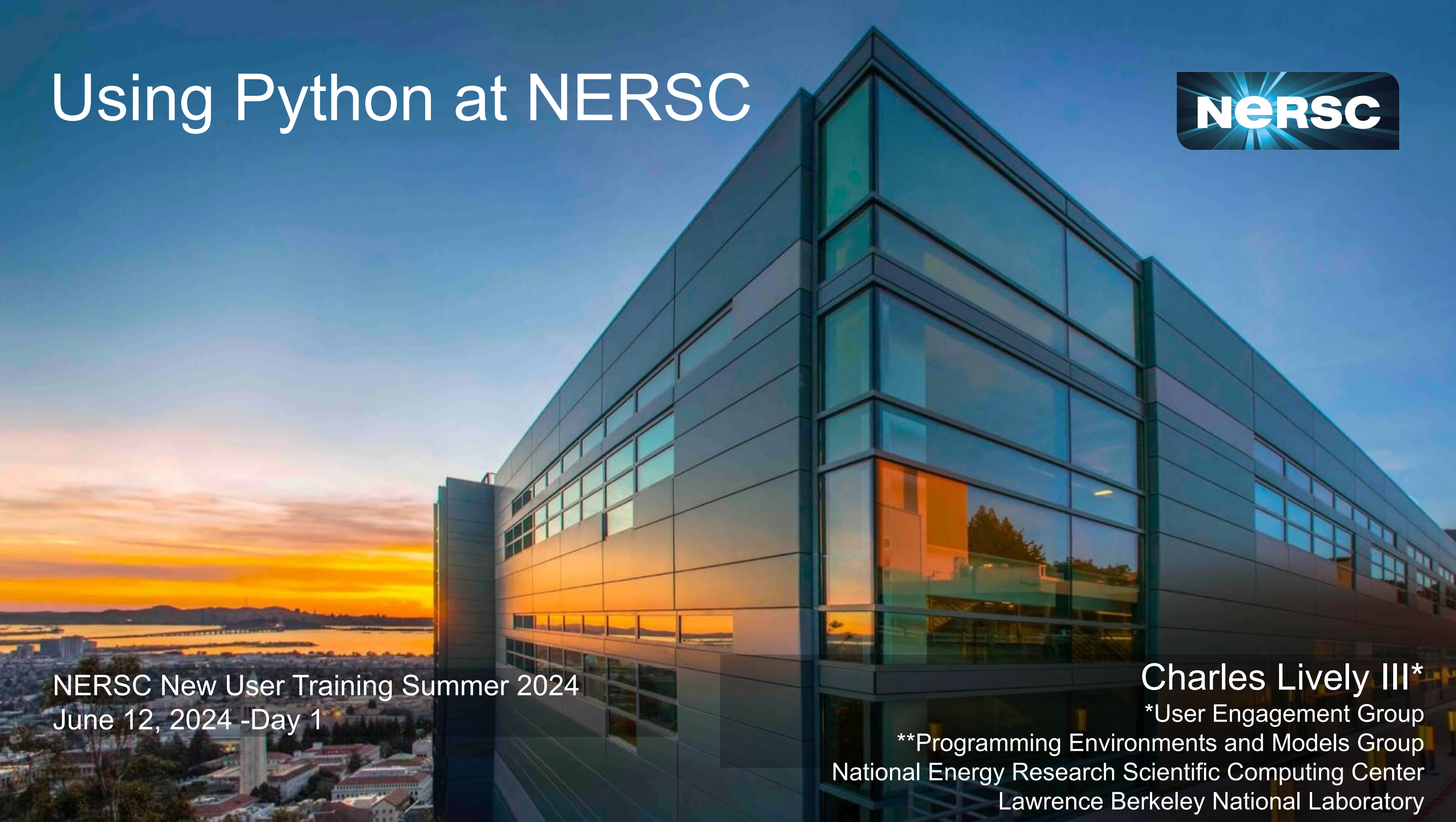
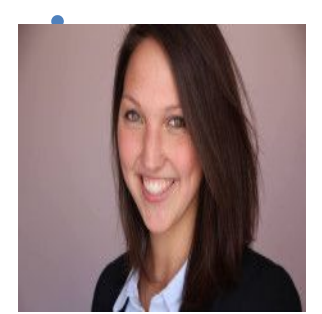


Using Python at NERSC
NERSC
NERSC New User Training Summer 2024
June 12, 2024 -Day 1
Charles Lively III*
*User Engagement Group
**Programming Environments and Models Group
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory

User Engagement Group - NERSC Warriors



Lisa Claus

Kevin Gott

Lipi Gupta

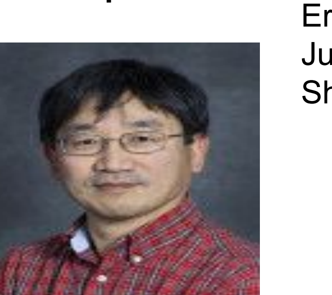
Rebecca Hartman-Baker
UEG Group Lead

Helen He

Charles Lively

Kelly Rowland

Kadidia Konate



Woo-Sun Yang

Alumni:

Tiffany Connors
Zhengji Zhao
Steve Leak
Erik Palmer
Justin Cook
Shahzeb Siddiqui

Python users, welcome to NERSC!

What we'll cover in this intro to Python at NERSC:

- Using Python at NERSC
    - python/conda modules
    - creating environments/installing packages
    - tips for parallel python
- Getting started with Python on GPUs!
    - high-level overview

Using Python at NERSC

How can I use Python at NERSC?

- To get started, load the python module:

```
> module load python/3.11

(nersc-python)> python
Python 3.11.7 | packaged by conda-forge | (main, Dec 23 2023,
14:43:09) [GCC 12.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.

>>> print("Welcome to NERSC New User Training June 2024!!!")
Welcome to NERSC New User Training June 2024!!!
```

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

The NERSC Conda Module

conda is an environment and package management tool that is very popular in the scientific python community.

conda environments are great for creating isolated and reproducible software environments for your projects. The conda package manager is great for installing and resolving package dependencies for your projects.

Loading the conda module initializes conda. There is no need to run “conda init” or initialize conda in your ~/.bashrc (or similar shell startup file).

The NERSC Python Module

The NERSC python module provides python via a conda environment. It's convenient for simple use cases that only need relatively common Python packages in the scientific computing.

```
> module load python/3.11
(nersc-python)> conda list
# packages in environment at /global/common/software/.../nersc-python:
#
# Name                    Version                          Build   Channel
…
matplotlib                3.8.2            py311h38be061_0    conda-forge
…
numpy                     1.26.3           py311h64a7726_0    conda-forge
…
scipy                     1.11.4           py311h64a7726_0    conda-forge
…
```

Other options for using Python at NERSC

Create a custom conda environment:

```
> module load conda
> conda create -n myenv python=3.11 numpy scipy
> conda activate myenv
(myenv)> python
Python 3.11.7 (main, Dec 23 2023, 14:43:09) [GCC 12.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

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

This is not the Python you're looking for!

Use Python from a container:

```
> shifter --image=docker:library/python:latest python
Python 3.11.4 (main, Aug 16 2023, 19:58:34) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

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 potentially h5py (if you're using parallel IO features).
- 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 with CUDA support in cray-mpich
- To install mpi4py with CUDA support in cray-mpich, follow this recipe:

```
> module load PrgEnv-gnu cudatoolkit craype-accel-nvidia80 conda
> conda create -n mpi4py-gpu python=3.9 -y
> conda activate mpi4py-gpu
> MPICC="cc -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 maintains a local package cache which , 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 may be set to something like `$HOME/.local/perlmutter/python-3.11`
- 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?)

Running Python at scale at NERSC

Python startup/module imports can put significant load on shared global filesystems, especially when running in parallel.

To avoid this we recommend:

- Use a container (run with shifter or podman)
- Use /global/common/software/<project>
  - use the -p/--prefix option when creating conda environments:

```
> conda create -p /global/common/software/<project>/envs/myenv python=3.11
> conda activate /global/common/software/<project>/envs/myenv
```

- Avoid $HOME
- Avoid $CFS

Other common parallel Python pitfalls

- Unexpected oversubscription due to indirect parallelism.
  - numpy uses OpenMP threading under the hood.
    - When using multiple processes, make sure num_processes_per_node x OMP_NUM_THREADS does not exceed the number of physical CPU cores per node.
  - Default worker pool size in multiprocessing.
    - It's common for Python applications to use the value of `os.cpu_count()` to set a default value for the number of processes / workers. This does not account for cpu-binding.
    - For nested parallel applications, you should specify the number of workers to use and not trust the default.

Using Python at NERSC Summary

- Use conda environments (or containers!) for customizable Python sandboxes.
- Use the `/global/common/software/<project>` filesystem (or containers!) for better performance when running in parallel.
- 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 !
- Watch out for defaults which may unexpectedly lead to oversubscription of resources.

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

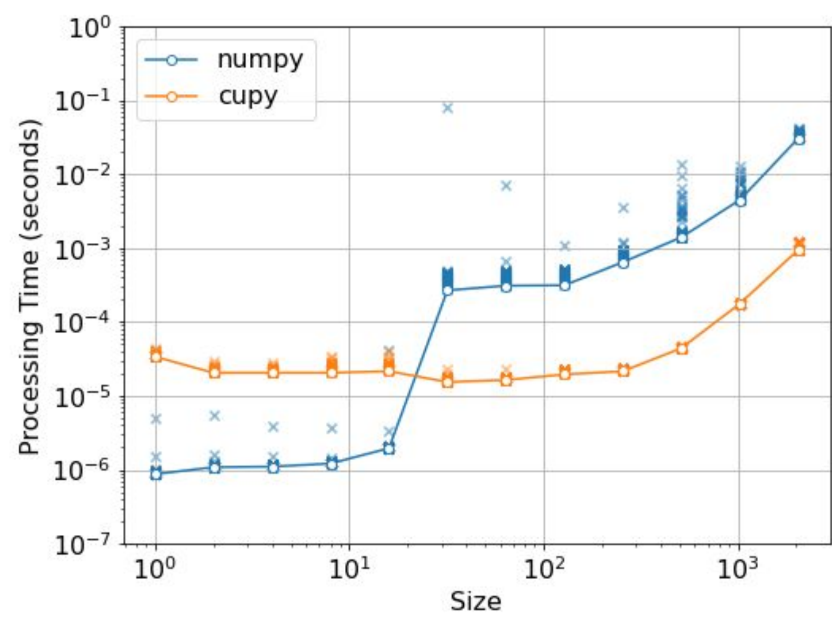
GPUs are likely a good fit if the following are true for your application:

- 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)
```

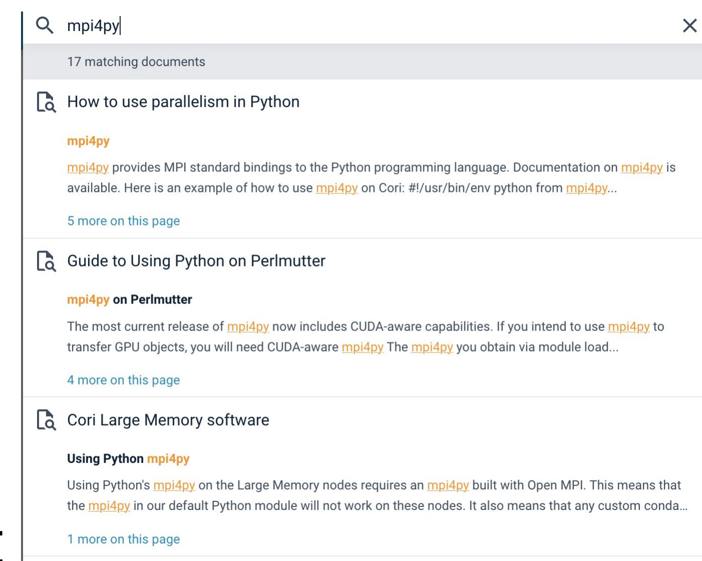


Wrap Up

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



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

Thank You and
Welcome to
NERSC!
NERSC