CAPS Computing Boot Camp

Day 2 - Advanced Topics

Cynthia Trendafilova
Srinivasan Raghunathan

Outline

- ❏ Slurm documentation
- ❏ Additional HPC resources
- ❏ Parallelization example with Python
- ❏ Astronomical data analysis with Jupyter notebooks

Slurm Documentation

Full documentation of **sbatch** options and syntax, used when submitting jobs:
https://slurm.schedmd.com/sbatch.html

Important highlights:

- **`-t, --time=<time>`**
  Set a limit on the total run time of the job allocation.
  CAPS computing etiquette: maximum **24 hours** duration per job.
- **`-d, --dependency=<dependency_list>`**
  Require that this job does not start until the **dependency** is met, e.g. job B will not start until job A has finished.

Slurm Documentation

We know how to **submit** jobs. How do we **cancel** jobs?
https://slurm.schedmd.com/scancel.html

- To cancel one job:
  ```
  scancel <jobid>
  ```
- To cancel all the jobs for a user:
  ```
  scancel -u <username>
  ```
- To cancel all the pending jobs for a user:
  ```
  scancel -t PENDING -u <username>
  ```

Slurm Documentation

Full documentation of **scontrol** options and syntax, used to suspend/resume jobs: https://slurm.schedmd.com/scontrol.html

Important highlights:

- **`suspend <job_list>`**
  Suspend a running job.Use the resume command to resume its execution.
- **`resume <job_list>`**
  Resume a previously suspended job.

Slurm Documentation

Full documentation of **sacct** options and syntax, used to retrieve information about jobs:
https://slurm.schedmd.com/sacct.html

By default, **sacct** will only pull up jobs that were run on the current day.

- `sacct --starttime=YYYY-MM-DD`

You can see many details about your past and current jobs:

- `sacct --starttime=YYYY-MM-DD --format=jobid,jobname,nnodes,ncpus,elapsed,state`

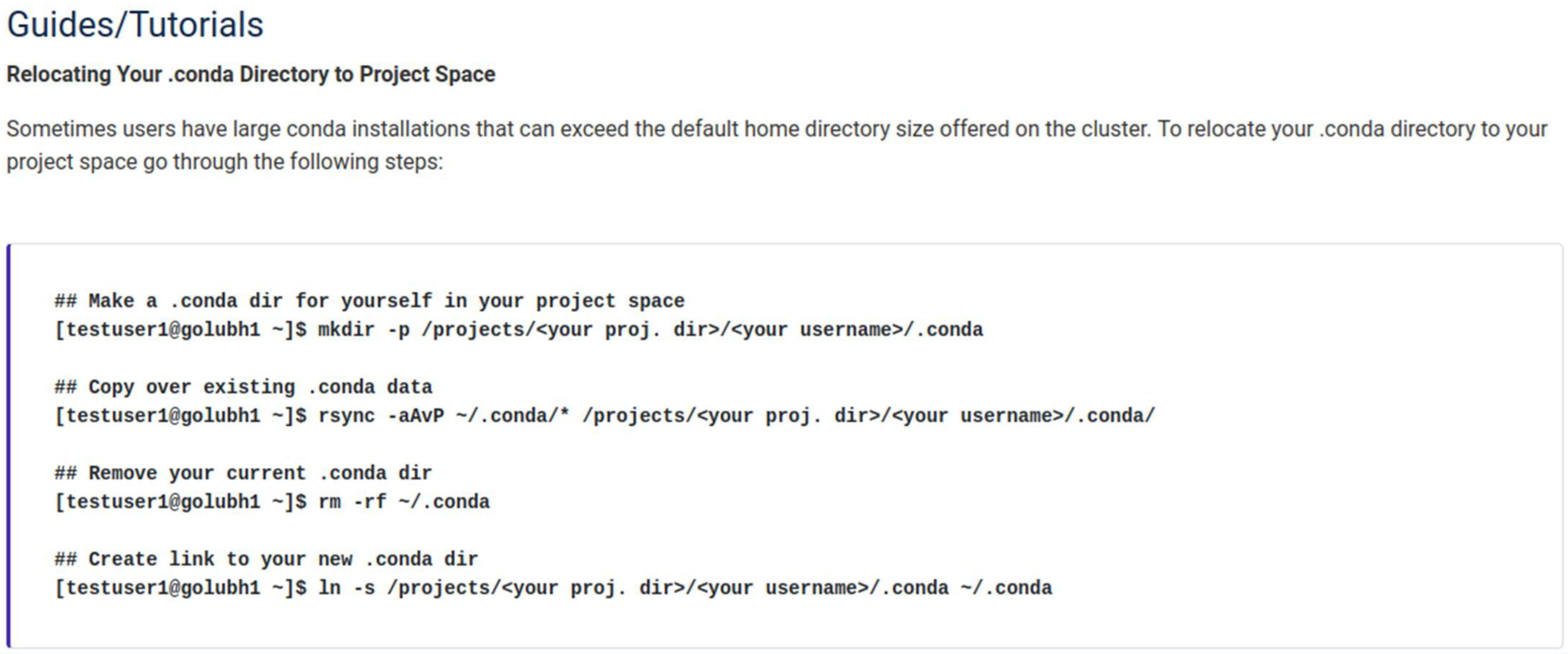
ICCP Documentation

Illinois Campus Cluster Program:
https://campuscluster.illinois.edu/resources/docs/

Many of these topics are being covered during this workshop. This website is a very helpful resource!

Relocating Your .conda Directory to Project Space

From: https://campuscluster.illinois.edu/resources/docs/storage-and-data-guide/



Guides/Tutorials

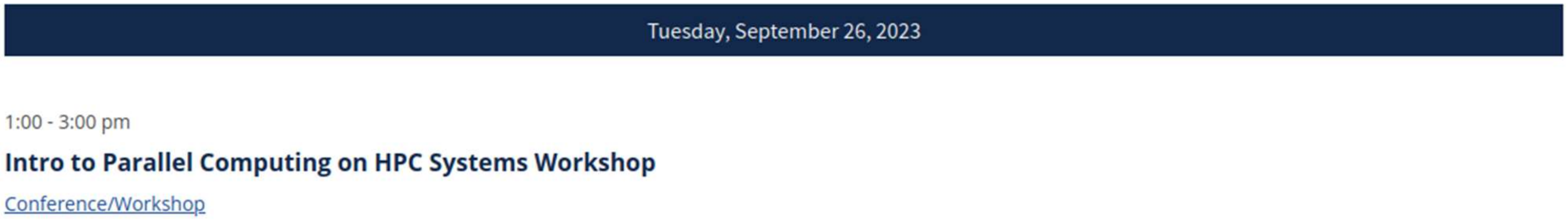
Relocating Your .conda Directory to Project Space

Sometimes users have large conda installations that can exceed the default home directory size offered on the cluster. To relocate your .conda directory to your project space go through the following steps:

```
## Make a .conda dir for yourself in your project space
[testuser1@golubh1 ~]$ mkdir -p /projects/<your proj. dir>/<your username>/.conda

## Copy over existing .conda data
[testuser1@golubh1 ~]$ rsync -aAvP ~/.conda/* /projects/<your proj. dir>/<your username>/.conda/

## Remove your current .conda dir
[testuser1@golubh1 ~]$ rm -rf ~/.conda

## Create link to your new .conda dir
[testuser1@golubh1 ~]$ ln -s /projects/<your proj. dir>/<your username>/.conda ~/.conda
```

Upcoming Workshops

NCSA hosts various computing workshops:
https://calendars.illinois.edu/list/7022


Tuesday, September 26, 2023

1:00 - 3:00 pm

Intro to Parallel Computing on HPC Systems Workshop

Conference/Workshop

Remember

https://caps.ncsa.illinois.edu/get-involved/comp_res_camp_clus/caps-campus-cluster-best-practices/

Follow best practices!

Ask questions!

Communicate!

Bonus: dependency examples

This will start after job **8474910** terminates with **ANY** exit code:

```
        #SBATCH -d afterany:8474910
```

This will start after job **8474910** terminates with **OK** exit code:

```
        #SBATCH -d afterok:8474910
```

This will start after job **8474910** terminates with **FAILED** exit code:

```
        #SBATCH -d afternotok:8474910
```

This will start if **my user** (ctrendaf) has **no other job** running with the **same name** (thisIsMyJobName):

```
        #SBATCH -J thisIsMyJobName
        #SBATCH -d singleton
```

Parallelization

MPI - Message Passing Interface

- Industry standard for exchanging messages between multiple cores/computers that are working on the same problem

Open MPI

- Open source MPI implementation

mpi4py - MPI for Python

- Provides Python bindings for the MPI standard

Parallelization

COMM: The communication "world" defined by MPI

RANK: an ID number given to each internal process to define communication

SIZE: total number of processes allocated

BROADCAST: One-to-many communication

SCATTER: One-to-many data distribution

GATHER: Many-to-one data distribution

Parallelization

mpi4py *can* be installed through Conda (which will install all dependencies, including MPI). This may be fine on personal computers.

On an **HPC cluster**, however, it is best to **use existing MPI modules on the cluster**.

On some clusters, Conda mpi4py may not work at all, while on others it may work but more slowly.

Parallelization - Python

Let’s create a new Conda environment for our mpi4py (and install numpy):

```
module load anaconda

conda create --name my-mpi4py python=3.8 numpy
conda activate my-mpi4py
```

Parallelization - Python

You can check all available modules on the Campus Cluster. Then, be sure to load a **matching** version of a compiler (in this case, we'll use gcc) and openmpi. For example:

```
module list
module avail
module load gcc/7.2.0
module load openmpi/4.1.0-gcc-7.2.0
module list
```

Parallelization - Python

Set the loaded compiler to be used for installing mpi4py and check that this variable was set correctly:

```
export MPICC=$(which mpicc)
echo $MPICC
```

You should see some output like:

```
/usr/local/mpi/rh7/openmpi/4.1.1/gcc/7.2.0/bin/mpicc
```

Parallelization - Python

Now, install mpi4py using pip:

```
python -m pip install mpi4py --no-cache-dir
```

And check if it worked:

```
python -c "import mpi4py"
    python -c "from mpi4py import MPI"
```

Both commands should produce no error messages.

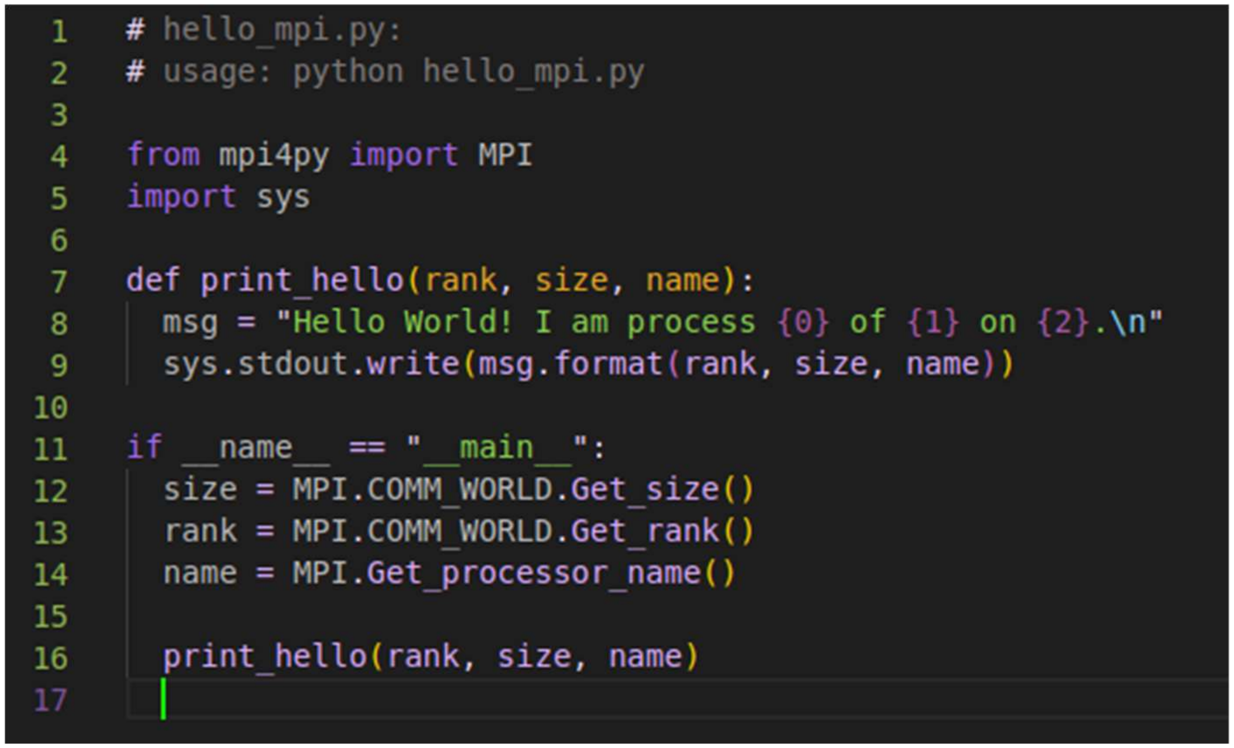
Parallelization - Python

Let’s run a simple script to test our mpi4py installation and see how tasks are assigned to different nodes!

```
mkdir myScripts
cd myScripts
touch hello_mpi.py
vim hello_mpi.py
```

Press **a** to enter **INSERT** mode, then write your code (next slide).

Parallelization - Python



```python
# hello_mpi.py:
# usage: python hello_mpi.py

from mpi4py import MPI
import sys

def print_hello(rank, size, name):
  msg = "Hello World! I am process {0} of {1} on {2}.\n"
  sys.stdout.write(msg.format(rank, size, name))

if __name__ == "__main__":
  size = MPI.COMM_WORLD.Get_size()
  rank = MPI.COMM_WORLD.Get_rank()
  name = MPI.Get_processor_name()

  print_hello(rank, size, name)

```

When finished, press **ESC**, then type **`:wq`** (write+quit) and hit **ENTER**.

Vi editor

Hit Esc+U to undo the edits.

Hit Esc+q! to quit w/o saving (incase of many wrong edits).

Hit Esc+w to save w/o quitting.

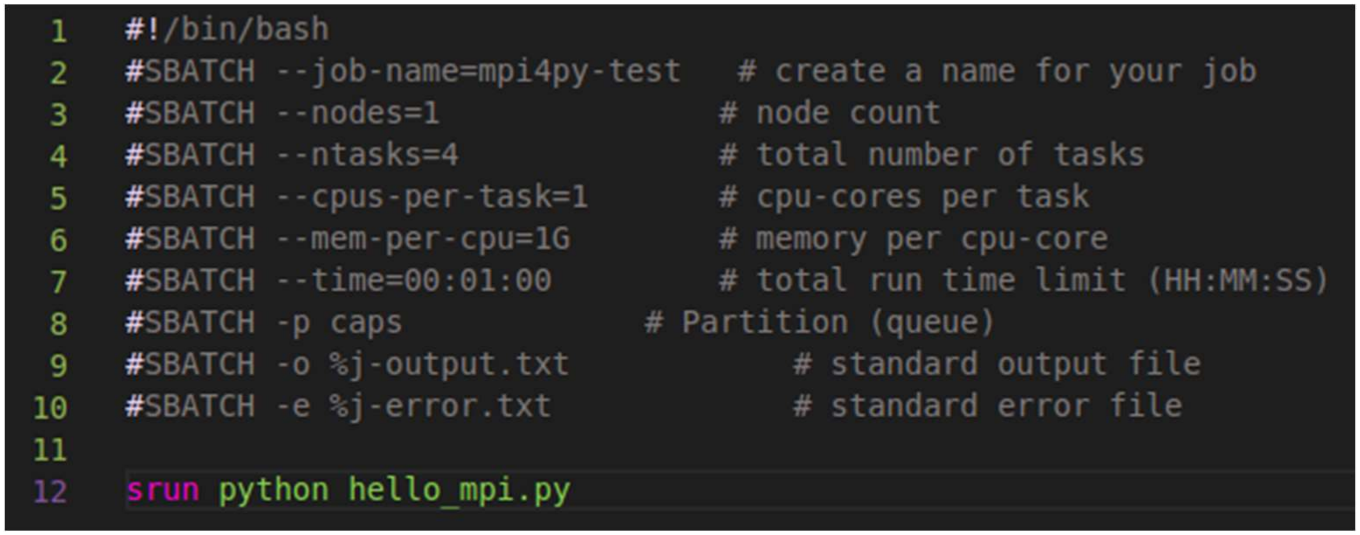
Hit Esc+n to search for a character (forward). Use Esc+N for backward.

Use \[...] for special characters.

Parallelization - Python

Let’s copy the sbatch file to submit the job to Slurm:

```
cp /home/ctrendaf/scratch/teaching/run_hello.sbatch .
```



```
#!/bin/bash
#SBATCH --job-name=mpi4py-test   # create a name for your job
#SBATCH --nodes=1               # node count
#SBATCH --ntasks=4              # total number of tasks
#SBATCH --cpus-per-task=1       # cpu-cores per task
#SBATCH --mem-per-cpu=1G        # memory per cpu-core
#SBATCH --time=00:01:00         # total run time limit (HH:MM:SS)
#SBATCH -p caps             # Partition (queue)
#SBATCH -o %j-output.txt            # standard output file
#SBATCH -e %j-error.txt             # standard error file

srun python hello_mpi.py
```

Parallelization - Python

Now submit it:

```
sbatch run_hello.sbatch
```

View the resulting output file with:

```
cat 0101010-output.txt
```

It should look like this:

```
Hello World! I am process 0 of 4 on ccc0344.campuscluster.illinois.edu.
     Hello World! I am process 1 of 4 on ccc0344.campuscluster.illinois.edu.
     Hello World! I am process 2 of 4 on ccc0344.campuscluster.illinois.edu.
     Hello World! I am process 3 of 4 on ccc0344.campuscluster.illinois.edu.
```

Parallelization - Python

Try changing the number of tasks:

```
    #SBATCH --ntasks=7                  # total number of tasks
```

What does your output look like?

Try also changing the number of nodes:

```
    #SBATCH --nodes=2                   # node count
```

What does your output look like?

Parallelization - Python

What if we want to control how many processes are distributed on each node?
Then we must specify the **number of tasks per node**:

```
    #SBATCH --ntasks=4              # total number of tasks
        #SBATCH --nodes=2                  # node count
        #SBATCH --ntasks-per-node=2   # Number of tasks per
node
```

What does your output look like?

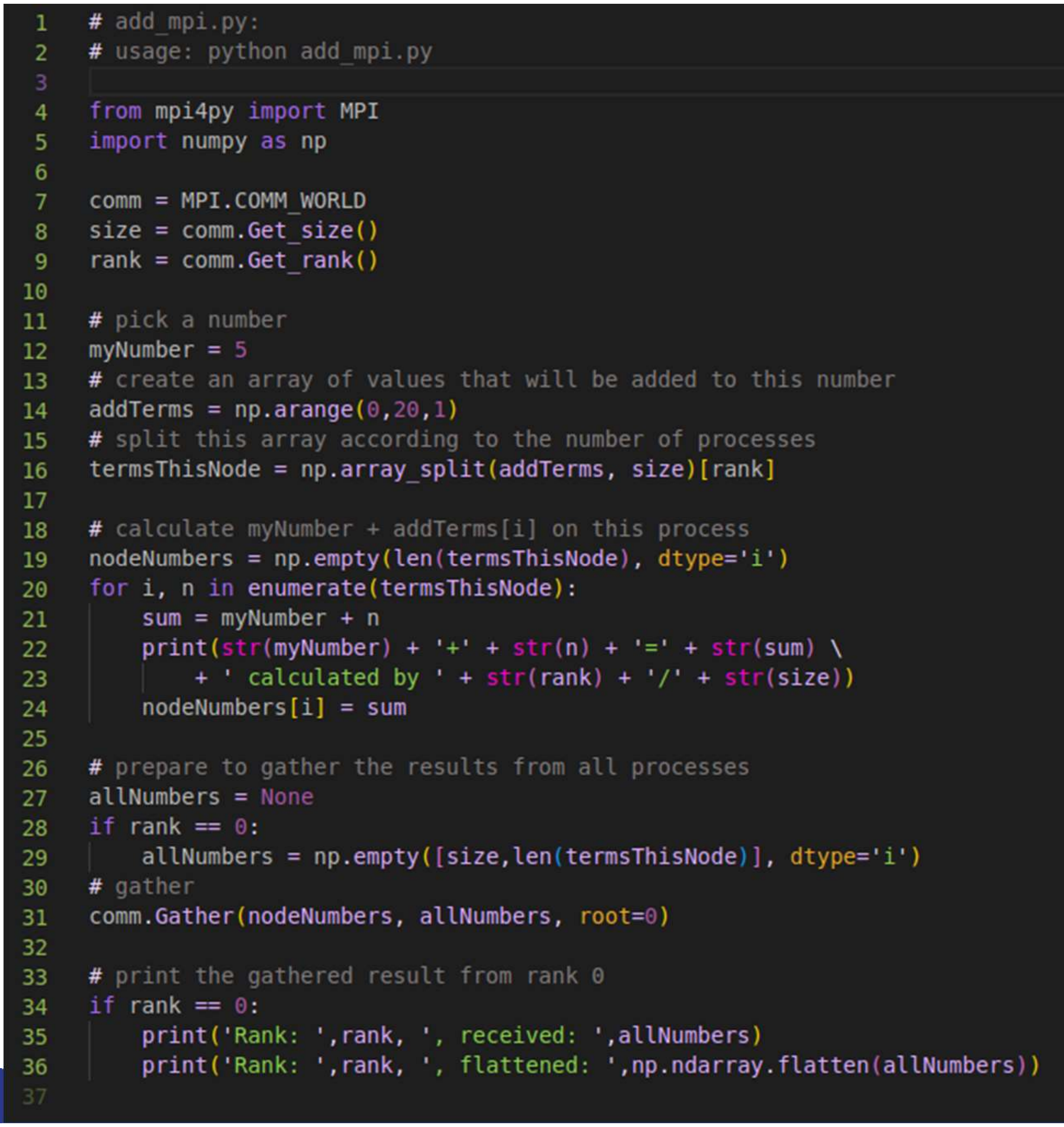
Parallelization - Python

This example will perform some simple arithmetic on each node, then **GATHER** the results from all nodes.

Copy the file **`add_mpi.py`**:

```
cp /home/ctrendaf/scratch/teaching/add_mpi.py .
```

Try running this with **4 total tasks**.

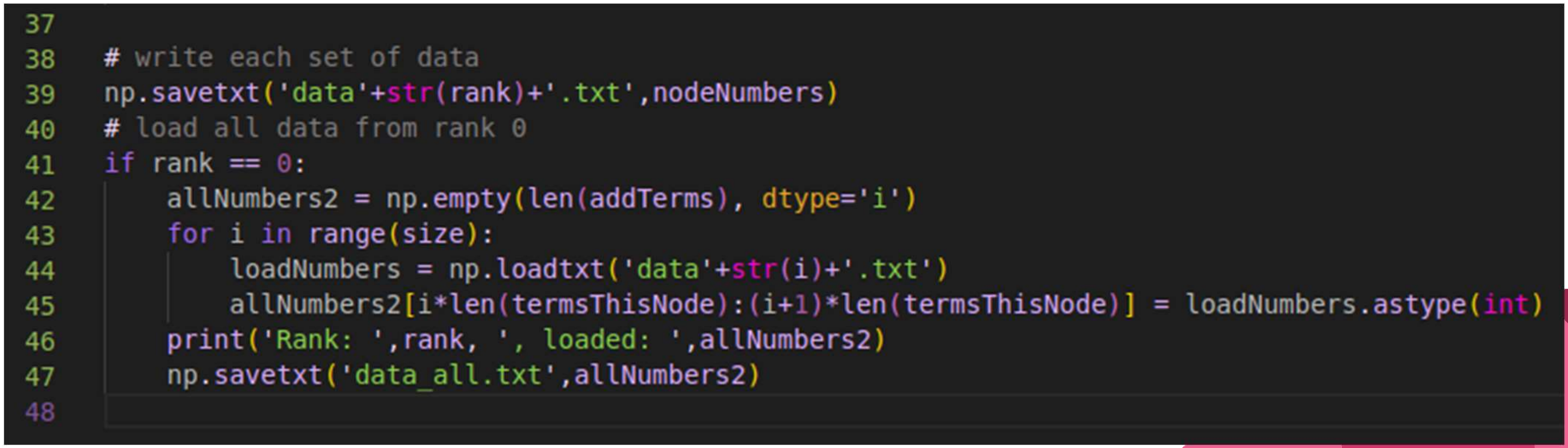


```python
# add_mpi.py:
# usage: python add_mpi.py

from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

# pick a number
myNumber = 5
# create an array of values that will be added to this number
addTerms = np.arange(0,20,1)
# split this array according to the number of processes
termsThisNode = np.array_split(addTerms, size)[rank]

# calculate myNumber + addTerms[i] on this process
nodeNumbers = np.empty(len(termsThisNode), dtype='i')
for i, n in enumerate(termsThisNode):
    sum = myNumber + n
    print(str(myNumber) + '+' + str(n) + '=' + str(sum) \
        + ' calculated by ' + str(rank) + '/' + str(size))
    nodeNumbers[i] = sum

# prepare to gather the results from all processes
allNumbers = None
if rank == 0:
    allNumbers = np.empty([size,len(termsThisNode)], dtype='i')
# gather
comm.Gather(nodeNumbers, allNumbers, root=0)

# print the gathered result from rank 0
if rank == 0:
    print('Rank: ',rank, ', received: ',allNumbers)
    print('Rank: ',rank, ', flattened: ',np.ndarray.flatten(allNumbers))
```

Parallelization - Python

We can **write** the data from each node to disk and **load it once** from node 0, instead of using **GATHER**.

Try adding these lines to your code:



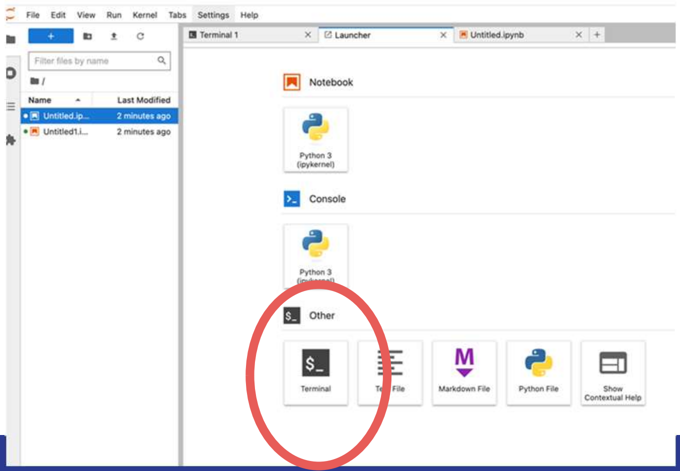
```python

# write each set of data
np.savetxt('data'+str(rank)+'.txt',nodeNumbers)
# load all data from rank 0
if rank == 0:
    allNumbers2 = np.empty(len(addTerms), dtype='i')
    for i in range(size):
        loadNumbers = np.loadtxt('data'+str(i)+'.txt')
        allNumbers2[i*len(termsThisNode):(i+1)*len(termsThisNode)] = loadNumbers.astype(int)
    print('Rank: ',rank, ', loaded: ',allNumbers2)
    np.savetxt('data_all.txt',allNumbers2)

```

Jupyter

Astronomical Data Analysis with Jupyter Notebooks

Jupyter

- Connect to https://jupyter.ncsa.illinois.edu/ [info] with your campuscluster credentials.
- Open a notebook and type:
  - import numpy as np
    - You will most likely get an error message "ModuleNotFoundError: No module named 'numpy'"
  - Within the Jupyter notebook, you also have an option to open the "terminal"



Jupyter

- Now install the necessary modules:
  - pip install numpy
  - pip install matplotlib
  - pip install scipy
  - pip install astropy
  - pip install pandas

  More advanced

  - pip install healpy
  - pip install camb
- https://github.com/sriniraghunathan/cosmology_school
  - https://github.com/sriniraghunathan/cosmology_school/blob/main/caps_computing_bootcamp.ipynb

https://forms.gle/GeBgSgTLknkd8F3s6

Happy computing!