Using Python and Jupyter on Perlmutter

Perlmutter New User Training, 2022

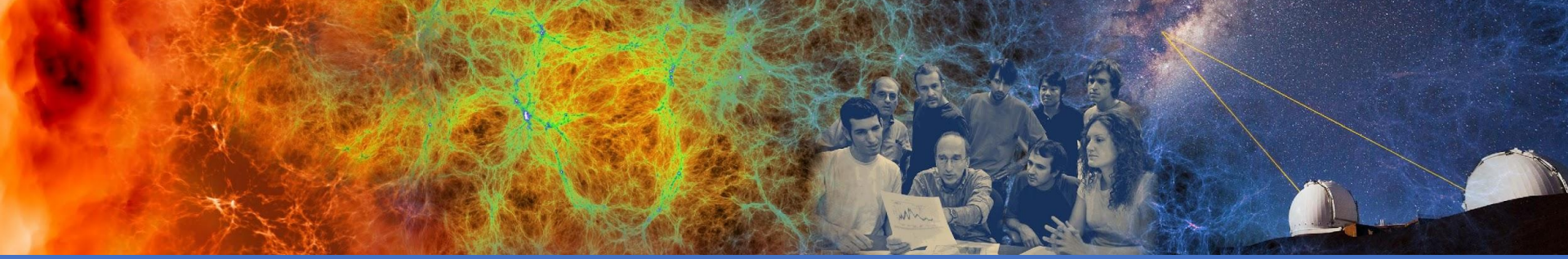
L. Stephey, D. Margala, R. Thomas

Jan 7, 2022

Python/Jupyter users, welcome to Perlmutter!

What we'll cover in this short 30 mins

- General Python advice
- Using Python on GPUs
- Using Jupyter on Perlmutter
- Open Q&A (5 mins at end)



Python on Perlmutter

What should I know moving from Cori to Perlmutter?

- More unified Python module + Jupyter configuration 👋
  - Jupyter Python 3 kernel is based on Python default module, currently `python/3.9-anaconda-2021.11`
  - Python and Jupyter Python 3 Kernel share `$PYTHONUSERBASE`, which determines location and search path of pip installed packages
- Many “best practices” remain
  - Use our `/global/common/software/<your project>` filesystem for better performance
  - Use custom conda environments for customizable Python sandboxes, can be easily converted into a Jupyter kernel
  - Use a Shifter container, can be easily converted into a Jupyter kernel
- 6000+ GPUs!

More systems, more problems! How to avoid common problems:

- Cori and Perlmutter are two very different systems that are **still sharing filesystems**, including `$HOME` and dotfiles (`.bashrc`, `.bash_profile`, etc.)
- Don't put system-specific modifications in your dotfiles
  - Actually, put as little in your dotfiles as you can and make sure to periodically review their contents
- Most likely code/environments you have built for Cori will not work on Perlmutter, and vice-versa. **mpi4py will not work across systems**.
- Tip– append system name to your custom conda environments, like dask-cori and dask-pm to help you keep track
- Be more careful using pip (we'll cover this)

Use environments with conda activate

- You can now use **`conda activate`** on Perlmutter without using **`conda init`** and it will not make changes to your **`.bashrc`**
- This is already possible on Perlmutter and will change on Cori at the AY rollover on Jan 19

Old 😓

```
module load python
source activate myenv
conda deactivate
```

or

```
module load python
conda init
conda activate myenv
conda deactivate
```

New 👌

```
module load python
conda activate myenv
conda deactivate
```

The current default **`python`** module is **`python/3.9-anaconda-2021.11`**

Check out our pending updates to the Python docs for more information

Building and using mpi4py

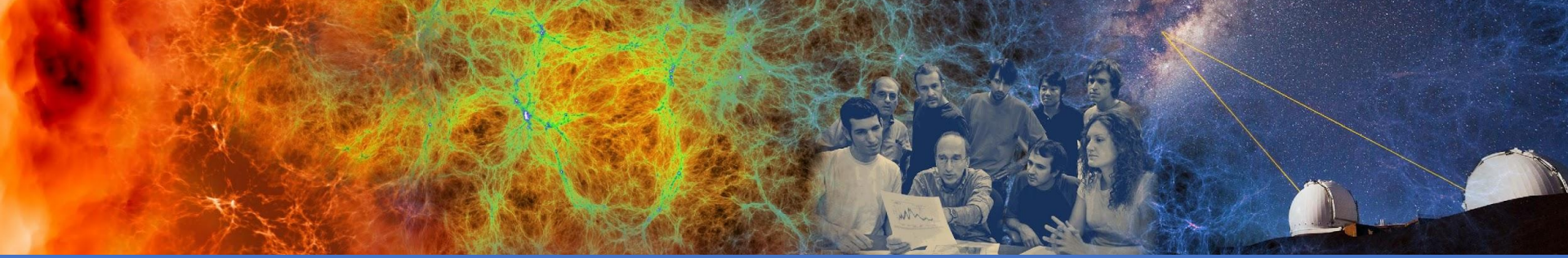
- mpi4py is available via `module load python`
- This mpi4py is CUDA-aware (can communicate GPU objects)
- To build your own CUDA-aware mpi4py, follow this recipe:

```
module load PrgEnv-gnu cudatoolkit python
conda create -n cudaaware python=3.9 -y
conda activate cudaaware
MPICC="cc -target-accel=nvidia80 -shared" pip
install --force --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

Use pip with caution ⚠️

- Be careful with pip!!!! pip will try to be clever and find existing packages to save time, but most likely you don't want this
- Best practices for pip
  - Use it inside of a conda environment, not outside (don't use `--user`)
  - Always pip install `<package> --force` (Did you notice this in our mpi4py recipe?)
  - This will force a rebuild, which is important if tries to reuse a package from Cori
- If you use `pip --user`, it will install packages to the location specified by `PYTHONUSERBASE`, which is by default `$HOME/.local/perlmutter/3.9-anaconda-2021.11`
- It is safe to delete this directory if you want to clean up/save space



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**, **cuNumeric** (coming soon?)
  - "machine learning" libraries that also support general GPU computing
    - **PyTorch**, **TensorFlow**
  - "I want to write my own GPU kernels"
    - **Numba**, **PyOpenCL**, **PyCUDA**
- Many of these GPU libraries have adopted the CUDA Array Interface which makes it easier to share array-like objects stored in GPU memory between the libraries
- There is also some 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)
```

Getting started with GPUs in Python (CuPy)

Numpy features support by CuPy:

- Basic indexing (indexing by ints, slices, newaxes, and Ellipsis)
- Most of Advanced indexing (except for some indexing patterns with boolean masks)
- Data types (dtypes): bool_, int8, int16, int32, int64, uint8, uint16, uint32, uint64, float16, float32, float64, complex64, complex128
- Most of the array creation routines (empty, ones_like, diag, etc.)
- Most of the array manipulation routines (reshape, rollaxis, concatenate, etc.)
- All operators with broadcasting
- All universal functions for elementwise operations (except those for complex numbers)
- Linear algebra functions, including product (dot, matmul, etc.) and decomposition (cholesky, svd, etc.), accelerated by cuBLAS and cuSOLVER
- Multi-dimensional fast Fourier transform (FFT), accelerated by cuFFT
- Reduction along axes (sum, max, argmax, etc.)

Getting started with GPUs in Python (CuPy)

```
> ssh perlmutter
> module load cudatoolkit python
> conda create -y --name cupy-demo python=3.9 numpy scipy
> source activate cupy-demo
> pip install cupy-cuda114
> python
>>> import cupy as cp
>>> print(cp.array([1, 2, 3]))
[1 2 3]
```

Current default version is: cudatoolkit/21.9_**11.4**

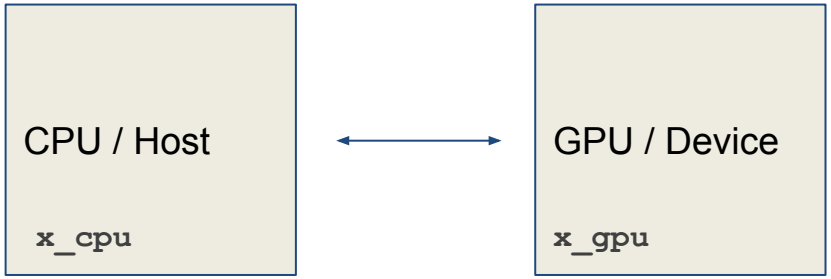
Version should match the CUDA version from the cudatoolkit module

See documentation at https://docs.nersc.gov/development/languages/python/using-python-perlmutter/ or open a ticket at https://help.nersc.gov/

Getting started with GPUs in Python (CuPy)

```
>>> import numpy as np
>>> import cupy as cp


# Create an array on GPU/device
>>> x_gpu = cp.array([1, 2, 3])
>>> isinstance(x_gpu, cp.ndarray)
True
```



In general, it is a good idea to minimize data movement between Host and Device

```
# Data Transfer
>>> x_cpu = np.array([1, 2, 3]) # create an array on CPU/host
>>> x_gpu = cp.asarray(x_cpu)  # move the data to the GPU/device


>>> x_gpu = cp.array([1, 2, 3])  # create an array in the GPU/device
>>> x_cpu = cp.asnumpy(x_gpu)  # move the array to the CPU/host
```

Advanced GPU programming in Python

```python
import cupy
import numba.cuda
import numpy

# CUDA kernel
@numba.cuda.jit
def _cuda_addone(x):
    i = numba.cuda.grid(1)
    if i < x.size:
        x[i] += 1

# convenience wrapper with thread/block configuration
def addone(x):
    # threads per block
    tpb = 32
    # blocks per grid
    bpg = (x.size + (tpb - 1)) // tpb
    _cuda_addone[bpg, tpb](x)
```

```python
# create array on device using cupy
x = cupy.zeros(1000)

# pass cupy ndarray to numba.cuda kernel
addone(x)

# Use numpy api with cupy ndarray
total = numpy.sum(x)
```

https://docs.cupy.dev/en/stable/user_guide/basic.html
https://numba.readthedocs.io/en/stable/cuda/index.html

- NumPy's **__array_function__** protocol (NEP 18)
  - `numpy.sum(x)` -> `cupy.sum(x)`
- CPU and GPU execution paths can share same implementation (sometimes)
- Can also use helper functions to get the appropriate array module. For example:
  - `xp = cupy.get_array_module(x)`

Profiling using NVIDIA Nsight Systems

```
import cupy as cp

cp.cuda.nvtx.RangePush(message)
…
cp.cuda.nvtx.RangePop()

@cp.prof.TimeRangeDecorator(message)
def function():
    pass

with cp.prof.time_range(message):
    pass
```

CuPy supports for NVIDIA Tools Extension (NVTX) markers and ranges

Or use decorator syntax without modifying function body

Can also use `with`-statement context blocks

Run your application with Nsight Systems:

```
> nsys profile --trace cuda,nvtx --stats=true python myapp.py
```

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

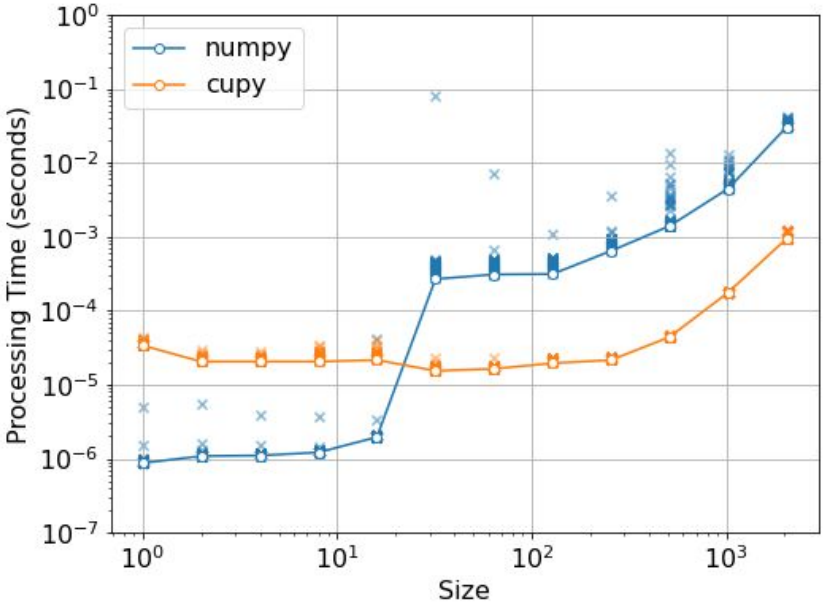
For more help choosing a GPU-accelerated Python framework:

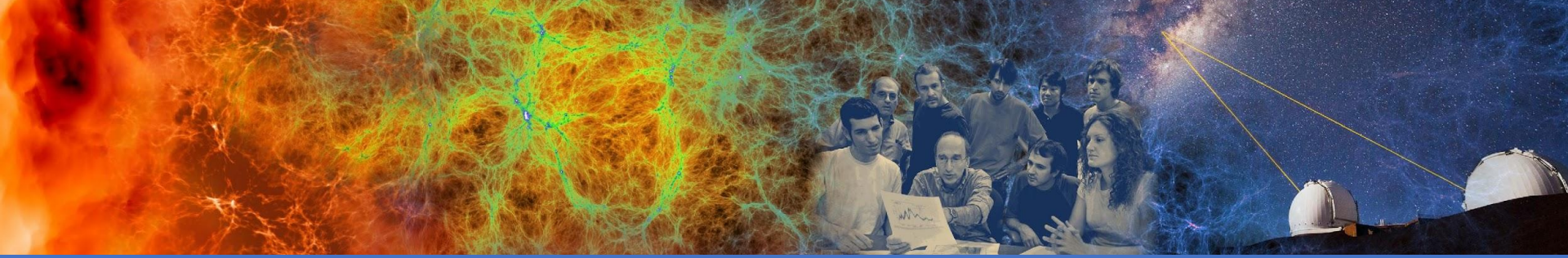
https://docs.nersc.gov/development/languages/python/perlmutter-prep/

or open a ticket at https://help.nersc.gov/

```
a = xp.random.rand(size, size)
b = xp.random.rand(size, size)

def f(a, b):
    return xp.dot(a, b)
```





Jupyter on Perlmutter

How do I run Jupyter notebooks on Perlmutter Phase I?

Making sure you can access Perlmutter with Jupyter
Available configurations and what they are for

How do I make use of Perlmutter GPUs from Jupyter?

Hardware you can use for each configuration
Monitoring GPU usage in JupyterLab

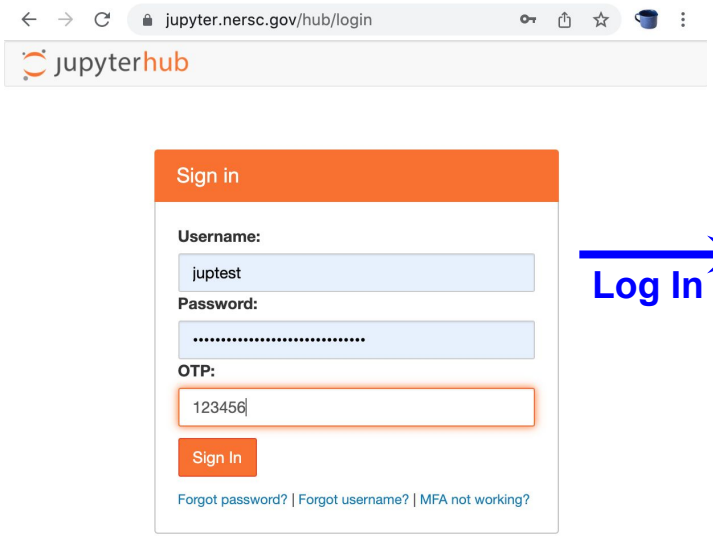
What should I know moving from Cori to Perlmutter?

What differences in the deployment matter to you

How do I run Jupyter notebooks on Perlmutter Phase I?

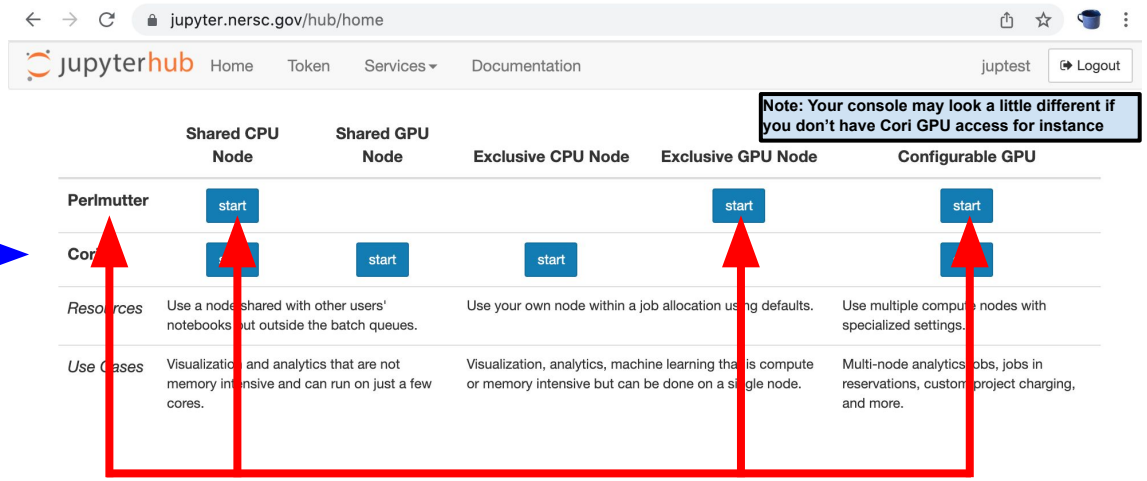
Making sure you can access Perlmutter with Jupyter

https://jupyter.nersc.gov



Log In

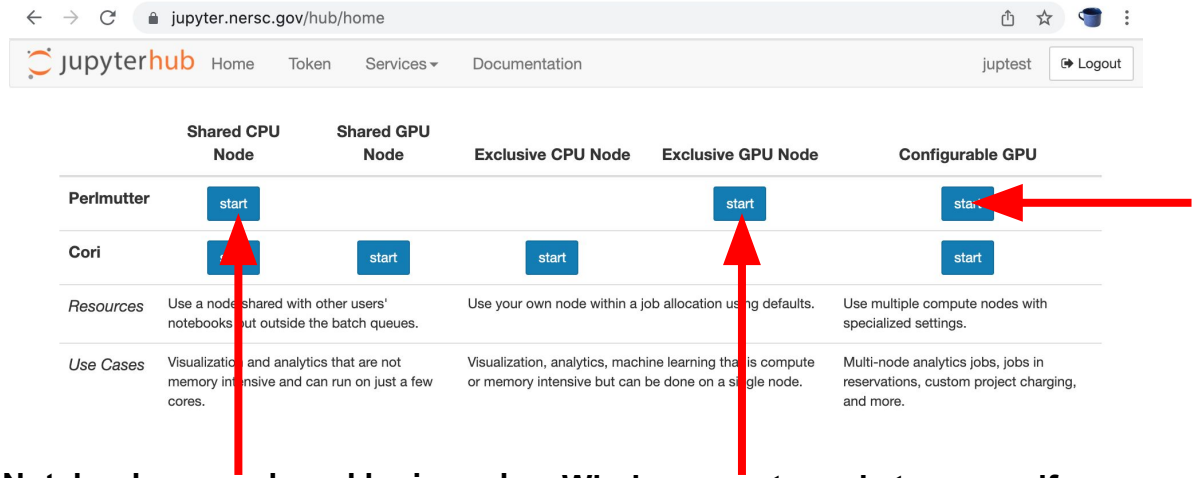
https://jupyter.nersc.gov/hub/home (home or "console")

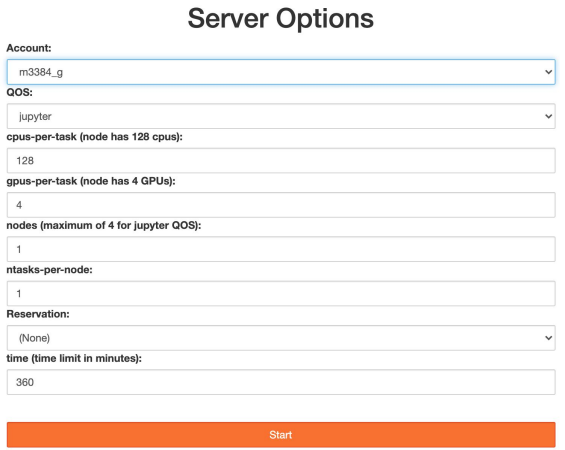


If this row does not show up, you need Perlmutter to be added to your list of "server logins" in Iris.

How do I run Jupyter notebooks on Perlmutter Phase I?

Available configurations and what they are for





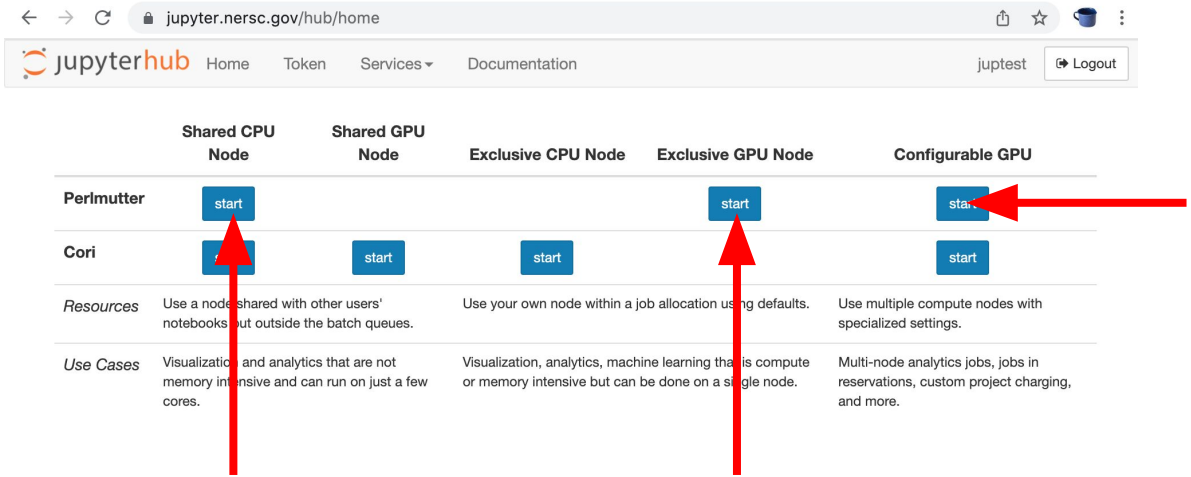
Notebooks on a shared login node
Will not be charged
No CPU/GPU/memory limits yet
Debug/test/develop
Not as compute intensive

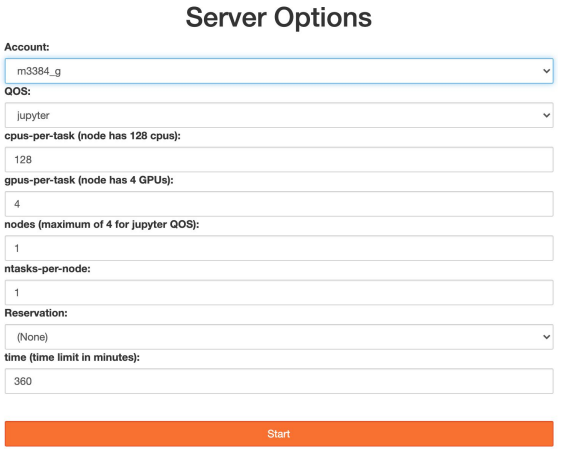
Whole compute node to yourself
Will be charged to sensible default
6 hour time limit
Interactive GPU work
Compute intensive

Up to 4 nodes, up to 6 hours
Customize your Slurm allocation
Interactive GPU work
You need to scale, baby!
… need more? Contact me!

How do I make use of Perlmutter GPUs from Jupyter?

Hardware you can use for each configuration





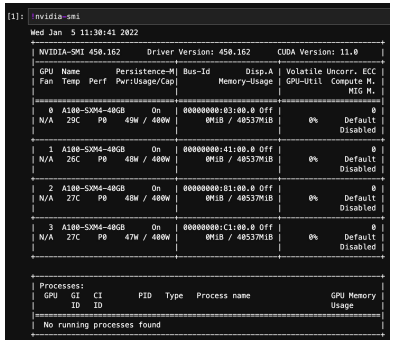
Login Node
1 "shared" NVIDIA A100
Kind of free-for-all for now, future:
NVIDIA Multi Instance GPU (MIG)?

1 Compute Node
4 NVIDIA A100's

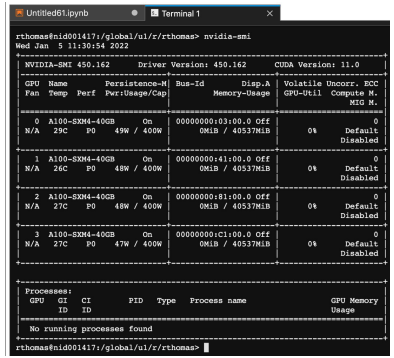
4 Compute Nodes
16 NVIDIA A100's
Limits may change per demand

How do I make use of Perlmutter GPUs from Jupyter?

Notebook: `!nvidia-smi`

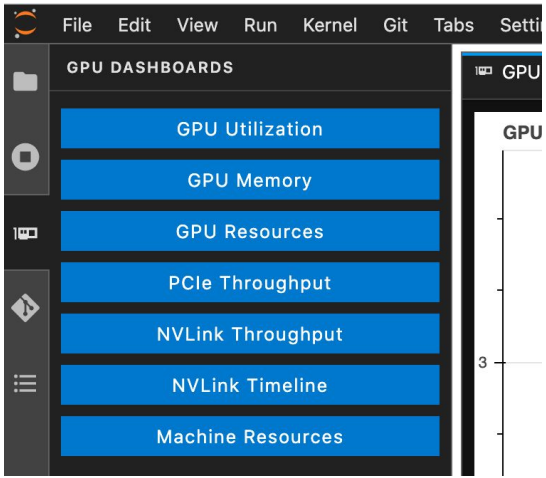


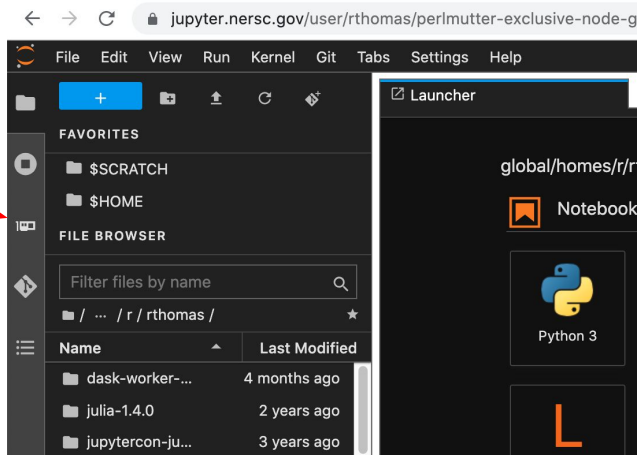
Terminal: `nvidia-smi`



Monitoring GPU usage in JupyterLab

NVDashboard LabExtension



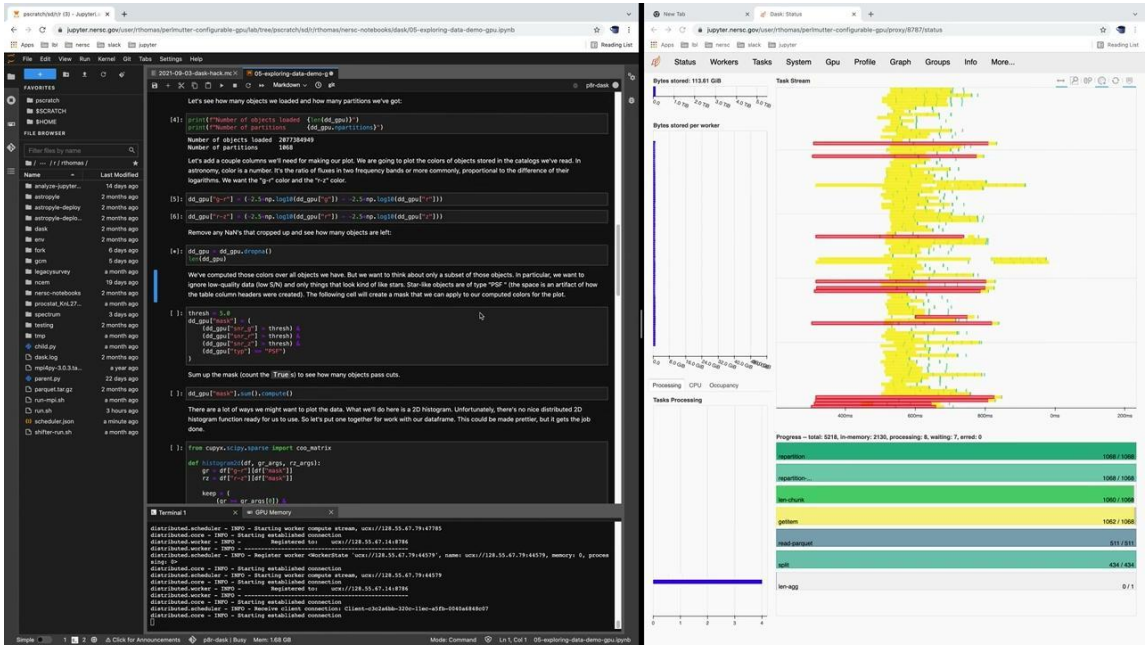


Monitor GPUs on the node (Compute or login node)

For multi-node jobs, only GPUs on the same node as the notebook

How do I make use of Perlmutter GPUs from Jupyter?

Monitoring GPU usage in JupyterLab



Like Dask?
Want to try Dask?

Dask Dashboard

Monitor everything
Watch progress
Profile your workflow
Separate tab for now
(Demo on 100 GPUs)

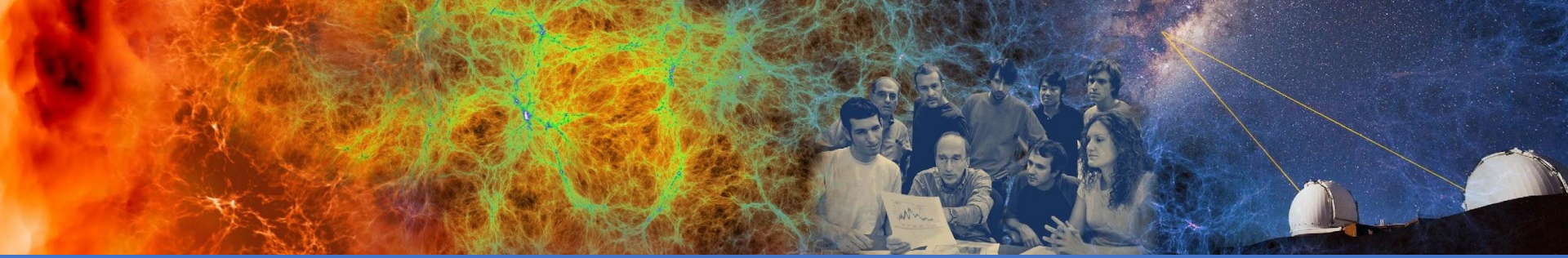
Example notebook with dashboard setup instructions:
https://gitlab.com/NERSC/nersc-notebooks/-/tree/master/dask

dask-labextension needed a “fix” for us to deploy
Just merged so it’s coming soon…!

What should I know moving from Cori to Perlmutter?

What differences in the deployment matter to you

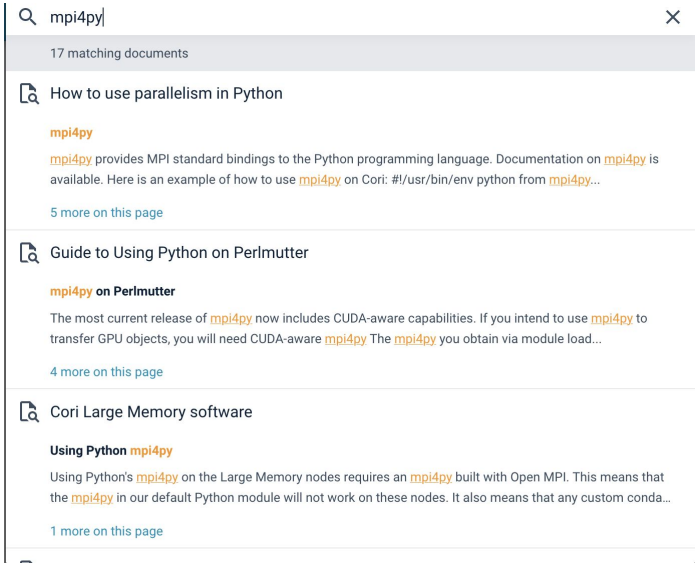
	Cori	Perlmutter
Shared Node Notebooks	24 login nodes total 4 dedicated to Jupyter No other usages allowed CPU/memory limits in place	40 login nodes total Jupyter *not restricted to a subset* Runs alongside ssh-based logins No resource limits in place yet
Exclusive Node Notebooks	Limited resources available Requires access to special QOS Kind of fussy setup	Jupyter jobs are first class No plans for special QOS Watching allocation success rate Will make adjustments to queues
Configurable Notebooks	Limited to GPU partition	
Setting up Kernels	Conda envs, custom, helpers: See NERSC Jupyter docs	Same deal as on Cori Your conda envs may be bigger!



Wrap Up

Where to get Python/Jupyter information, help

- Have a question? Try our documentation (updated almost daily!)
  - Using Perlmutter
  - 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



Summary

- Welcome to Perlmutter!
- We are here to help you use Python and Jupyter productively on Perlmutter
- If you have questions, please check our docs.nersc.gov or file a ticket at help.nersc.gov
- Don’t be shy– now is the time to ask us questions!