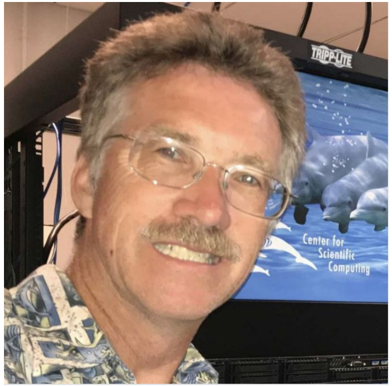
UC **SANTA BARBARA**

Parallel Programming with Python on HPC

Paul Weakliem, Fuzzy Rogers, and Jay Chi

March 01, 2023

Our Team



Paul Weakliem, PhD
Co-Director
Center for Scientific Computing &
California Nanosystems Institute
Eling 3231
weakliem@cnsi.ucsb.edu

Fuzzy Rogers
That guy in the MRL
Center for Scientific Computing &
Materials Research Laboratory
MRL 2066B
fuz@ucsb.edu

Yu-Chieh "Jay" Chi, PhD
Research Computing Consultant
Center for Scientific Computing &
Enterprise Technology Services
Elings 3229
jaychi@ucsb.edu

CSC Center for Scientific Computing

CNSI

UC SANTA BARBARA MRL Materials Research Laboratory
NSF Materials Research Science and Engineering Center [DMR 1720256]

ENTERPRISE TECHNOLOGY SERVICES

Purposes of the Workshop

- Write an Python code and use the HPC resource to get your computational result efficiently.

- What you will do in this workshop
  - Quickly get started learning parallel Python programming
  - Learn different parallel Python modules
  - Implement some basic algorithms by using parallel techniques
  - Basic benchmark the code and address the performance issues

- Basic Python Workshop
  - UCSB Software Carpentries (https://ucsbcarpentry.github.io/?field_location_tid=All&field_series_tid=1218)

Suggestions

- Install Anaconda
  - $ wget https://repo.anaconda.com/archive/Anaconda3-2022.10-Linux-x86_64.sh
  - $ sh Anaconda3-*.sh
- Create your Environment
  - $ conda create –name parallel_env
  - $ conda env list
  - $ conda activate parallel_env
- Install Python Packages
  - $ conda install numpy scipy sympy pandas matplotlib
  - $ conda install -c conda-forge multiprocess
  - $ conda install -c conda-forge mpi4py
  - $ conda install -c anaconda pillow
  - $ conda install -c conda-forge glob2
  - $ conda install -c conda-forge cupy cudatoolkit=11.0

Configure the environment

1. ssh to POD cluster from your local machine

   $ ssh your_user_name@pod-login1.cnsi.ucsb.edu

2. Load the openmpi module

   $ module load openmpi/3.1.3

3. Export the Anaconda Path

   $ export PATH=/sw/csc/anaconda/anaconda3/bin:$PATH

4. Check your Python

   $ which python

   /sw/csc/anaconda/anaconda3/bin/python

5. Copy Files to your directory

   https://drive.google.com/drive/folders/1GLtvL3eRCqCZnubVbKcyt-XcvsW3IXy3?usp=sharing

Parallel Modules

- There are many different Python parallel modules. Please refer the link: https://wiki.python.org/moin/ParallelProcessing
- In this workshop, we will introduce following parallel modules
  - multiprocessing
  - mpi4py
- Parallel programming is a broad with numerous possibilities for learning. The workshop introduces some parallel modules available in Python for simple parallel programming.
- If you are interested in the parallel programming, you can take parallel programming and parallel algorithm class.

Scenario (Distributed Computing)

Professor

<u>Exam:</u>
16 Questions
300 Students

Scenario

Teaching Assistants

TA #1 TA #2 TA #3

Data Parallelism

75 Exams per everyone

TA #1

TA #2

TA #3

The Multiprocessing Module

- Two simple classes from the multiprocessing module we are going to use for today's workshop:
  - Process class
  - Pool Class
- Process class represents an activity that will be run in a separate process and execute a function across multiple values in parallel.
- The Pool class represents a pool of worker processes, and control a set of worker processes via parallel map implementation.
- Ref: https://docs.python.org/3/library/multiprocessing.html

Sequential Example

```
import numpy as np
import time

def task_sleep(job, sec):
    print(f'Task {job} Starts to SLEEP now!!!!')
    time.sleep(sec)
    print(f'Task {job} Done for SLEEP!!!')


sleep_time = 1

num_jobs = 5

# Time counter
start_time = time.perf_counter()

for idx in range(num_jobs):
    task_sleep(idx, sleep_time)

end_time = time.perf_counter()
exe_time = end_time - start_time
print("Time taken: %.10f" %exe_time)
```

```
[jay@pod-login1 MultiPro_NEW]$ python mp_process_seq.py
Task 0 Starts to SLEEP now!!!!
Task 0 Done for SLEEP!!!
Task 1 Starts to SLEEP now!!!!
Task 1 Done for SLEEP!!!
Task 2 Starts to SLEEP now!!!!
Task 2 Done for SLEEP!!!
Task 3 Starts to SLEEP now!!!!
Task 3 Done for SLEEP!!!
Task 4 Starts to SLEEP now!!!!
Task 4 Done for SLEEP!!!
Time taken: 5.0131262760
```

The Process class

```python
import multiprocessing as mp
import numpy as np
import time

def task_sleep(job, sec):
    print(f'Task {job} Starts to SLEEP now!!!!')
    time.sleep(sec)
    print(f'Task {job} Done for SLEEP!!!')


sleep_time = 1

# Request No. of Cores
#####n_proc = os.getenv('SLURM_NTASKS', '1')  # env var is always a 'str'
#####n_proc = int(n_proc)                     # convert to 'int'
n_proc = 5
print('Number of Processor is requested: ',  n_proc)

# Time counter
start_time = time.perf_counter()

p0 = mp.Process(target=task_sleep, args=(0, sleep_time))
p1 = mp.Process(target=task_sleep, args=(1, sleep_time))
p2 = mp.Process(target=task_sleep, args=(2, sleep_time))
p3 = mp.Process(target=task_sleep, args=(3, sleep_time))
p4 = mp.Process(target=task_sleep, args=(4, sleep_time))

p0.start()
p1.start()
p2.start()
p3.start()
p4.start()

p0.join()
p1.join()
p2.join()
p3.join()
p4.join()

end_time = time.perf_counter()
exe_time = end_time - start_time
print("Time taken: %.10f" %exe_time)
```

```
[jay@pod-login1 MultiPro_NEW]$ python mp_process_para.py
Number of Processor is requested:  5
Task 0 Starts to SLEEP now!!!!
Task 1 Starts to SLEEP now!!!!
Task 2 Starts to SLEEP now!!!!
Task 3 Starts to SLEEP now!!!!
Task 4 Starts to SLEEP now!!!!
Task 0 Done for SLEEP!!!
Task 1 Done for SLEEP!!!
Task 2 Done for SLEEP!!!
Task 3 Done for SLEEP!!!
Task 4 Done for SLEEP!!!
Time taken: 1.0135198948
```

The Process class ~ using the for loop

```python
import multiprocessing as mp
import numpy as np
import time

def task_sleep(job, sec):
    print(f'Task {job} Starts to SLEEP now!!!!')
    time.sleep(sec)
    print(f'Task {job} Done for SLEEP!!!')


sleep_time = 1

# Request No. of Cores
#####n_proc = os.getenv('SLURM_NTASKS', '1')  # env var is always a 'str'
#####n_proc = int(n_proc)                      # convert to 'int'

n_proc = 10

# Time counter
start_time = time.perf_counter()

pro_id = []
for idx in range(n_proc):
    p = mp.Process(target=task_sleep, args=(idx, sleep_time))
    #print('P: ', p)
    p.start()
    pro_id.append(p)

#print("Proc_ID: ", pro_id)
for proc in pro_id:
    proc.join()
   # print('Proc: ', proc)

end_time = time.perf_counter()
exe_time = end_time - start_time
print("Time taken: %.10f" %exe_time)
```

```
[jay@pod-login1 MultiPro_NEW]$ python mp_process_para_for.py
Task 0 Starts to SLEEP now!!!!
Task 1 Starts to SLEEP now!!!!
Task 2 Starts to SLEEP now!!!!
Task 3 Starts to SLEEP now!!!!
Task 4 Starts to SLEEP now!!!!
Task 5 Starts to SLEEP now!!!!
Task 6 Starts to SLEEP now!!!!
Task 7 Starts to SLEEP now!!!!
Task 8 Starts to SLEEP now!!!!
Task 9 Starts to SLEEP now!!!!
Task 0 Done for SLEEP!!!
Task 1 Done for SLEEP!!!
Task 2 Done for SLEEP!!!
Task 3 Done for SLEEP!!!
Task 4 Done for SLEEP!!!
Task 6 Done for SLEEP!!!
Task 5 Done for SLEEP!!!
Task 7 Done for SLEEP!!!
Task 8 Done for SLEEP!!!
Task 9 Done for SLEEP!!!
Time taken: 1.0177016947
```

The Pool class

- The Pool class in multiprocessing can handle an enormous number of processes. It allows you to run multiple jobs per process.
- Pool class comes with different functions:
    - apply()
    - apply_async()
    - map()
    - map_sasync()
    - imap()
    - imap_unordered()
    - starmap()
    - Starmap_async()
- The map function supports concurrency, but does not accept multiple arguments.
- Ref: https://docs.python.org/3/library/multiprocessing.html

The Pool.map() function

```
def task_sleep(sec):
    print(f'PID = {os.getpid()}, Starts to SLEEP now!!!!')
    time.sleep(sec)
    print(f'PID = {os.getpid()}, Done for SLEEP!!!')

sleep_time = 2
n_proc = 5
print('\nNo. of core is requested: ', n_proc, '\n')

sleep_list = [int(sleep_time) for i in range(n_proc)]

start_time = time.perf_counter()
with mp.Pool(processes = n_proc) as pool:
    pool.map(task_sleep, sleep_list)
end_time = time.perf_counter()

print("Elapsed Time: ", end_time-start_time, "sec.")
```

```
[[jay@pod-login1 MultiPro_NEW]$ python mp_map.py

No. of core is requested:  5

PID = 156875, Starts to SLEEP now!!!!
PID = 156876, Starts to SLEEP now!!!!
PID = 156877, Starts to SLEEP now!!!!
PID = 156878, Starts to SLEEP now!!!!
PID = 156879, Starts to SLEEP now!!!!
PID = 156875, Done for SLEEP!!!
PID = 156878, Done for SLEEP!!!
PID = 156876, Done for SLEEP!!!
PID = 156879, Done for SLEEP!!!
PID = 156877, Done for SLEEP!!!
Elapsed Time:  2.046982287429273 sec.
```

SLURM job script

```
#!/bin/bash
#SBATCH --job-name='Py_MultiPro' ### -J 'testJob'
#SBATCH --ntasks=20                ### -n 1
#SBATCH -p batch                   ### Partition to submit job to
#SBATCH -o outLog
#SBATCH -e errLog
#SBATCH -t 00:10:00

####SBATCH --mail-user=your_account@ucsb.edu
####SBATCH --mail-type ALL

module load openmpi/3.1.3
export PATH=/sw/csc/anaconda/anaconda3/bin:$PATH

cd $SLURM_SUBMIT_DIR

python mp_process_para_for.py
```

Example 1: Monte Carlo PI Calculation

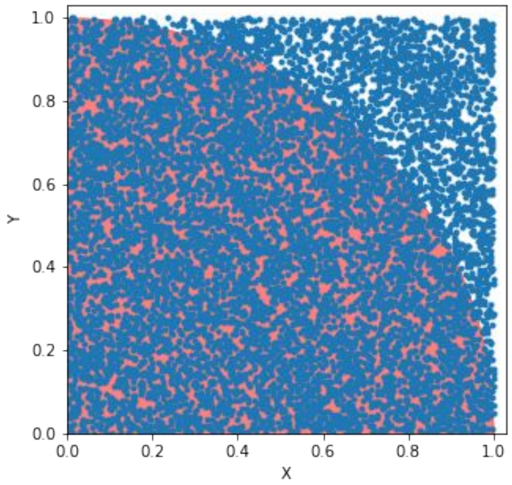
- https://en.wikipedia.org/wiki/Monte_Carlo_method#/media/File:Pi_30K.gif
- The error in the MC estimate

$$\epsilon_{mc} \sim \frac{1}{\sqrt{n}}$$

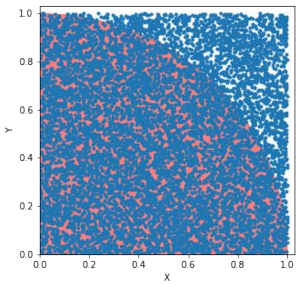
This dependence is foreshadowed by the beautiful theory called the **central limit theorem (CLT)**.

- We know that the area of square is $4r^2$, and the area of circle is πr^2. PI can be estimated as the ratio of these two area as following:

$$\pi = 4 \times \frac{\text{No. of points generated inside the circle}}{\text{No. of points generated inside the square}}$$



MC PI Calculation ~ Sequential Code




```python
import os
import time
import numpy as np
import multiprocessing as mp

def pi_mc(num_gen):
    count = 0
    np.random.seed()

    for i in range(num_gen):
        x_val = np.random.random_sample()
        y_val = np.random.random_sample()

        radius = x_val*x_val + y_val*y_val

        if radius <= 1.0:
            count = count + 1

    print(f"PID = {os.getpid()}, No. of Samples is {num_gen}\n")
    return count
```

```python
num_gen = 10000000

# Serial
start = time.time()
mc_cnt = pi_mc(num_gen)
PI_approx = 4*mc_cnt/num_gen
end = time.time()
print("Monta Carlo PI is: ", PI_approx)
print("Time: ", end - start)
```

MC PI Calculation ~ Pool.map() function

```
    #####n_proc = os.getenv('SLURM_NTASKS', '1')  # env var is always a
    #####n_proc = int(n_proc)                      # coerce to 'int'
    n_proc = 20
    print('Number of core is requested: ',  n_proc, '\n')

    partial = [int(num_gen/n_proc) for i in range(n_proc)]

    start = time.perf_counter()
    with mp.Pool(processes=n_proc) as pool:
        cnt_arr = pool.map(pi_mc, partial)
    PI_approx_map = 4*np.sum(cnt_arr)/num_gen
    print("PI_approx_multi_Core: ", PI_approx_map)
    end = time.perf_counter()
    print("Time: ", end - start)
```

```
PID = 257633, No. of Samples is 10000000.
Monta Carlo PI is:  3.1413648
Time:  5.758965492248535

Number of core is requested:  20

PID = 257693, No. of Samples is 500000.
PID = 257684, No. of Samples is 500000.
PID = 257685, No. of Samples is 500000.
PID = 257683, No. of Samples is 500000.
PID = 257696, No. of Samples is 500000.
PID = 257690, No. of Samples is 500000.
PID = 257688, No. of Samples is 500000.
PID = 257692, No. of Samples is 500000.
PID = 257689, No. of Samples is 500000.
PID = 257687, No. of Samples is 500000.
PID = 257700, No. of Samples is 500000.
PID = 257698, No. of Samples is 500000.
PID = 257695, No. of Samples is 500000.
PID = 257697, No. of Samples is 500000.
PID = 257691, No. of Samples is 500000.
PID = 257686, No. of Samples is 500000.
PID = 257694, No. of Samples is 500000.
PID = 257702, No. of Samples is 500000.
PID = 257699, No. of Samples is 500000.
PID = 257701, No. of Samples is 500000.
PI_approx_multi_Core:  3.141742
Time:  0.3860473190434277
```

Process Class with shared data

- In multiprocessing module programming, we might need to share data between processes.
- This can be achieved using shared memory via shared ctypes.
- What Are ctypes?
  - The ctypes module provides tools for working with C data types.
  - The ctypes module allows Python code to read, write, and generally interoperate with data using standard C data types.
- What are shared ctypes?
  - Python provides the capability to share ctypes between processes on one system.
  - This is primarily achieved via the following classes:
    - multiprocessing.Value
    - multiprocessing.Array
- Ref: https://superfastpython.com/multiprocessing-shared-ctypes-in-python/

MC PI Calculation ~ Process Class with shared data

```python
    #n_proc = os.getenv('SLURM_NTASKS', '1')  # env var is always a 'str'
    #n_proc = int(n_proc)                     # coerce to 'int'
    n_proc = 20
    print('Number of Processor is requested: ',  n_proc, '\n')

    init_zeros = [0 for i in range(n_proc)]

    arr_seed = mp.Array('i', range(n_proc))
    arr_cnt = mp.Array('i', init_zeros)
    partial = int(num_gen/n_proc)

    num_proc = []

    start_time = time.perf_counter()

    for idx in range(n_proc):
        p = mp.Process(target=pi_mc_para, args=(idx, arr_cnt, partial, idx))
        p.start()
        num_proc.append(p)

    for proc in num_proc:
        proc.join()

    PI_approx_para = 4*np.sum(arr_cnt)/num_gen

    end_time = time.perf_counter()
    print("Time: ", end_time - start_time)
    print(arr_cnt[:])
    print("Monta Carlo PI Parallel: ", PI_approx_para)
```

```
PID = 4444, No. of Samples is 10000000

Monta Carlo PI is:  3.1414284
Time:  5.302408933639526
Number of Processor is requested:  20

No. of Samples is 500000 in Process 4 with Rand Seed 4
No. of Samples is 500000 in Process 0 with Rand Seed 0
No. of Samples is 500000 in Process 2 with Rand Seed 2
No. of Samples is 500000 in Process 1 with Rand Seed 1
No. of Samples is 500000 in Process 5 with Rand Seed 5
No. of Samples is 500000 in Process 6 with Rand Seed 6
No. of Samples is 500000 in Process 3 with Rand Seed 3
No. of Samples is 500000 in Process 8 with Rand Seed 8
No. of Samples is 500000 in Process 7 with Rand Seed 7
No. of Samples is 500000 in Process 9 with Rand Seed 9
No. of Samples is 500000 in Process 10 with Rand Seed 10
No. of Samples is 500000 in Process 14 with Rand Seed 14
No. of Samples is 500000 in Process 12 with Rand Seed 12
No. of Samples is 500000 in Process 15 with Rand Seed 15
No. of Samples is 500000 in Process 13 with Rand Seed 13
No. of Samples is 500000 in Process 16 with Rand Seed 16
No. of Samples is 500000 in Process 17 with Rand Seed 17
No. of Samples is 500000 in Process 19 with Rand Seed 19
No. of Samples is 500000 in Process 18 with Rand Seed 18
No. of Samples is 500000 in Process 11 with Rand Seed 11
Time:  0.34324445482343435
Monta Carlo PI Parallel:  3.1408292
```

MC PI Calculation ~ Process Class with shared data

```python
def pi_mc(proc, count, num_gen, seed):
    np.random.seed(seed)

    count = 0
    for i in range(num_gen):
        x_val = np.random.random_sample()
        y_val = np.random.random_sample()

        radius = x_val*x_val + y_val*y_val

        if radius <= 1.0:
            count = count + 1
```

```python
    mc_cnt = 0
    num_gen = 10000000
    seed = 1

    # Serial
    start = time.time()
    mc_cnt = pi_mc(seed, mc_cnt, num_gen, seed)
    PI_approx = 4*mc_cnt/num_gen
    end = time.time()
    print("Monta Carlo PI is: ", PI_approx)
    print("Time: ", end - start)
```

```python
def pi_mc_para(proc, count, num_gen, seed):
    #np.random.seed(seed[proc])
    np.random.seed(seed)

    cnt = 0
    for i in range(num_gen):
        x_val = np.random.random_sample()
        y_val = np.random.random_sample()

        radius = x_val*x_val + y_val*y_val

        if radius <= 1.0:
            cnt = cnt + 1
            #count[proc] = count[proc]+1

    count[proc] = cnt
```

Example 2: Add Gaussian Noise signal to the Image

- We have multiple images in the folder.
- Each image is given an image of (H*W*3) dimensions. Let us write a program to add Gaussian noise to the image.
- We can directly use np.random.normal(mu, sigma, size) to sample a pixel intensity from a Gaussian distribution. We can specify mu as 0, and sigma as the standard deviation.
- Next, generate a (H*W*3) dimensional Gaussian noise array, where H is the height of the image, W is the width, and 3 is the (RGB) channels. Then, add this Gaussian noise array to the given image.

Example 2: Add Gaussian Noise signal to the Image

Example 2: Add Gaussian Noise signal to the Image

```
def img_noise(img_file):
    # extract the base file name
    file_path = img_file[9:-4]
    print('Image File: ', file_path)

    # open the given file
    open_img = Image.open(img_file)
    print('Image size:', np.shape(open_img))

    # convert to numpy array
    np_img_arr = np.zeros(np.shape(open_img))
    np_img_arr = np.array(open_img)

    # Convert img_arr values between [0, 1]
    np_img_arr = np_img_arr / 255

    # Generate normal random noise
    mu, sigma = 0, 0.1 # mean and standard deviation
    normal_random_noise = np.random.normal(mu, sigma, np.shape(np_img_arr))

    # Add noise to the image
    noise_image = np_img_arr + normal_random_noise

    # convert back to integers by multiplying with 255 (add code) and cast it as "uint8"
    noise_image = (noise_image*255).astype(np.uint8)
    #print(type(noise_image))

    # Save new image to a new folder with new image name
    matplotlib.image.imsave('../image_noise_para/' + file_path + "_noise.jpg", noise_image)

    print('The noise has been addded on this image!!!')
```

Difference between Pool.map and Process

- Accept single argument vs. multiple arguments
- Multiple Tasks vs. Single Task

```python
n_cores = 16
print('\nNumber of core is requested: ',  n_cores, '\n')

start_time = time.perf_counter()

with mp.Pool(processes=n_cores) as pool:
    pool.map(img_noise, file_list)

end_time = time.perf_counter()
print("Elpased Time: ", end_time - start_time)
```

```python
n_cores = 16
print('\nNumber of core is requested: ',  n_cores, '\n')

start_time = time.perf_counter()

pro_id = []

for idx in range(n_cores):
    p = mp.Process(target=img_noise, args=(file_list[idx],))
    p.start()
    pro_id.append(p)

for proc in pro_id:
    proc.join()

end_time = time.perf_counter()
print("Elpased Time: ", end_time - start_time)
```

Difference between Pool.map and Process

```
Number of core is requested:  8

Image File:  animal_beautiful_big
Image File:  ESO_Very_Large_Telescope
Image File:  time_on_big_ben_192639
Image File:  maple_big_tree_red
Image File:  big_bend_texas_deer
Image File:  Chess_Large
Image File:  big_brother_is_watching_you_196554
Image File:  architecture_big_ben
Image size: (3048, 3640, 3)
Image size: (4971, 3314, 3)
Image size: (5000, 3333, 3)
Image size: (2832, 4256, 3)
Image size: (2848, 4288, 3)
Image size: (3456, 4608, 3)
Image size: (3333, 5000, 3)
Image size: (5370, 3580, 3)
The noise has been addded on this image!!!
Image File:  animal_big_carnivore
Image size: (5000, 3334, 3)
The noise has been addded on this image!!!
Image File:  animal_beak_big
Image size: (3655, 3004, 3)
The noise has been addded on this image!!!
Image File:  Sample-jpg-image-5mb
The noise has been addded on this image!!!
Image File:  big_city_facades_view
The noise has been addded on this image!!!
Image File:  big_board_check
The noise has been addded on this image!!!
Image File:  animal_big_black
```

```
Image File:  animal_big_black
Image size: (3264, 2448, 3)
Image size: (3744, 5616, 3)
Image size: (4660, 3106, 3)
Image size: (6761, 5072, 3)
The noise has been addded on this image!!!
Image File:  rocks_amp_waves_big_sur_2_563664
The noise has been addded on this image!!!
Image File:  africa_animal_big
The noise has been addded on this image!!!
Image size: (3456, 5184, 3)
Image size: (3333, 5000, 3)
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
Elpased Time:  13.984213357791305
```

```
Number of core is requested:  8

../image/animal_beautiful_big.jpg
Image File:  animal_beautiful_big
../image/ESO_Very_Large_Telescope.jpg
Image File:  ESO_Very_Large_Telescope
../image/time_on_big_ben_192639.jpg
Image File:  time_on_big_ben_192639
../image/maple_big_tree_red.jpg
Image File:  maple_big_tree_red
../image/big_bend_texas_deer.jpg
Image File:  big_bend_texas_deer
../image/Chess_Large.jpg
Image File:  Chess_Large
../image/big_brother_is_watching_you_196554.jpg
Image File:  big_brother_is_watching_you_196554
../image/architecture_big_ben.jpg
Image File:  architecture_big_ben
Image size: (3048, 3640, 3)
Image size: (4971, 3314, 3)
Image size: (5000, 3333, 3)
Image size: (2848, 4288, 3)
Image size: (3333, 5000, 3)
Image size: (5370, 3580, 3)
Image size: (2832, 4256, 3)
Image size: (3456, 4608, 3)
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
The noise has been addded on this image!!!
Elpased Time:  5.226678730919957
```

What is MPI?

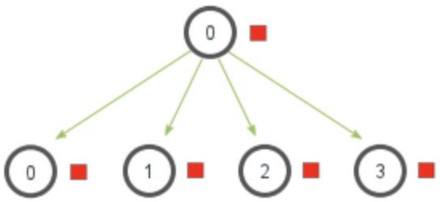
- Message Passing Interface (MPI) primarily addresses the message-passing parallel programming model. The data is moved from one process's address space to another through cooperative operations on each process.
- Compare multiprocessing and mpi4py modules
  - Shared Memory: Multiple processes share a single memory space with full read/write ability
  - Distributed Memory: Each process receives a copy of the memory space when they are first initialized. Communication is handled through message passing.
- Command for running MPI Python script

  mpirun -np 8 python example.py

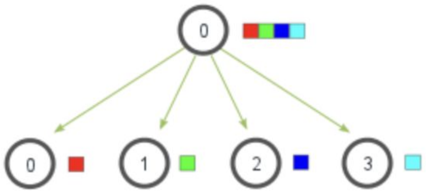
- Ref: https://rabernat.github.io/research_computing/parallel-programming-with-mpi-for-python.html

MPI Collective Communication

- Broadcasting: Broadcasting takes a variable and sends an exact copy to all processes on a communicator.
  - comm.bcast(send_data, root=0)

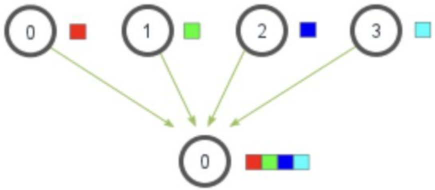


- Scattering: Scatter takes an array and distributes contiguous sections of it across the ranks of a communicator.
  - comm.scatter(send_data, root=0)

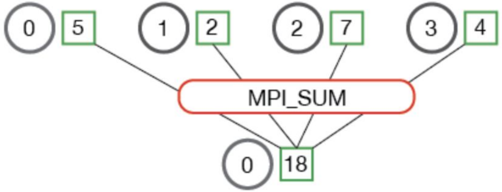


MPI Collective Communication

- Gathering: Gather takes subsets of an array that are distributed across the ranks, and gathers them back into the full array.
  - comm.gather(obj, root=0)



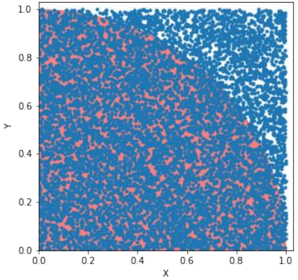
- Reduction: Reduce operation takes values from an array on each process and reduces them to a single result on the root process.
  - comm.reduce(recv_data, op=, root=0)



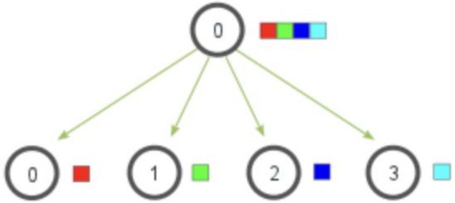
Reduce Operation

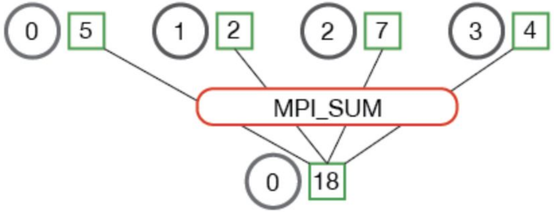
- MPI.MAX: Returns the maximum element.
- MPI.MIN: Returns the minimum element.
- MPI.SUM: Sums the elements.
- MI.PROUD: Multiplies all elements.
- MPI.LAND: Performs a logical AND across the elements.
- MPI.LOR: Performs a logical OR across the elements.
- MPI.BAND: Performs a bitwise AND across the bits of the elements.
- MPI.BOR: Performs a bitwise OR across the bits of the elements.

MC PI Calculation ~ MPI Scatter and Reduce




```python
from mpi4py import MPI
import numpy as np
import sys
import os

def pi_mc(count, num_gen, seed):
    np.random.seed(seed)

    for i in range(num_gen):
        x_val = np.random.random_sample()
        y_val = np.random.random_sample()

        radius = x_val*x_val + y_val*y_val

        if radius < 1:
            count = count + 1

    print(f'This is Process: {rank}, Rand Seed is: {seed}, Count is
    return count
```






```python
    comm = MPI.COMM_WORLD
    size = comm.Get_size()
    rank = comm.Get_rank()
    my_name = MPI.Get_processor_name()
    PID = os.getpid()

    Master = 0
    seed = None
    num_gen = 10000000
    partial = int(num_gen / size)
    #print('Data Type: ', type(partial), 'Partial: ', partial)
    cnt = 0

    if rank == Master:
        seed = np.arange(size, dtype = 'i')
        print('Total No. of Sampling: ', num_gen)
        print('We are scattering the Random Seed:', seed, ' to each Rank.')

    start_time = MPI.Wtime()
    seed_s = comm.scatter(seed, root=Master)

    # function from here
    cnt = pi_mc(cnt, partial, seed_s)

    print("Hi, My PID is: ", PID, ', Hello World!!!')
    print('Rank is: ' , rank, ' and seed = ', seed_s)

    cnt_g = comm.gather(cnt, root=Master)

    end_time = MPI.Wtime()
    elapsed_time = end_time - start_time

    tot = comm.reduce(cnt, op=MPI.SUM, root=Master)

    #print('SEED Gather: ', seed_g)

    if rank == Master:
        print('seed:', seed)
        print('Count Gather:', cnt_g)
        print('PI: ', 4*tot/num_gen)
        print('Elapsed Time: ', elapsed_time)
```

MPI Point to Point Communication

- For our previous MC example, we used the simple communication routines, comm_scatter() and comm_Reduce().
- But you can send any piece of data from any process to any other process, using comm_send() and comm_receive().
- Basically, send and receive some numbers from one program to another.
- If you understand the Send and Receive commands, you should be able to create pretty much any parallel program you need in MPI.
- comm.send(obj, dest, tag=1)
    - “tag” can be used as a filter
- comm.recv(source, tag=1)

Task Parallelism

75 Exams per everyone

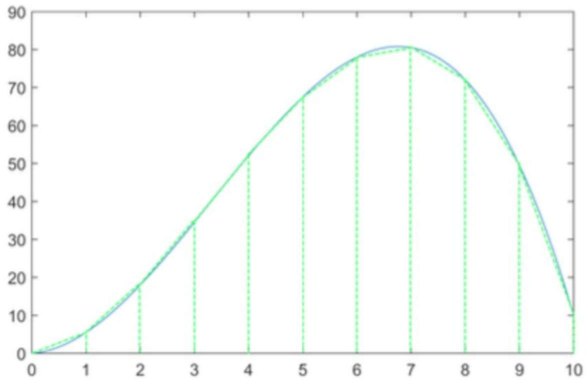
Question 5-8

Question 9-12

Question 13-16

Question 1-4

Example 3: Numerical Integration



$$f(x) = x + 5x^2 - 0.5x^3$$

Integral $\int_a^b f(x)dx$ can be approximately computed using the trapezoid method, which is illustrated in figure. We divided the function into n subinterval with the node $\{x_0, x_1, \dots, x_n\}$ where $x_0 = a$, and $x_n = b$. The width is $\Delta x = \frac{b-a}{n}$. The area of the trapezoidal over the interval $[x_i, x_i + 1]$ is $A_i = \frac{\Delta x}{2}(f(x_i) + f(x_{i+1}))$. $\int_a^b f(x)dx = \sum_{i=1}^{n-1} A_i$.

Write a MPI program to integral $f(x) = x + 5x^2 - 0.5x^3$ (shown in the picture) over the interval $[0, 10]$ using trapezoidal method.

In this program, the interval is evenly divided to N_p subintervals. N_p is the number of processes. The process $i(i = 0, 1, \dots, N_{p-1})$ is in charge of the interval $[x_i, x_{i+1}]$ and computes the area A_i.

The process i only evaluates the function $f(x)$ at x_i and gets $f(x_i + 1)$ from the process $i + 1$.

This algorithm indicates that the processes send data to each other in a ring-like fashion, except for the last process which calculates both $f(xN_{p-1})$ and $f(xN_p)$.

Numerical Integration ~ MPI Send and Receive

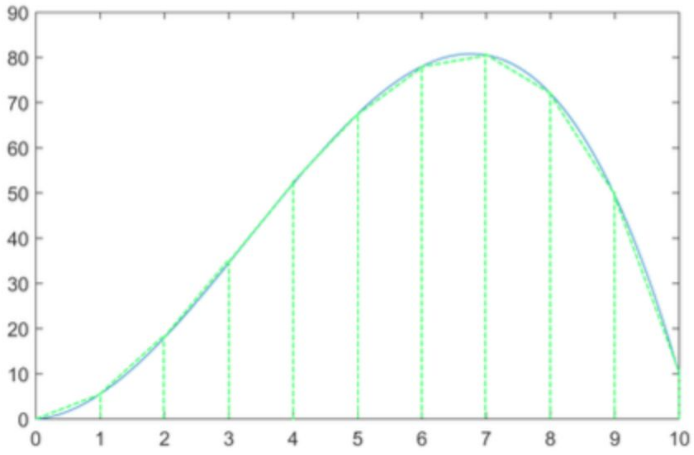
```python
def func(x):
    f = x + 5.0*x**x - 0.5*x**x**x
    return f
```

```python
    a = 0
    b = 10

    comm = MPI.COMM_WORLD
    size = comm.Get_size()                  # No. of Processors
    rank = comm.Get_rank()                  # Process ID
    my_name = MPI.Get_processor_name()
    PID = os.getpid()

    # Define Master as 0
    Master = 0

    h = (b-a) / size
    xi = rank * h
    f_xi = func(xi)

    SOURCE = rank + 1
    DESTINATION = rank - 1

    print(f'####################### My Rank ID is: {rank} ########################')
    print("Process Name: ", my_name)
    print("Hi, My PID is: ", PID, ', Hello World!!!')
    print(f'Calculate the (x_i) =  {xi} and f(x_i) = {f_xi}')
```

```python
    if rank != Master:
        comm.send(f_xi, dest=DESTINATION, tag=1)
    print(f'PASS f(xi) = {f_xi} to Rank ID: {DESTINATION}.')

    if rank != (size-1):
        f_xi1 = comm.recv(source=SOURCE, tag=1)
        #print('f_xi = ', f_xi)
        print(f'Get the f(x_i+1) = {f_xi1} from Source ID: {SOURCE}')
        subArea = 0.5 * h * (f_xi + f_xi1)
        print(f'Sub Area is: {subArea}')
    else:
        xi1 = size*h
        f_xi1 = func(xi1)
        print(f'Calculate the (x_i+1) = {xi1} and f(x_i+1) = {f_xi1}')
        subArea = 0.5 * h * (f_xi + f_xi1)
        print(f'Sub Area is: {subArea}')

    tot = comm.reduce(subArea, op=MPI.SUM, root=Master)

    if rank == Master:
        print('\n###########################')
        print('Total No. of Processor is: ', size)
        print('No. of Processor is: ', size)
        print(f'This is Master (Rank ID is: {rank})')
        print(f'Source is: {SOURCE}, NO DESTINATION!!!')
        print('Final Integral Result is: ', tot)
        print('###########################\n')
```

Numerical Integration ~ MPI Send and Receive

```
######################## My Rank ID is: 7 ########################
Process Name:  pod-login1.podcluster
Hi, My PID is:  228091 , Hello World!!!
Calculate the (x_i) =  7.0 and f(x_i) = 80.5
PASS f(xi) = 80.5 to Rank ID: 6.
Get the f(x_i+1) = 72.0 from Source ID: 8
Sub Area is: 76.25
######################## My Rank ID is: 0 ########################
Process Name:  pod-login1.podcluster
Hi, My PID is:  228084 , Hello World!!!
Calculate the (x_i) =  0.0 and f(x_i) = 0.0
PASS f(xi) = 0.0 to Rank ID: -1.
Get the f(x_i+1) = 5.5 from Source ID: 1
Sub Area is: 2.75

###########################
Total No. of Processor is:  10
No. of Processor is:  10
This is Master (Rank ID is: 0)
Source is: 1, NO DESTINATION!!!
Final Integral Result is:  462.5
###########################

######################## My Rank ID is: 1 ########################
Process Name:  pod-login1.podcluster
Hi, My PID is:  228085 , Hello World!!!
Calculate the (x_i) =  1.0 and f(x_i) = 5.5
PASS f(xi) = 5.5 to Rank ID: 0.
Get the f(x_i+1) = 18.0 from Source ID: 2
Sub Area is: 11.75
```



Testing Parallel Code on the Cluster

- Perform a small test on your computer first
- Test your small Parallel Code on the ***short*** partition or your local machine
- Submit your slurm script job to the queue

Conclusion

- In today's workshop, I hope it helps you to learn some ***concepts*** of parallel Python programming.
- What is the difference between Process class and Pool class? Which one is suitable for you?
- You can see that the mpi4py module requires more programming effort than the multiprocess module, but it is much more powerful.
- Parallel programming is a broad with numerous possibilities for learning. The workshop **JUST introduces** a few parallel modules available in Python for simple parallel programming.
- Find which parallel module suits your computational research project and dig into it.

Questions and Thought

- What else content should we cover?
- Other ideas for a workshop?

- More Information:

  https://csc.cnsi.ucsb.edu/