

# Using Python with GPUs & Running Jobs on Perlmutter

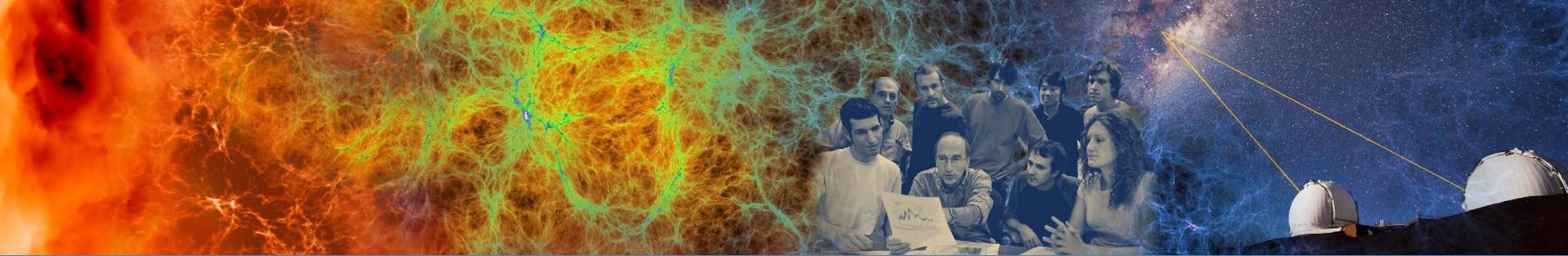


NERSC New User Training Summer 2024  
June 13, 2024 -Day 2

Charles Lively III\*  
Nick Tyler\*\*

\*User Engagement Group

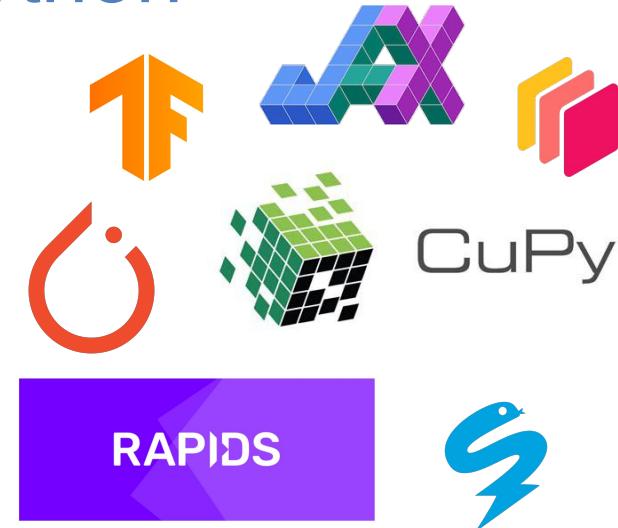
\*\*Programming Environments and Models Group  
National Energy Research Scientific Computing Center  
Lawrence Berkeley National Laboratory



# Using Python on GPUs

# 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, CUDA Python**
  - multi-gpu / multi-node / distributed memory:
    - **mpi4py+X, dask, cuNumeric**
- Many of these GPU libraries have adopted the [CUDA Array Interface](#) which makes it easier to pass array-like objects stored in GPU memory between the libraries
- There is also effort in the community to standardize around a common [Python array API](#)



```
numpy:      mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)
dask.array: mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)
cupy:       mean(a, axis=None, dtype=None, out=None, keepdims=False)
jax.numpy:   mean(a, axis=None, dtype=None, out=None, keepdims=False)
mxnet.np:   mean(a, axis=None, dtype=None, out=None, keepdims=False)
sparse:     s.mean(axis=None, keepdims=False, dtype=None, out=None)
torch:      mean(input, dim, keepdim=False, out=None)
tensorflow: reduce_mean(input_tensor, axis=None, keepdims=None, name=None,
                       reduction_indices=None, keep_dims=None)
```

# cudatoolkit dependency via module

```
> module load conda
```

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

```
> conda create --name cupy-demo python=3.11 numpy scipy
```

```
> conda activate cupy-demo
```

```
> pip install cupy-cuda11x
```

```
> python
```

```
>>> import cupy as cp
```

```
>>> print(cp.array([1, 2, 3]))
```

```
[1 2 3]
```

Check your package documentation to see  
cudatoolkit compatibility requirements

See documentation at <https://docs.nersc.gov/development/languages/python/using-python-perlmutter/>



# cudatoolkit dependency via conda

```
> module load conda
> module unload cudatoolkit
> conda create --name cupy-demo python=3.11 numpy scipy
> conda activate cupy-demo
> conda install -c conda-forge cupy
> python
>>> import cupy as cp
>>> print(cp.array([1, 2, 3]))
[1 2 3]
```

cupy conda-forge package will pull cudatoolkit dependencies into conda env

cupy conda-forge package will pull cudatoolkit dependencies into conda env

See documentation at <https://docs.nersc.gov/development/languages/python/using-python-perlmutter/>



# Is my code a good fit for a GPU?

CPUs → low latency  
GPUs → high throughput

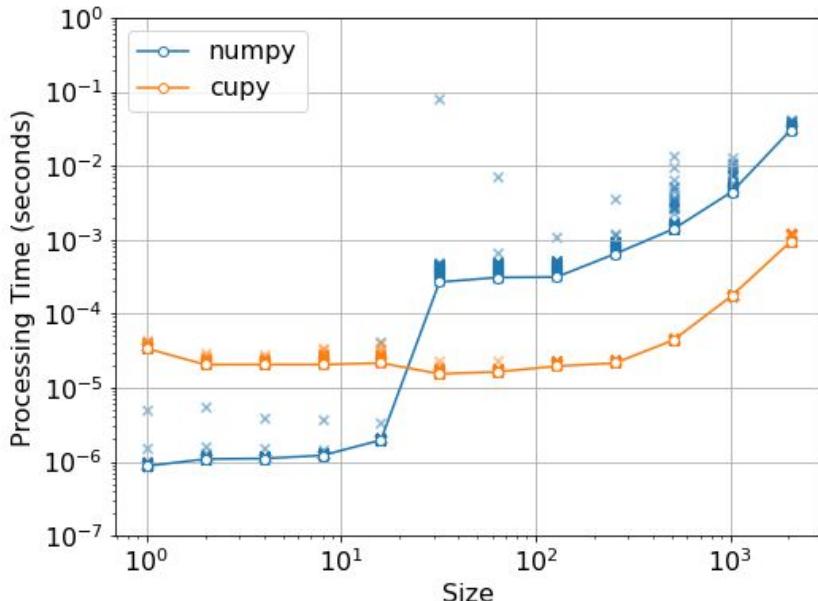
GPUs are likely a good fit if the following are true for your application:

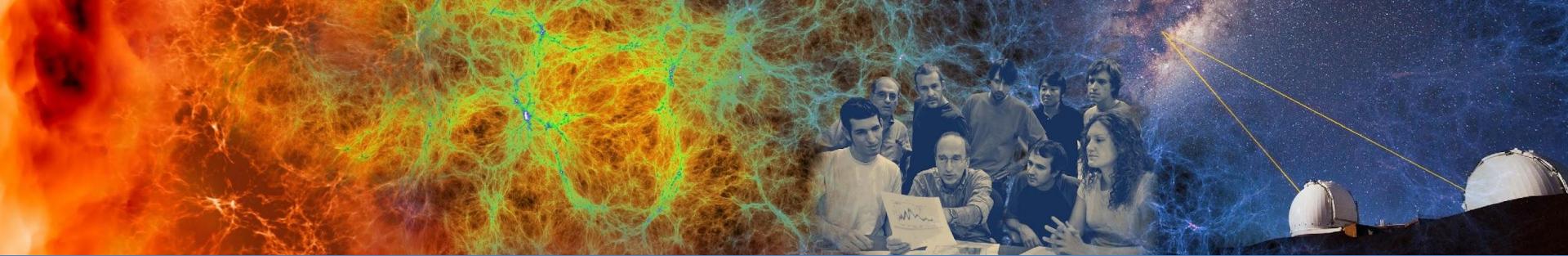
- Performs computation using large arrays, matrices, or images
- Dataset can fit in GPU memory
  - (40GB for Perlmutter's A100 GPUs)
- IO is not a bottleneck

For more help choosing a GPU-accelerated Python framework:

<https://docs.nersc.gov/development/languages/python/perlmutter-prep/>

```
a = xp.random.rand(size, size)
b = xp.random.rand(size, size)
def f(a, b):
    return xp.dot(a, b)
```





7



**BERKELEY LAB**  
Bringing Science Solutions to the World



**U.S. DEPARTMENT OF  
ENERGY** | Office of  
Science

# Best Practices & Where to get Python information

- Utilize Conda
- Check out Python in NERSC docs:
  - [Python at NERSC](#)
  - [Python on Perlmutter](#)
  - [Jupyter at NERSC](#)
  - Try the search bar at [docs.nersc.gov](#), it's pretty good!
- Can't find the answer? Submit a ticket at [help.nersc.gov](#)

A screenshot of a search results page. The search bar at the top contains the text "mpi4py". Below the search bar, it says "17 matching documents". The results are listed in a vertical scrollable list:

- How to use parallelism in Python**  
The [mpi4py](#) provides MPI standard bindings to the Python programming language. Documentation on [mpi4py](#) is available. Here is an example of how to use [mpi4py](#) on Cori: #!/usr/bin/env python from [mpi4py](#)...
- [5 more on this page](#)
- Guide to Using Python on Perlmutter**  
**mpi4py on Perlmutter**  
The most current release of [mpi4py](#) now includes CUDA-aware capabilities. If you intend to use [mpi4py](#) to transfer GPU objects, you will need CUDA-aware [mpi4py](#). The [mpi4py](#) you obtain via module load...
- [4 more on this page](#)
- Cori Large Memory software**  
**Using Python mpi4py**  
Using Python's [mpi4py](#) on the Large Memory nodes requires an [mpi4py](#) built with Open MPI. This means that the [mpi4py](#) in our default Python module will not work on these nodes. It also means that any custom conda...
- [1 more on this page](#)



# Summary

- Welcome to NERSC!
- We are here to help you use Python productively on Perlmutter
- If you have questions, please check our [docs.nersc.gov](https://docs.nersc.gov) or file a ticket at [help.nersc.gov](https://help.nersc.gov)

I has a question...



# Pipeline

- If you're brand new to HPC, Welcome!
  - What is a job?
  - How to run your code as a job?
- If you're just new to NERSC, Also Welcome!
  - Get to more advanced topics later
    - Running a job in container
    - Workflows
- Docs and Script Generator

<https://docs.nersc.gov>

The screenshot shows the NERSC Documentation homepage. At the top, there's a header with the NERSC logo and a search bar. Below the header is a navigation menu with links like Home, Getting Started, Tutorials, Accounts, Irs, Systems, Storage Systems, Connecting, Environment, Policies, Development, Developer Tools, Running Jobs, Applications, and a link to the National Energy Research Scientific Computing (NERSC) page. To the right of the menu, there's a section titled "Top documentation pages" with links to Getting Started, Getting Help, Job Queue Policy, Example Jobs, and Jobs overview.

NERSC Technical Documentation

The screenshot shows the "Jobscript Generator" tool on the My.NERSC website. On the left is a sidebar with links for Sign In, Dashboard, Jobs (Jobscript Generator, Completed Jobs), Perlmutter Queues, Queue Backlog, Center Status, File Browser, Service Tickets, Data Dashboard, and PI Toolbox. The main area has sections for "Job Information" (describing the tool as generating a batch script template), "Machine" (set to Perlmutter - GPU), "Application Name" (myapp.x), and "Job Name". A large text area on the right is labeled "Your script will be displayed here." and contains the generated jobscript.





# Basic Job Submission

# What is a Job? How do I get one?

- When you connect to Perlmutter you are on a login node
  - This includes Jupyter sessions
- Login nodes are **NOT** meant for large computing tasks!
  - They are shared by all users
  - Be kind to your fellow user
  - We only have 40 login nodes
- So where does my computation go?
  - On a compute node!
  - Perlmutter has 4864 compute nodes
    - 1792 GPU nodes, 3072 CPU nodes



# What is a Job? How do I get one?

- There are two ways to access a compute node
  - Interactive job
    - Directly connect to the compute node
    - Through a command line interface
    - Have a jupyter notebook on a compute node
  - Batch job
    - Place the work you want to do in a script
    - Submit the script to a queue
    - Wait for the work to be done

# How are jobs managed?

- Perlmutter uses Slurm workload manager
  - Slurm is an open source tool that performs job scheduling
- Slurm takes care of three key responsibilities
  - Allocating computer resources to jobs
  - Executes and monitors all jobs
  - Managing priorities of the jobs
- Even if you're familiar with Slurm it is configured differently per site



# How do I get a job from Slurm?

- Interactive
  - `salloc` - Slurm allocation
    - Gets an allocation on a node or set of nodes
  - At NERSC this defaults to running your login shell on a node in the allocation

```
clively@nid001465:/global/u1/c/clively> salloc -A m4388 -N 1 -t 10:00 -C gpu
salloc: Pending job allocation 25915811
salloc: job 25915811 queued and waiting for resources
salloc: job 25915811 has been allocated resources
salloc: Granted job allocation 25915811
salloc: Waiting for resource configuration
salloc: Nodes nid004053 are ready for job
clively@nid004053:/global/u1/c/clively>
```



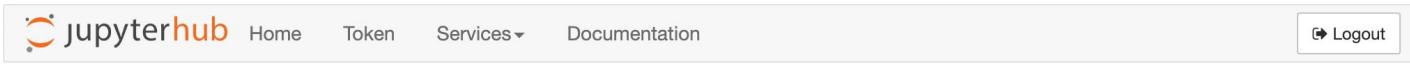
# What did I ask Slurm to do?

- `salloc -A m0000 -N 1 -t 10:00 -C gpu`
- `salloc`
  - Give me some compute nodes to use
- `-A m0000 | --account=m0000`
  - Charge to this NERSC account (usually starts with m)
- `-N 1 | --nodes=1`
  - Get 1 compute node to work on
- `-t 10:00 | --time=10:00`
  - Give me that node for 10 minutes
- `-C gpu | --constraint=gpu`
  - The type of node you want, either cpu or gpu



# How do I get a job from Slurm?

- Interactive allocations in Jupyter
  - These options can get you on a compute node
  - Come tomorrow to learn more about Jupyter!



Login Node		Shared GPU Node	Exclusive CPU Node	Exclusive GPU Node	Configurable Job
<b>Perlmutter</b>	<a href="#">start</a>	<a href="#">start</a>	<a href="#">start</a>	<a href="#">start</a>	<a href="#">start</a>
<i>Resources</i>	Use a login node shared with other users, outside the batch queues.	Use a single GPU on a node within a job allocation using defaults.	Use your own node within a job allocation using defaults.		Use multiple compute nodes with specialized settings.
<i>Use Cases</i>	Visualization and analytics that are not memory intensive and can run on just a few cores.	Work that fits on a single GPU, and uses at most a quarter of a GPU node's CPU cores and host memory.	Visualization, analytics, machine learning that is compute or memory intensive but can be done on a single node.		Multi-node analytics jobs, jobs in reservations, custom project charging, and more.



# When do I use an interactive job?

- Use interactive jobs to test and debug code
  - Also good option for profiling code
- Jobs in the interactive queue have limits
  - `-q interactive | --qos=interactive`
    - **1-4 nodes && 4 hours max walltime**
  - `-q shared_interactive | --qos=shared_interactive`
    - **$\frac{1}{2}$  node max && 4 hours max walltime**
      - 2 GPUs, 32 cores, 64 threads, ~120GB ram
      - 64 cores, 128 threads, ~250GB ram

# I need more time and nodes!

- Use a batch job
  - Submits the work you want to do into a queue
  - Lets Slurm schedule your work
    - Allows Slurm to give your job more time
    - Allows Slurm to schedule more compute nodes

```
clively@nid004053:/global/u1/c/clively> sbatch job_script.sh
Submitted batch job 25916143
clively@nid004053:/global/u1/c/clively>
```

# How do I submit a batch job?

- sbatch - Slurm Batch
  - Submit a batch script to Slurm
    - sbatch job\_script.sh
    - Slurm gives you back a job id

```
clively@nid004053:/global/u1/c/clively> sbatch job_script.sh
```

```
Submitted batch job 25916143
```

```
clively@nid004053:/global/u1/c/clively>
```

# What does script.sh look like?

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
srun -n $SLURM_NNODES hostname
```

- Similar options to salloc
- Add the special #SBATCH comment
- Slurm reads options from script
- Ask for 4 nodes for 8 hours
  - -J science | --job-name=science
  - Organize slurm outputs
    - %x - job name
    - %j - job id



# What does script.sh look like?

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
srun -n $SLURM_NNODES hostname
```

- Slurm adds environment variables to your job
  - Use the `$SLURM_NNODES` to get number of nodes requested
- Slurm run - `srun`
  - Run parallel jobs
    - Use this instead of `mpirun`
  - This will run one hostname per node



# Helpful Slurm environment variables

SLURM\_JOB\_NUM\_NODES # -N/--nodes=

SLURM\_NTASKS\_PER\_NODE # --ntasks-per-node=

SLURM\_CPUS\_ON\_NODE # Set by Slurm

SLURM\_GPUS\_ON\_NODE # Set by Slurm

Total CPUs: \$((SLURM\_JOB\_NUM\_NODES \* SLURM\_CPUS\_ON\_NODE))

Total Tasks: \$((SLURM\_JOB\_NUM\_NODES \* SLURM\_NTASKS\_PER\_NODE))

CPUs per Task: \$(((SLURM\_JOB\_NUM\_NODES \* SLURM\_CPUS\_ON\_NODE) / SLURM\_NTASKS))

Total GPUs: \$((SLURM\_JOB\_NUM\_NODES \* SLURM\_GPUS\_ON\_NODE))

GPUs per Task: \$(((SLURM\_JOB\_NUM\_NODES \* SLURM\_GPUS\_ON\_NODE) / SLURM\_NTASKS))

# What does the -q option do?

- Different queues with different limits
- `-q qdebug | --qos=debug`
  - 1-8 nodes && 30 minute max walltime
  - Test your script
  - Scaling before running larger jobs
- regular and shared
  - Where science gets done!
  - 24 hour max walltime, 5000 max job submissions
  - `-q regular | --qos=regular`
  - `-q shared | --qos=shared`
    - $\frac{1}{2}$  node max per job



# How do I debug my script?

- Override options in the script with CLI options
- Helpful for debugging or scaling tests
  - Use the debug queue
    - `sbatch -q debug -t 10 script.sh`
  - Scale testing
    - `sbatch -N 2 script.sh`
    - `sbatch -N 20 script.sh`

# How do I see if my jobs working?

- `squeue` - Slurm queue
  - view information about jobs in the Slurm queue
  - Returns information from all jobs
    - Can be a lot on a big system like Perlmutter
- `sqs`
  - NERSC shortcut with some helpful output options
- Shows job state `R` - Running, `PD` - Pending
- `TIME` - How long the job has been running

clively@nid001465:/global/u1/c/clively> sqs											
JOBID	ST	USER	NAME	NODES	TIME_LIMIT	TIME	SUBMIT_TIME	QOS	START_TIME	FEATURES	NODELIST(REASON)
24125002	PD	clively	/global/cfs/	1	10:00	0:00	2024-05-22T03:31:22	cron	2024-05-29T03:31:00	cron	(BeginTime)
24125000	PD	clively	/global/cfs/	1	10:00	0:00	2024-05-22T02:12:08	cron	2024-05-23T02:11:00	cron	(BeginTime)
24125001	PD	clively	/global/cfs/	1	10:00	0:00	2024-05-22T00:21:49	cron	2024-05-23T00:21:00	cron	(BeginTime)
25915811	CG	clively	interactive	1	10:00	10:17	2024-05-22T07:29:14	gpu_debug	2024-05-22T07:30:26	gpu&a100	nid004053
25915593	R	clively	interactive	1	1:00:00	15:47	2024-05-22T07:25:48	gpu_interactive	2024-05-22T07:26:16	gpu&a100	nid001465



# How do I end a job?

- `scancel` - Slurm cancel
  - Send stop signal to jobs or job steps managed by Slurm
    - Stop job running too long or with the wrong parameters
    - Conserve your NERSC hours if you made a mistake!

```
clively@login22:/global/u1/c/clively> sbatch -q debug -t 20 job_script.sh
Submitted batch job 26735450
clively@login22:/global/u1/c/clively> sqs
      JOBID      ST USER      NAME      NODES TIME_LIMIT      TIME      SUBMIT_TIME      QOS      START_TIME
      FEATURES      NODELIST(REASON
24125002      PD clively /global/cfs/ 1      10:00      0:00 2024-06-12T03:31:37 cron      2024-06-19T03:31:00
      cron      (BeginTime)
24125000      PD clively /global/cfs/ 1      10:00      0:00 2024-06-12T02:11:56 cron      2024-06-13T02:11:00
      cron      (BeginTime)
24125001      PD clively /global/cfs/ 1      10:00      0:00 2024-06-12T00:21:11 cron      2024-06-13T00:21:00
      cron      (BeginTime)
26735450      PD clively job_script.s 1      20:00      0:00 2024-06-12T20:48:40 gpu_debug      N/A
      gpu&a100      (Resources)
clively@login22:/global/u1/c/clively> scancel 26735450
clively@login22:/global/u1/c/clively> sqs
```



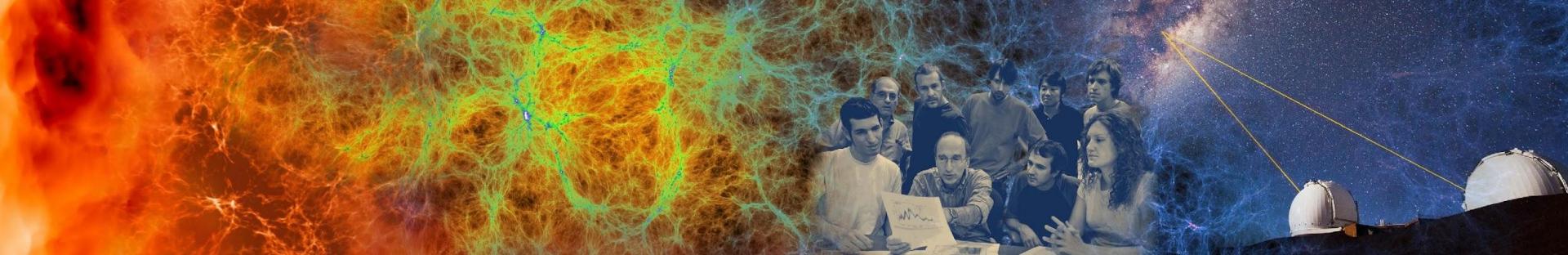
# How to look at completed jobs?

- `sacct` - Slurm accounting
  - Accounting data for all jobs and job steps in the Slurm job accounting log or Slurm database
  - By default shows jobs completed in the last day

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
24125002	/global/c+	cron	nstaff	2	REQUEUED	126:0
24125002.ba+	batch		nstaff	2	FAILED	126:0
24125002.ex+	extern		nstaff	2	COMPLETED	0:0
24125001	/global/c+	cron	nstaff	2	REQUEUED	0:0
24125001.ba+	batch		nstaff	2	COMPLETED	0:0
24125001.ex+	extern		nstaff	2	COMPLETED	0:0
24125000	/global/c+	cron	nstaff	2	REQUEUED	0:0
24125000.ba+	batch		nstaff	2	COMPLETED	0:0
24125000.ex+	extern		nstaff	2	COMPLETED	0:0
26705767	fsp	regular_m+	nstaff	256	FAILED	1:0
26705767.ba+	batch		nstaff	256	FAILED	1:0
26705767.ex+	extern		nstaff	256	COMPLETED	0:0
26734859	job_script	gpu_ss11	m4388_g	128	COMPLETED	0:0
26734859.ba+	batch		m4388_g	128	COMPLETED	0:0
26734859.ex+	extern		m4388_g	128	COMPLETED	0:0
26734859.0	hello_mpi		m4388_g	2	COMPLETED	0:0
26734988	job_script	gpu_ss11	m4388_g	0	CANCELLED+	0:0

# How to look at completed jobs?

- `sacct -j jobid`
  - Shows information about one jobid
- `sacct --name science --constraint gpu`
  - Search through jobs by other attributes
- `sacct --state running`
  - Select jobs based on their current state



# Jobs in containers

# Running jobs in containers

- Containers are a great
  - Make your software portable between systems
  - Decrease start time of large jobs
    - python
- NERSC Supports two container technologies
  - Shifter
  - podman-hpc - **New**
    - Can build images on login nodes!
- We don't support Singularity/Apptainer on Perlmutter



podman

# What is a container?

- A way to pack up all your software
- Docker is just one technology
- On your personal computer
  - Build
    - docker build ...
  - Ship
    - docker push ...
  - Run
    - docker run ...



```
#Dockerfile
FROM ubuntu:latest

RUN apt-get update &&
    apt-get install -y \
        cmake python3-pip

RUN pip install pandas

COPY code /mycode
WORKDIR /mycode
RUN cmake --build .
```

# Where do I ship it?

- NERSC has a registry
  - `registry.nersc.gov`
  - Build
    - `docker build -t registry.nersc.gov/m0000/test:v1.0 .`
  - Ship
    - `docker login registry.nersc.gov`
    - `docker push registry.nersc.gov/m0000/test:v1.0`
  - Run with Shifter or Podman-HPC



# How do I run a Shifter container?

- Pull your image before you start your job
  - shifterimg pull registry/image:tag

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --image=registry/image:tag
```



SHIFTER

```
srun -n $SLURM_NNODES shifter hostname
```



# How do I run a Shifter container?

- Extra options for shifter
  - --volume=/pscratch/sd/u/user:/scratch
  - --env=MYENV=1234
  - --clearenv
  - --workdir=/work
  - --module=...
    - none
    - mpich
    - cvmfs
    - gpu
    - cuda-mpich
    - nccl-2.15
    - network

# How do I run a podman-hpc container?

- Pull your image before you start your job
  - `podman-hpc pull registry/image:tag`

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
srun -n $SLURM_NNODES \
    podman-hpc run registry/image:tag hostname
```



# How do I run a podman-hpc container?

- Pull, Or build images on login nodes, then migrate to scratch
  - `podman-hpc build -t image_name:tag .`
  - `podman-hpc migrate image_name:tag`
- Docker/Podman options work
  - `--volume=/pscratch/sd/u/user:/scratch`
  - `--net host`
- Extra options similar to shifter modules
  - `--mpi`
  - `--gpu`
  - `--cuda-mpi`



# Multiple jobs and Workflows

# I have multiple things I need to do

- Bundling jobs with slurm
  - Run multiple executables sequentially or simultaneously
- Use a Slurm job array
  - Same job task with different inputs
- Workflow tools
  - GNU Parallel
    - Many small tasks, fit onto one node
  - More complex tasks
    - Parsl, Fireworks, etc.

# Bundling work into one job

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
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
```

- Bundling jobs with slurm
  - Programs run **sequentially**
  - Only have to wait for scheduler once
    - Reuse the same allocated nodes for different steps in your workflow



# Bundling work into one job

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
srun -N 1 -n 256 ./a.out input0 &
srun -N 1 -n 256 ./a.out input1 &
srun -N 1 -n 256 ./a.out input3 &
srun -N 1 -n 256 ./a.out input4 &
wait
```

- Bundling jobs with slurm
  - Programs run **simultaneously**
  - Only have to wait for scheduler once
    - This example runs same program with different inputs per srun



# Using Job Arrays

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 1
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
#SBATCH --array=1-4
```

```
echo $SLURM_ARRAY_JOB_ID
```

```
srun -n 256 ./a.out $SLURM_ARRAY_JOB_ID
```

- Slurm manages each job independently
  - If one task fails it won't affect others
- Good option for getting
  - Large statistics on same inputs
  - Parameter sweep over input files

# Using GNU Parallel

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 1
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
module load parallel
```

```
parallel -j256 ./a.out {} :::: inputs*
```

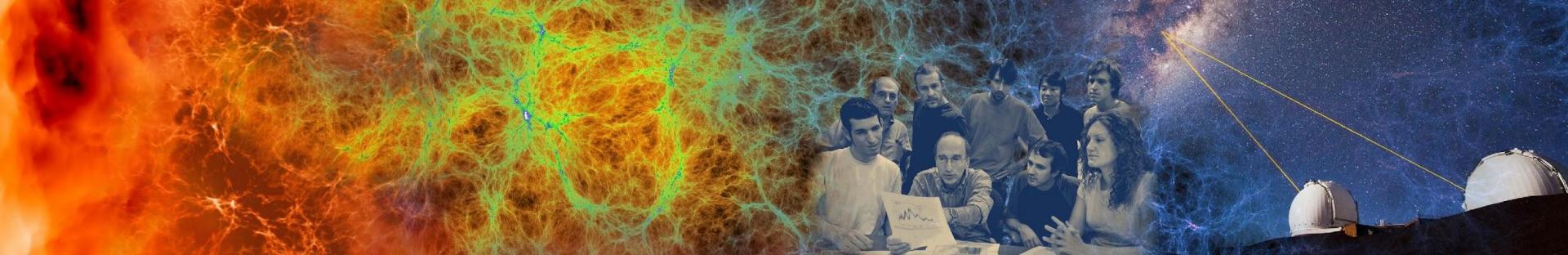
- You manage tasks inside of allocation
  - Great for many small tasks
    - Faster start times than `sruns`
  - Reuse allocation for all your tasks
- As tasks finish the next one starts
  - Use allocation efficiently



# More complex workflows with dependencies

- Use a workflow management system
  - Parsl/FuncX/Globus Compute
  - Fireworks
  - Many more...
- Write code to define workflow
- Often written in python
- Handle dependencies between different types of tasks
- [github.com/CrossFacilityWorkflows/DOE-HPC-workflow-training](https://github.com/CrossFacilityWorkflows/DOE-HPC-workflow-training)
  - Resources from previous training with ALCF and OLCF
- Reach out at `help.nersc.gov` with more questions





# Best Practices

# Jobs Scheduling

- Each job has a priority value
  - Grouped by user, QOS, and account
  - Only two jobs per these groupings gain priority at a time
    - More jobs can run, only two will age
- Main scheduler uses priority list
  - Schedules a few days in the future
- Backfill scheduler puts shorter jobs in “holes”
  - Prioritize utilization

# Jobs Scheduling Tips

- One job with a large allocation
  - Per node priority ageing is the highest
  - Can get scheduled first
- Shorter time length jobs
  - Easier to schedule as backfill
  - Use a workflow manager
- Choose the right time from Slurm
  - Balance between enough runtime
  - Waiting in the queue for a long job

# Job script generator: More advanced threading options

## Jobscript Generator

**Job Information**

This tool generates a batch script template which also realizes specific process and thread binding configurations.

**Machine**  
Select the machine on which you want to submit your job.

**Application Name**  
Specify your application including the full path.

**Job Name**  
Specify a name for your job.

**Email Address**

```
#!/bin/bash
#SBATCH -N 128
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J Science
#SBATCH -t 00:30:00

#OpenMP settings:
export OMP_NUM_THREADS=64
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

#run the application:
srun -n 512 -c 64 --cpu_bind=cores myapp.x
```



# Options for OpenMP Code

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 1
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
export OMP_NUM_THREADS=8
export OMP_PLACES=cores
export OMP_PROC_BIND=spread
```

```
srun -n 256 ./a.out $SLURM_ARRAY_JOB_ID
```

- OpenMP
  - config through env variables
- Some libraries use OpenMP by default
  - BLAS/LAPACK
  - numpy in python
    - Small numpy arrays can be faster with less threads

# Options for MPI codes

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 2
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

- Settings to Address NUMA Performance
  - Use `--cpu_bind=cores` when
    - $\#MPI\ tasks \leq \#cores$
  - Use `--cpu_bind=threads` when
    - $\#MPI\ tasks > \#cores$

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

# Options for Hybrid OpenMP/MPI codes

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 2
#SBATCH -t 8:00:00
#SBATCH -C cpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err
```

```
export OMP_NUM_THREADS=8
export OMP_PLACES=cores
export OMP_PROC_BIND=spread
```

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

- Hybrid MPI/OpenMP code
  - Number of cores per task – c
  - $-c \geq \text{OMP\_NUM\_THREADS}$
  - Give enough cpus to be able to use OpenMP threads efficiently

# Options for gpu codes

```
#!/bin/bash
#SBATCH -A m0000
#SBATCH -q regular
#SBATCH -N 2
#SBATCH -t 8:00:00
#SBATCH -C gpu
#SBATCH -J science
#SBATCH -o %x_%j.out
#SBATCH -e %x_%j.err

export OMP_NUM_THREADS=8
export OMP_PLACES=cores
export OMP_PROC_BIND=spread
```

```
srun -n 8 -c 8 --gpus-per-task=1 \
--cpu_bind=cores ./a.out
```

- GPU codes
  - Can specify the number of gpus per task
    - --gpus-per-task=n
- More advanced
  - Specific gpu mapping
    - --gpu-bind



# What did we cover?

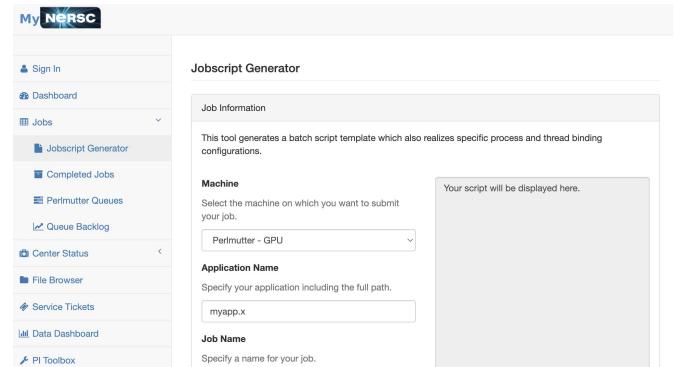
- What is a job?
- How to run your code as a job?
- Running a job in container
- Workflows
- Docs and Script Generator

<https://docs.nersc.gov>



The screenshot shows the NERSC Documentation homepage. The top navigation bar includes the NERSC logo, a search bar, and a "NERSC Documentation" link. The main content area has two columns. The left column is titled "NERSC Documentation Home" and lists various documentation categories: Getting Started, Tutorials, Accounts, Iris, Systems, Storage Systems, Connecting, Environment, Policies, Development, Developer Tools, Running Jobs, and Applications. Each category has a corresponding expandable arrow. The right column is titled "NERSC Technical Documentation" and contains a section for "Top documentation pages" with links to "Getting Started", "Getting Help", "Job Queue Policy", "Example Jobs", and "Jobs overview". Below this is a note about NERSC providing HPC and Storage facilities for research sponsored by the U.S. Department of Energy.

[https://my.nersc.gov/script\\_generator.php](https://my.nersc.gov/script_generator.php)



The screenshot shows the "Jobscript Generator" tool within the "My NERSC" interface. On the left is a sidebar with links for Sign In, Dashboard, Jobs (which is expanded to show Jobscript Generator, Completed Jobs, Perllmutter Queues, Queue Backlog, Center Status, File Browser, Service Tickets, Data Dashboard, and PI Toolbox), Application Name, and Job Name. The main panel is titled "Jobscript Generator" and contains sections for "Job Information" (describing the tool as generating a batch script template), "Machine" (set to "Perlmutter - GPU"), "Application Name" (set to "myapp.x"), and "Job Name" (set to "myjob"). A large text area on the right is labeled "Your script will be displayed here." and is currently empty.



Thank You for  
listening and  
Welcome to  
NERSC!

