Fahad Fiaz – (303141) – G2

System Info:

Processor	i7-5500U , 2.40GHz
Cores	4
Operating system	Windows 64 Bit
Ram	8GB
Programming Language	Python 3.7.7

Exercise 1: Point to Point Communication?

Part a) NsendAll method implementation:

Steps:

1. Consider 1st process as master node and other as worker node.
2. Master process will create array and send that to other worker processes.
3. Other processes received array.

Time Taken:

Size P	10^3	10^5	10^7
2	0.0009	0.0009	0.34
4	0.01	0.01	0.77
6	0.009	0.08	0.83
8	0.01	0.02	0.87
16	0.30	0.13	1.58
32	0.30	0.28	4.01

Part B) EsendAll method implementation:

Steps:

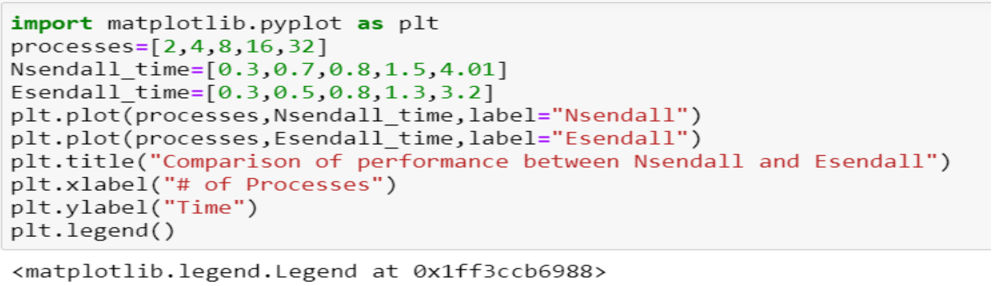
1. Consider 1st process as master node and other as worker node.
2. Master process will create array and send it to slave worker with Rank1 and slave worker with Rank 2. It also checks that these workers exist before passing the array.
3. Slave process received array from process by calculating process id of process which sent array to it by formula “*int*((*rank* – 1)/2)” .
4. It then send array to two more processes if these processes exist.

Time Taken:

Size P	10^3	10^5	10^7
2	0.0009	0.005	0.39
4	0.02	0.031	0.53
6	0.004	0.012	0.67
8	0.086	0.06	0.84
16	0.293	0.101	1.3
32	0.793	0.18	3.2

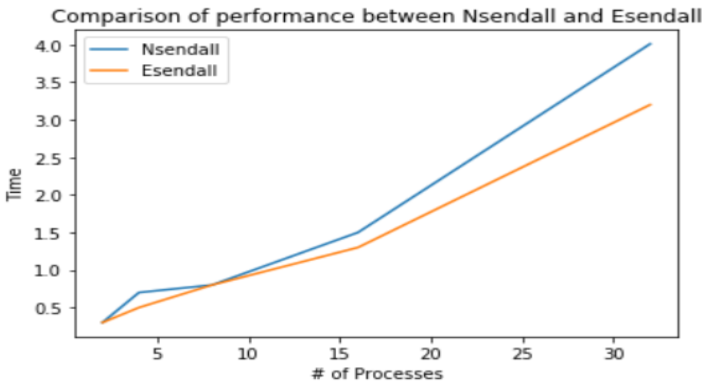
Comparison of performance of the two implementations:

Array size: 10**7



```python
import matplotlib.pyplot as plt
processes=[2,4,8,16,32]
Nsendall_time=[0.3,0.7,0.8,1.5,4.01]
Esendall_time=[0.3,0.5,0.8,1.3,3.2]
plt.plot(processes,Nsendall_time,label="Nsendall")
plt.plot(processes,Esendall_time,label="Esendall")
plt.title("Comparison of performance between Nsendall and Esendall")
plt.xlabel("# of Processes")
plt.ylabel("Time")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1ff3ccb6988>
```



All Outputs in the Question1.html file

Exercise 2: Collective Communication?

Steps:

1. Consider 1st process as master node and other as worker node.
2. Master process will read image and partition it by dividing image matrix to number of processes available and save these chunks in a list.
3. This list is scattered by root process to other slave processes using scatter function.
4. Slave processes find frequency count of pixels in their chunk.
5. All slave processes then send results to a root process using reduce function which perform the aggregate function sum on the buffer data and send it to root.

Grey Scale Image:

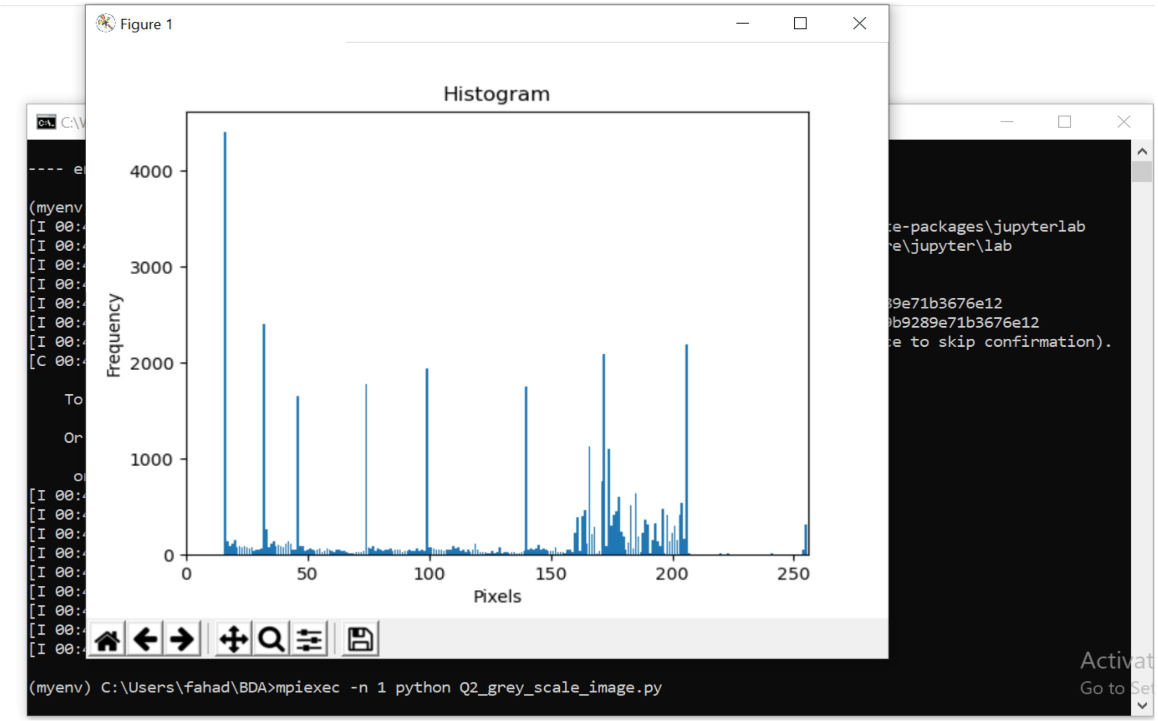
Image with pixels 2048 * 2048 was taking a lot of time because of large size so for grey scale image I reduced the image to 200 * 200 pixels and calculated the run time. But since my code works fine for reduced image then, it should work fine for images with larger size.

Time Taken:

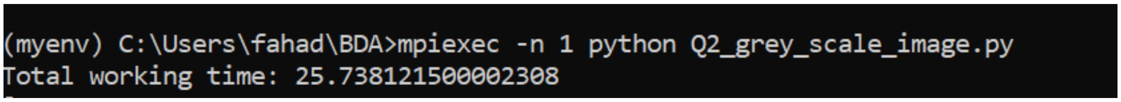
Processes	Time
1	25.7
2	15.7
3	14.8
4	11.8

Above table shows runtime decreases when we increase number of processes.

P = 1:

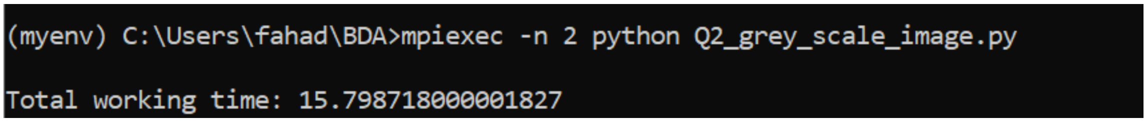


```
---- e
(myenv                                                              e-packages\jupyterlab
[I 00:                                                              e\jupyter\lab
[I 00:
[I 00:                                                              39e71b3676e12
[I 00:                                                              0b9289e71b3676e12
[I 00:                                                              e to skip confirmation).
[I 00:
[I 00:
[C 00:
    To
    Or
     o
[I 00:
[I 00:
[I 00:
[I 00:
[I 00:
[I 00:
[I 00:
[I 00:
[I 00:

(myenv) C:\Users\fahad\BDA>mpiexec -n 1 python Q2_grey_scale_image.py
```



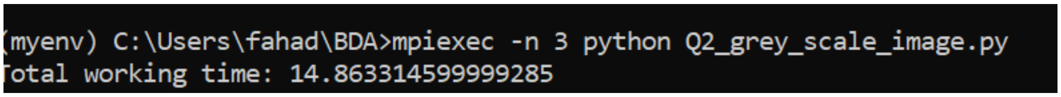
```
(myenv) C:\Users\fahad\BDA>mpiexec -n 1 python Q2_grey_scale_image.py
Total working time: 25.738121500002308
```

P = 2:



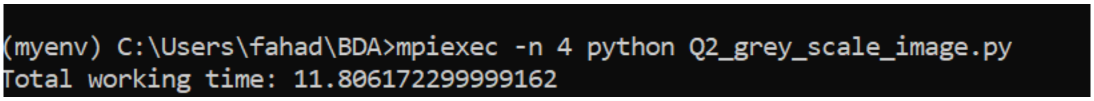
```
(myenv) C:\Users\fahad\BDA>mpiexec -n 2 python Q2_grey_scale_image.py

Total working time: 15.798718000001827
```

P = 3:



```
(myenv) C:\Users\fahad\BDA>mpiexec -n 3 python Q2_grey_scale_image.py
Total working time: 14.863314599999285
```

P = 4:



```
(myenv) C:\Users\fahad\BDA>mpiexec -n 4 python Q2_grey_scale_image.py
Total working time: 11.806172299999162
```

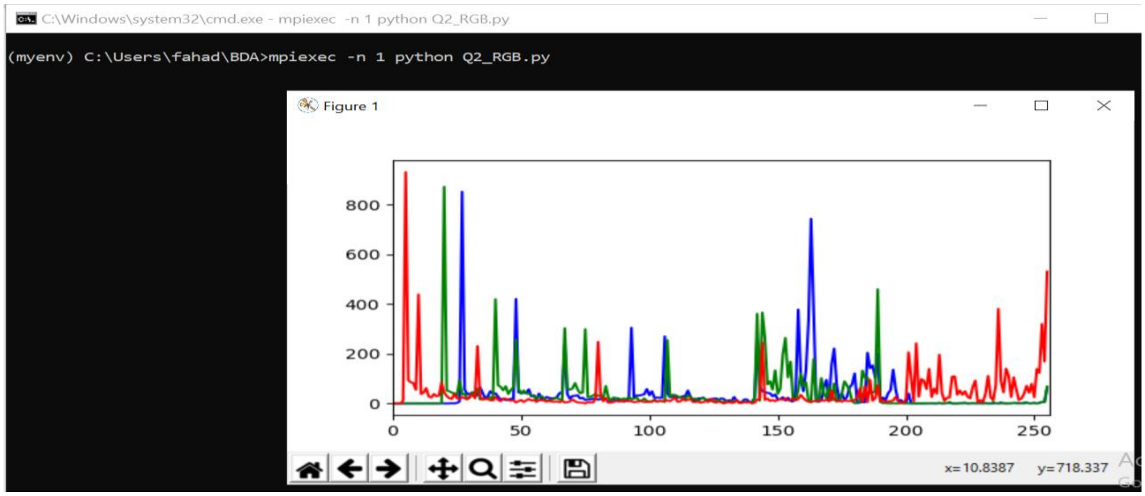
RGB Image:

Image with pixels 2048 * 2048 was taking a lot of time because of large size so I reduced the image to **100 * 100** pixels and calculated the run time. But since my code works fine for reduced image then, it should work fine for images with larger size.

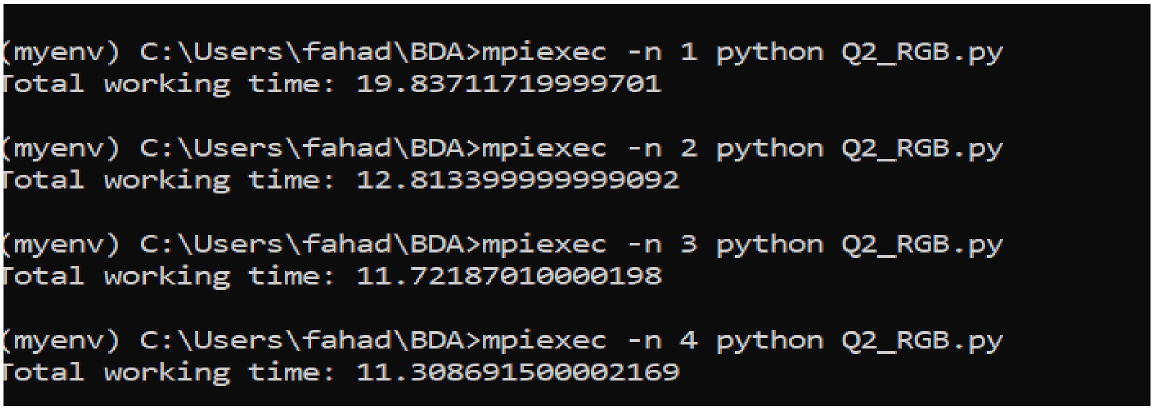
Time Taken:

Processes	Time
1	19.83
2	12.81
3	11.72
4	11.30

Above table shows runtime decreases when we increase number of processes.



```
C:\Windows\system32\cmd.exe - mpiexec  -n 1 python Q2_RGB.py
(myenv) C:\Users\fahad\BDA>mpiexec -n 1 python Q2_RGB.py
```



```
(myenv) C:\Users\fahad\BDA>mpiexec -n 1 python Q2_RGB.py
Total working time: 19.83711719999701

(myenv) C:\Users\fahad\BDA>mpiexec -n 2 python Q2_RGB.py
Total working time: 12.813399999999092

(myenv) C:\Users\fahad\BDA>mpiexec -n 3 python Q2_RGB.py
Total working time: 11.72187010000198

(myenv) C:\Users\fahad\BDA>mpiexec -n 4 python Q2_RGB.py
Total working time: 11.308691500002169
```