlmutter

To see images currently available:

```
shifterimg images | grep pytorch
```

To pull desired docker images onto Perlmutter:

```
shifterimg pull <dockerhub_image_tag>
```

To use interactively:

```
shifter --module gpu --image=nvcr.io/nvidia/pytorch:22.05-py3
```

Use Slurm image shifter options for best performance in batch jobs:

```
#SBATCH --image=nersc/pytorch:ngc-22.05_v1
srun shifter python my_python_script.py
```
Jupyter for Deep Learning

JupyterHub service provides a rich, interactive notebook ecosystem

- Very popular service with hundreds of users
- A favorite way for users to develop ML code

Users can run their deep learning workloads

- Using our pre-installed DL software kernels on dedicated Perlmutter GPU nodes
- **Using user custom kernels**

![Diagram showing popularity of code development platforms](image-url)
Hands-on Exercises
Compiling and Running Jobs on Perlmutter

- % ssh <user>@perlmutter.nersc.gov (or ssh <user>@saul.nersc.gov)
- % cd $SCRATCH
- % git clone https://github.com/NERSC/intro-NERSC-resources.git
- % cd intro-NERSC-resources

**CPU Examples:**
- 01-hello: build and run basic MPI program on CPU
- 02-matrix: build and run a hybrid MPI/OpenMP matrix multiply code on CPU
- 03-xthi: a hybrid MPI/OpenMP code, mainly on CPU affinity settings

**GPU Examples:**
- 04-pi_targ: build and run an OpenMP target offload program on GPU
- 05-gpus_for_tasks: build and run a CUDA code on GPU, and gpu affinity settings
Using Compute Node Reservations

- Existing NERSC users are added to “ntrain3” project
- Perlmutter node reservations available from 2-3:30 pm today
- User reservations with --reservation=xxx -A ntrain3, where
  - xxx is “intro_cpu” or “intro_gpu”
Thank You