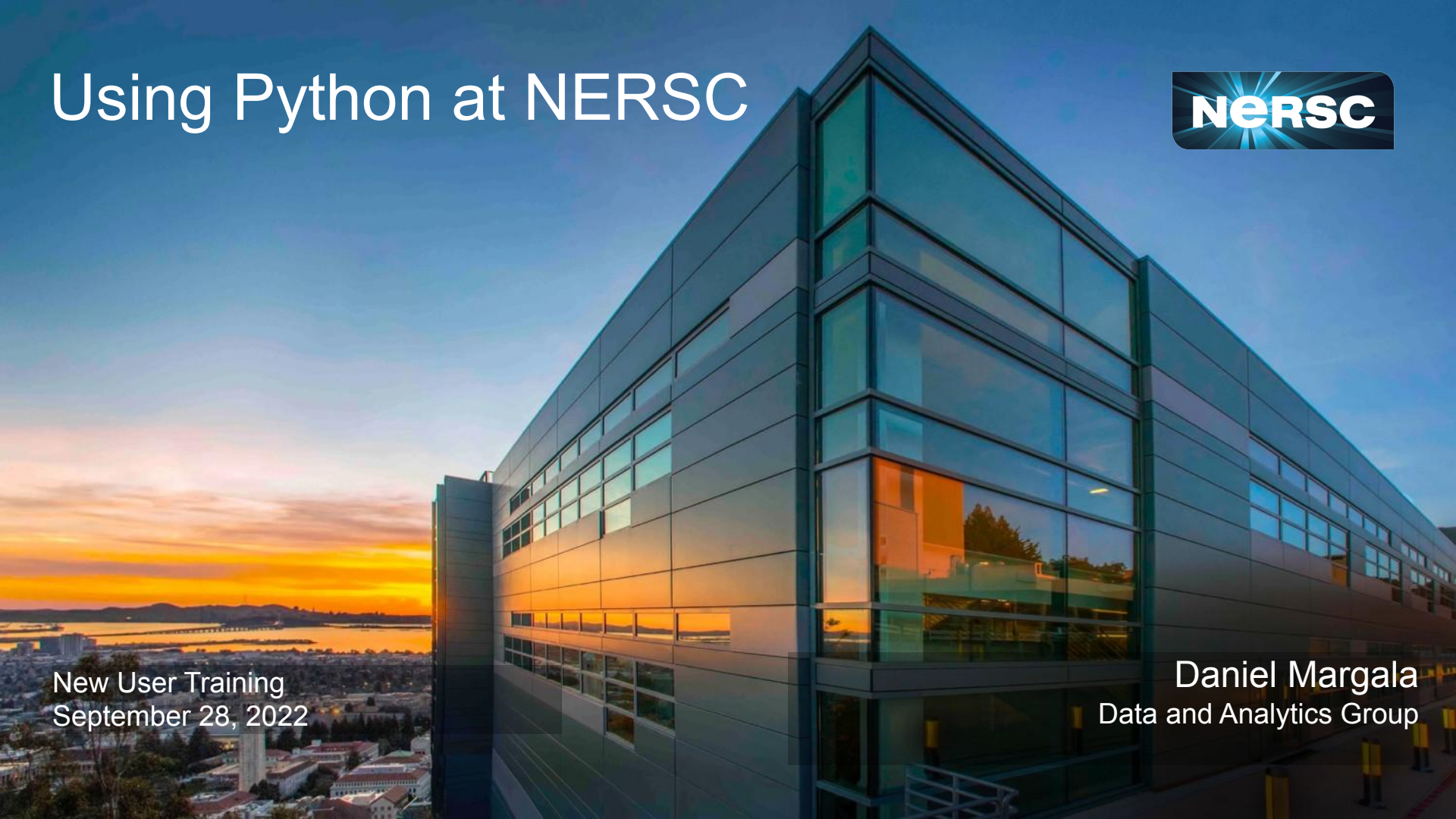
Using Python at NERSC



New User Training
September 28, 2022

Daniel Margala
Data and Analytics Group

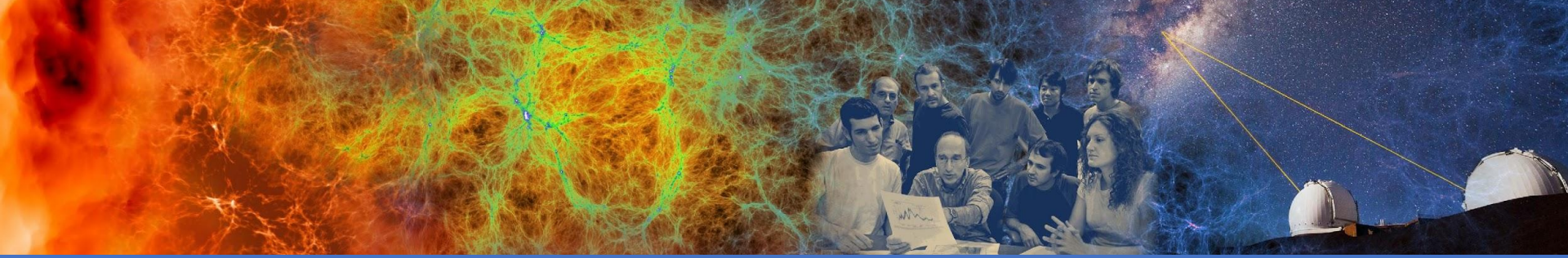
Python users, welcome to NERSC!

What we'll cover in this short 20 mins:

- Using Python at NERSC
- Getting started with Python on GPUs!
- Open Q&A (3-5 mins at end)

Not covered in this talk:

- Jupyter (see next presentation)



Using Python at NERSC

How do I use Python at NERSC?

- NERSC provides an Anaconda Python distribution for users, available via the “python” module:

```
perlmutter> module load python
perlmutter> python
Python 3.9.7 (default, Sep 16 2021, 13:09:58)
[GCC 7.5.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> print("Welcome to NERSC")
Welcome to NERSC
>>>
```

https://docs.nersc.gov/development/languages/python/

The NERSC Python Module

initializes conda for you (no need to modify ~/.bashrc or other shell startup files)

```
perlmutter> module load python
perlmutter> which conda
/global/common/software/.../3.9-anaconda-2021.11/condabin/conda
perlmutter> conda env list
# conda environments:
#
base         /global/common/software/.../3.9-anaconda-2021.11
lazy-h5py    /global/common/software/.../3.9-anaconda-2021.11/envs/lazy-h5py
lazy-mpi4py /global/common/software/.../3.9-anaconda-2021.11/envs/lazy-mpi4py

perlmutter> conda list
# packages in environment at /global/common/software/.../3.9-anaconda-2021.11:
#
# Name                    Version                   Build  Channel
...
```

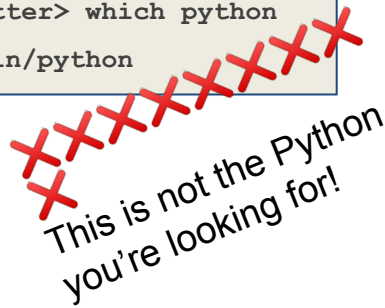
nearly 300 packages pre-installed

Other options for using Python at NERSC

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.
>>>
```

```
perlmutter> which python
/usr/bin/python
```



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/

Package installation tips:

- Most packages installed via conda or pip should work at NERSC
  - packages installed via conda can come from different “channels”. Channels are specified with “`-c defaults`” or “`-c conda-forge`”.
  - In many cases it’s fine to mix packages from different channels and/or pip but this can sometimes lead to version conflicts. Check the packages installed in your environment with “`conda list`”
- Some python packages should be compiled with the “compiler wrappers” available on the system. For example, mpi4py (see next slide) and h5py (if you’re using parallel IO).
- cudatoolkit: module vs conda package:
  - Some GPU-enabled packages installed from conda-forge will install cudatoolkit into your conda environment. This may conflict with the cudatoolkit module that is loaded by default.

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
```

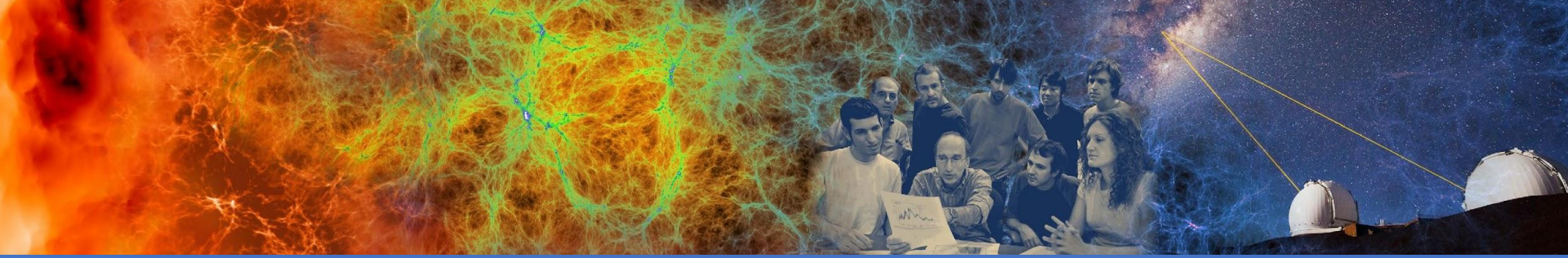
- Be aware that with any CUDA-aware mpi4py, you must have **`cudatoolkit`** loaded, even for code that does not use the GPU

Use pip with caution ⚠️

- Be careful with pip!!!! pip will try to be clever and find existing packages to save time, but sometimes you don't want this
- Packages installed with `--user` are not confined to a particular environment
  - If you use `pip install --user <package>`, it will install packages to the location specified by `PYTHONUSERBASE`, which is by default `$HOME/.local/perlmutter/3.9-anaconda-2021.11`
- Best practices for pip:
  - Install packages inside of a conda environment, not outside (don't use `pip install --user <package>`)
  - Use `pip install --no-cache-dir --force-reinstall <package>` (Did you notice this in our mpi4py recipe?)

Best practices for Python at NERSC

- Use conda environments (or Shifter containers) for customizable Python sandboxes
- Use our `/global/common/software/<your project>` filesystem for better performance
- Use the compiler wrappers to build packages such as mpi4py
- Avoid running "`conda init`" which will hardcode conda initialization in your shell startup file ($HOME/.bashrc)
- be careful using pip
- avoid using the system python from /usr/bin !



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, PyOpenCL, PyCUDA, CUDA Python**
  - multi-node / distributed memory:
    - **mpi4py+X, dask, cuNumeric**
- 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 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)

```
> 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

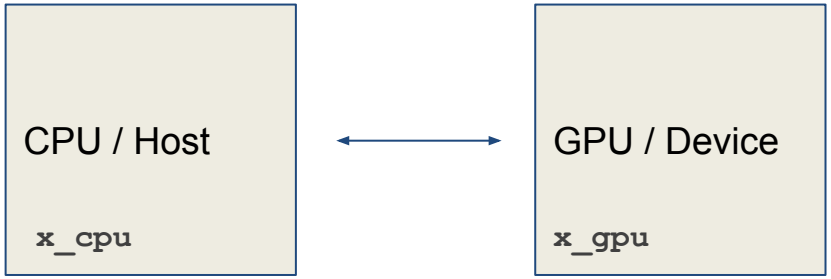
Check your package documentation to see cudatoolkit compatibility requirements

See documentation at https://docs.nersc.gov/development/languages/python/using-python-perlmutter/ 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, try 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
```

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)
```

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

```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
# (result is still on device)
total = numpy.sum(x)
```

- 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
from cupyx.profiler import time_range

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

@time_range(message)
def function():
    pass

with 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:

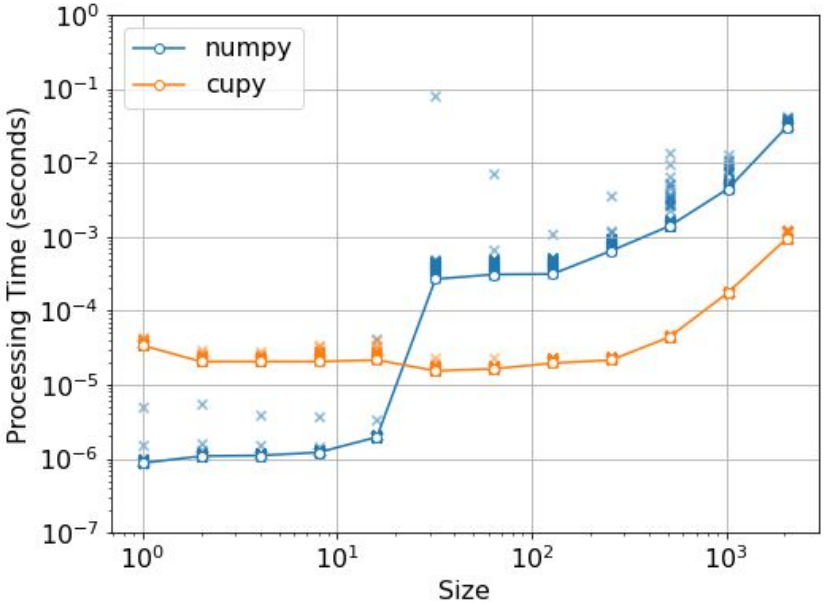
- Performs computation using large arrays, matrices, or images
- Dataset can fit in GPU memory
  - (40GB for Perlmutter's A100 GPUs)
- IO is not a bottleneck

For more help choosing a GPU-accelerated Python framework:

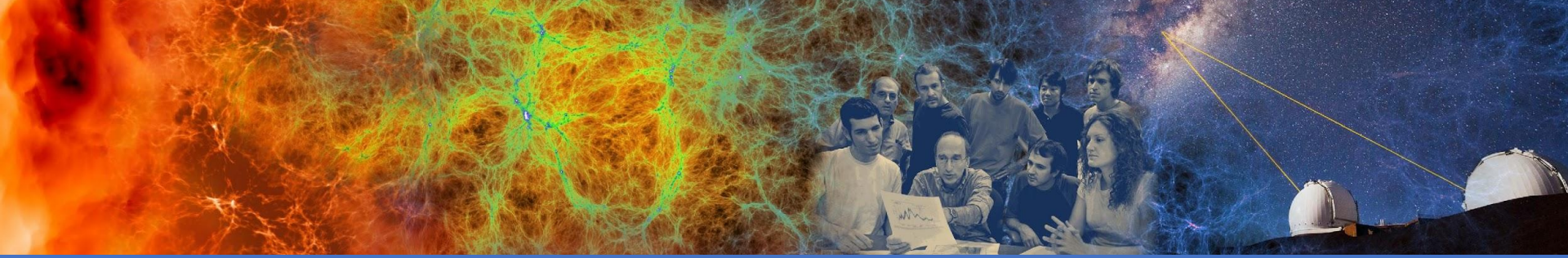
https://docs.nersc.gov/development/languages/python/perlmutter-prep/

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)
```



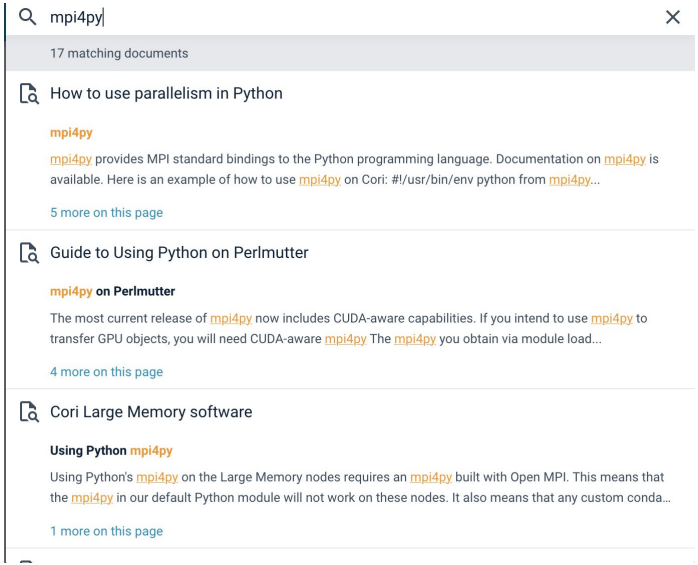
NERSC
BERKELEY LAB Bringing Science Solutions to the World
U.S. DEPARTMENT OF ENERGY Office of Science



Wrap Up

Where to get Python information

- 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 NERSC!
- We are here to help you use Python 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!

Thank You and Welcome to NERSC!