Lab Course: Distributed Data Analytics
Exercise Sheet 1 – Group 2 (Monday)
Raaghav Radhakrishnan (246097)

Exercise 0: Explain your system:

The hardware and software setup are listed as follows:

Hardware setup:

Processor	Intel® Core ™ i5-5200 CPU @ 2.20GHz
RAM	12 GB
OS	Windows 8
Cores	2

Software setup:
Python: 3.5.3

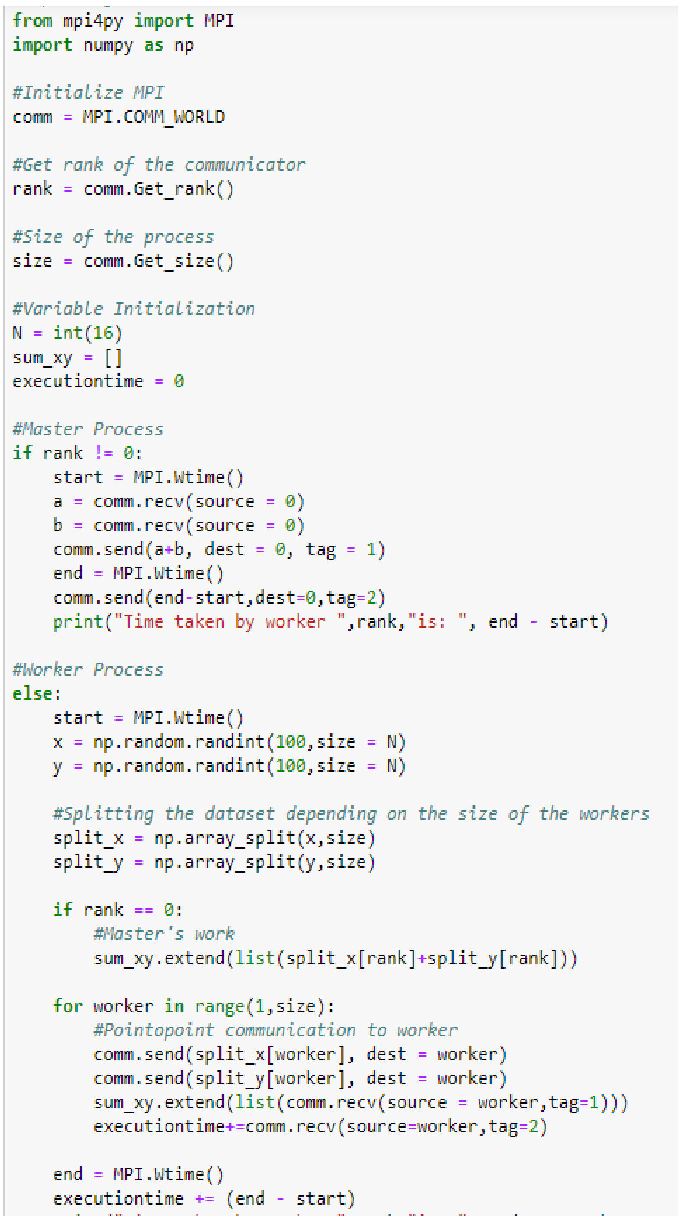
Exercise 1: Basic Parallel Vectors Operations with MPI:

In this exercise, it is required to add two given vectors and store the result in a third vector using MPI. The parallelization strategy used is similar to threading but in form of message passing i.e., Point to Point Communication. Here, I've chose rank 0 as master who assigns work to other workers. Also, the master performs a part of the work along with the workers.

Steps:

1. The data is divided based on the size of the process provided in the execution command
2. When the worker is master, it performs the vector addition operation and also sends copies of the data to the workers and receives the same from them

3. Once the vectors are received through communication, the master appends them to the final vector and the corresponding time are calculated and tabulated below:



```
from mpi4py import MPI
import numpy as np

#Initialize MPI
comm = MPI.COMM_WORLD

#Get rank of the communicator
rank = comm.Get_rank()

#Size of the process
size = comm.Get_size()

#Variable Initialization
N = int(16)
sum_xy = []
executiontime = 0

#Master Process
if rank != 0:
    start = MPI.Wtime()
    a = comm.recv(source = 0)
    b = comm.recv(source = 0)
    comm.send(a+b, dest = 0, tag = 1)
    end = MPI.Wtime()
    comm.send(end-start,dest=0,tag=2)
    print("Time taken by worker ",rank,"is: ", end - start)

#Worker Process
else:
    start = MPI.Wtime()
    x = np.random.randint(100,size = N)
    y = np.random.randint(100,size = N)

    #Splitting the dataset depending on the size of the workers
    split_x = np.array_split(x,size)
    split_y = np.array_split(y,size)

    if rank == 0:
        #Master's work
        sum_xy.extend(list(split_x[rank]+split_y[rank]))

    for worker in range(1,size):
        #Pointopoint communication to worker
        comm.send(split_x[worker], dest = worker)
        comm.send(split_y[worker], dest = worker)
        sum_xy.extend(list(comm.recv(source = worker,tag=1)))
        executiontime+=comm.recv(source=worker,tag=2)

    end = MPI.Wtime()
    executiontime += (end - start)
```

N: 10^4	Time		Total Time
Process	0	1	
0	0.0105		0.0105
1	0.003	0.003	0.006

N: 10^5	Time		Total Time
Process	0	1	
0	0.104		0.104
1	0.0411	0.0434	0.0845

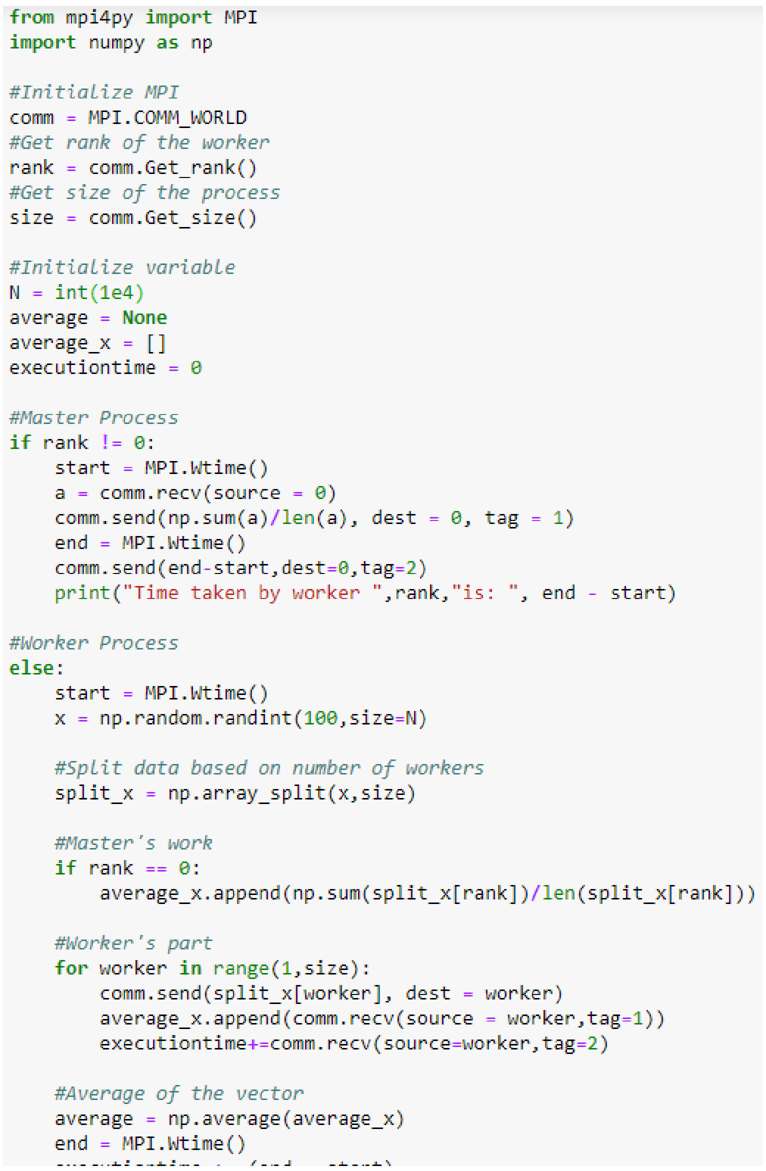
N: 10^6	Time		Total Time
Process	0	1	
0	0.864		0.864
1	0.345	0.478	0.823

N: 10^7	Time		Total Time
Process	0	1	
0	9.026		9.026
1	4.583	3.99	8.573

Verifying Results:

```
X: [94 95 57  7  2 30 77 34 32 79  5 22 60  9 48  7]
Y: [56 23 70  6 68 91 52 65 42 65 26 90 55 64 53  2]
Sum: [150, 118, 127, 13, 70, 121, 129, 99, 74, 144, 31, 112, 115, 73, 101, 9]
```

The above mentioned steps are followed for the finding the average of the vector. Instead of adding the two vectors, this program takes a vector as input, splits them between the cores and finds separate average which in the end is averaged to get the vector average.



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

#Initialize MPI
comm = MPI.COMM_WORLD
#Get rank of the worker
rank = comm.Get_rank()
#Get size of the process
size = comm.Get_size()

#Initialize variable
N = int(1e4)
average = None
average_x = []
executiontime = 0

#Master Process
if rank != 0:
    start = MPI.Wtime()
    a = comm.recv(source = 0)
    comm.send(np.sum(a)/len(a), dest = 0, tag = 1)
    end = MPI.Wtime()
    comm.send(end-start,dest=0,tag=2)
    print("Time taken by worker ",rank,"is: ", end - start)

#Worker Process
else:
    start = MPI.Wtime()
    x = np.random.randint(100,size=N)

    #Split data based on number of workers
    split_x = np.array_split(x,size)

    #Master's work
    if rank == 0:
        average_x.append(np.sum(split_x[rank])/len(split_x[rank]))

    #Worker's part
    for worker in range(1,size):
        comm.send(split_x[worker], dest = worker)
        average_x.append(comm.recv(source = worker,tag=1))
        executiontime+=comm.recv(source=worker,tag=2)

    #Average of the vector
    average = np.average(average_x)
    end = MPI.Wtime()
    executiontime += (end - start)
```

N: 10^2	Time		Total Time
Process	0	1	
0	0.0188		0.0188
1	0.00069	0.0043	0.00499

N: 10^3	Time		Total Time
Process	0	1	
0	0.0185		0.0185
1	0.0007	0.005	0.0057

N: 10^4	Time		Total Time
Process	0	1	
0	0.0128		0.0128
1	0.00105	0.007	0.0081

Question 2: Parallel Matrix Vector Multiplication using MPI

In this exercise, it is required to add multiply the given matrix with a vector and store the result in a third vector using MPI. The parallelization strategy used is similar to threading but in form of message passing i.e., Point to Point Communication. Here, I've chose rank 0 as master who assigns work to other workers. Also, the master performs a part of the work along with the workers.

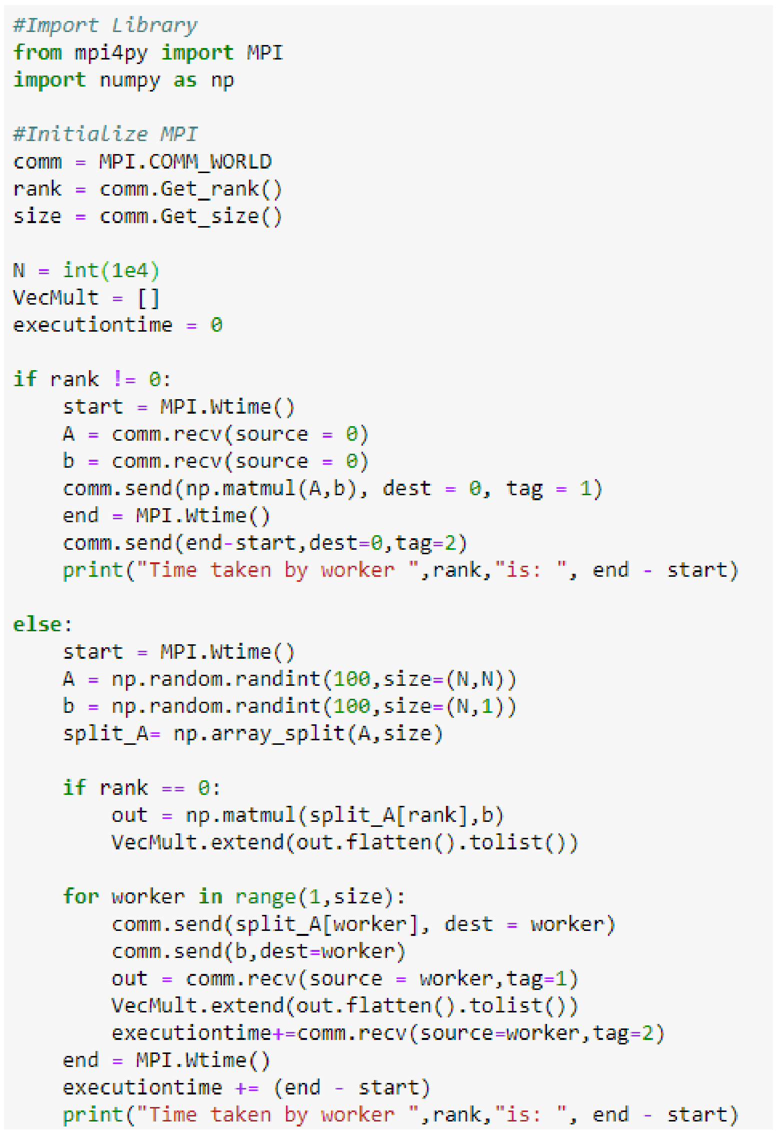
Steps:

1. The data is divided based on the size of the process provided in the execution command
2. When the worker is master, it performs the matrix vector multiplication and also sends copies of the data to the workers and receives the same from them
3. Once the multiplied vectors are received through communication, the master appends them to the final vector and the corresponding time are calculated and tabulated below:

N: 10^2	Time		Total Time
Process	0	1	
0	0.0145		0.0188
1	0.001	0.0059	0.0069

N: 10^3	Time		Total Time
Process	0	1	
0	0.226		0.226
1	0.092	0.111	0.202

N: 10^4	Time		Total Time
Process	0	1	
0	14.61		0.0128
1	6.37	5.221	11.581



```python
#Import Library
from mpi4py import MPI
import numpy as np

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

N = int(1e4)
VecMult = []
executiontime = 0

if rank != 0:
    start = MPI.Wtime()
    A = comm.recv(source = 0)
    b = comm.recv(source = 0)
    comm.send(np.matmul(A,b), dest = 0, tag = 1)
    end = MPI.Wtime()
    comm.send(end-start,dest=0,tag=2)
    print("Time taken by worker ",rank,"is: ", end - start)

else:
    start = MPI.Wtime()
    A = np.random.randint(100,size=(N,N))
    b = np.random.randint(100,size=(N,1))
    split_A= np.array_split(A,size)

    if rank == 0:
        out = np.matmul(split_A[rank],b)
        VecMult.extend(out.flatten().tolist())

    for worker in range(1,size):
        comm.send(split_A[worker], dest = worker)
        comm.send(b,dest=worker)
        out = comm.recv(source = worker,tag=1)
        VecMult.extend(out.flatten().tolist())
        executiontime+=comm.recv(source=worker,tag=2)
    end = MPI.Wtime()
    executiontime += (end - start)
    print("Time taken by worker ",rank,"is: ", end - start)
```

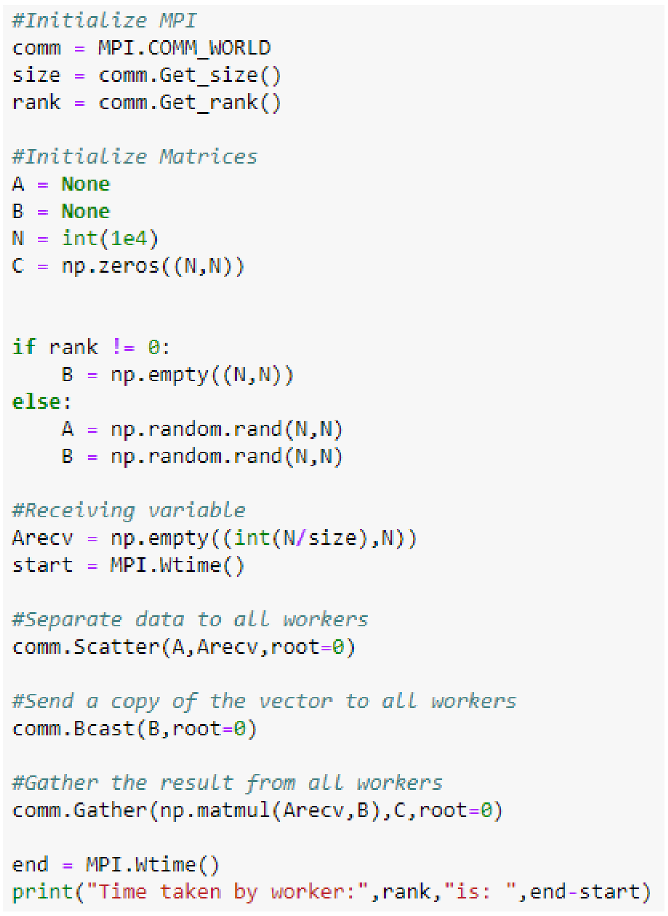
Exercise 3: Parallel Matrix Operation using MPI:

In this exercise, it is required to perform parallel matrix operation on given matrices and store the result in a third matrix using MPI. The parallelization strategy used is similar to threading but in form of message passing i.e., Collective Communication. Here, I've chose rank 0 as master who assigns work to other workers. Also, the master performs a part of the work along with the workers.

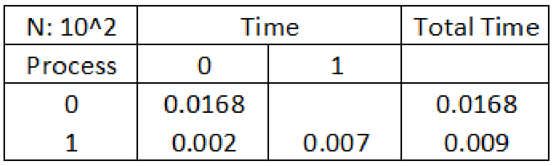
Steps:

1. The data is divided based on the size of the process provided in the execution command
2. When the worker is master, it creates the input matrices and the workers create an empty matrix

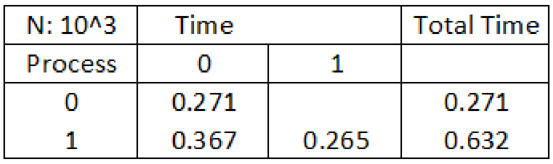
3. Using comm.Scatter, the master splits the data and shares them to the workers
4. Using comm.Bcast, the master sends a copy of the matrix to the workers
5. With the received matrices, the workers perform the parallel matrix operations
6. Once the operation is performed, the results are gathered by the master and stored in the resulting matrix. The code and time taken are as follows:



```
#Initialize MPI
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

#Initialize Matrices
A = None
B = None
N = int(1e4)
C = np.zeros((N,N))


if rank != 0:
    B = np.empty((N,N))
else:
    A = np.random.rand(N,N)
    B = np.random.rand(N,N)

#Receiving variable
Arecv = np.empty((int(N/size),N))
start = MPI.Wtime()

#Separate data to all workers
comm.Scatter(A,Arecv,root=0)

#Send a copy of the vector to all workers
comm.Bcast(B,root=0)

#Gather the result from all workers
comm.Gather(np.matmul(Arecv,B),C,root=0)

end = MPI.Wtime()
print("Time taken by worker:",rank,"is: ",end-start)
```



N: 10^2	Time		Total Time
Process	0	1	
0	0.0168		0.0168
1	0.002	0.007	0.009



N: 10^3	Time		Total Time
Process	0	1	
0	0.271		0.271
1	0.367	0.265	0.632