Exercise 2

Farjad Ahmed – 1747371

```
""" H/W path Device Class Description
========================================================
===
system HP ProBook 440 G5 (1MJ81AV)
/0 bus 837B
/0/1 memory 64KiB BIOS
/0/7 memory 16GiB System Memory
/0/10 processor Intel(R) Core(TM) i5-8250U CPU @ 1.60GH
/0/100 bridge Xeon E3-1200 v6/7th Gen Core Processor
/0/100/2 /dev/fb0 display UHD Graphics 620
/0/100/17/0 /dev/sda disk 1TB TOSHIBA MQ04ABF1
/0/100/17/0/1 /dev/sda1 volume 831GiB Windows NTFS volume
/0/100/17/1 /dev/sdb disk 256GB MTFDDAV256MBF-1A
/0/100/17/1/1 volume 511MiB Windows FAT volume
"""
```

The programs are run on the maximum functioning parameters, and dimension. For my system, 10**2, 10**3, and 10**4 were maximum and after this memory allocation error was being generated.

I faced difficulty in this assignment hence took help particularly from Areeba Rauf but understood the working and coded myself.

Question 1a:
In this question, the generate_mat(), generates a n dimension matrices A and B. Root 0 is the main node that uses this command and generates data. The matrices are split using array_split. After this a loop iterates over all the size of the processor, that is the number of processors and sends to each of the other processors. The root node does the first calculation in line 32 and adds it to the resultant array that is ‘output’.
Moving further, an extra conditional is placed just for my own understanding. It runs a loop for range ‘size’ and receives from all other processors and appends it to the resultant.

Furthermore, the else conditional caters to all other ranks other than the root rank (0). It receives the chunks of A and B matrices respectively and does the summation and send back to the root node.

It is observed that for higher number of processors the time to process is increasing.

n	Processors	time
10**2	2	execution time 0.001381037 execution time 0.001331146
10**2	3	execution time 0.000263412 execution time 0.00743406 execution time 0.002392785
10**2	4	execution time 0.000448132 execution time 0.011149278 execution time 0.002247547

		execution time 0.018990347
10**3	2	execution time 0.053822596 execution time 0.058812909
10**3	3	execution time 0.061056454 execution time 0.047622719 execution time 0.055628594
10**3	4	execution time 0.059704333 execution time 0.05268233 execution time 0.055085109 execution time 0.047171207
10**4	2	execution time 3.533837848 execution time 3.836336794
10**4	3	execution time 3.390754578 execution time 3.678320918 execution time 3.940160045
10**4	4	execution time 3.7123958 execution time 3.955925614 execution time 4.239708034 execution time 4.517415461

Question 1b:

In this question we follow the similar login as in question 1. However the difference is in the else conditional we take each chunk coming from the root node that is split parts of the vector and find mean for it. This is then sent to the root node which received the array of results of means and then the final mean is computed. This verifies with the actual mean for the vector.

The speed up in the processing time is not significant and

The times are mentioned below:

n	Processors	time
10**2	2	execution time 0.000514851 execution time 0.000478973
10**2	3	execution time 0.00034758 execution time 0.007594076 execution time 0.000767426
10**2	4	execution time 0.010838851 execution time 0.021490725 execution time 0.018856398 execution time 0.00057172
10**3	2	execution time 0.002346986 execution time 0.000195708
10**3	3	execution time 0.000772874 execution time 0.000573477
10**3	4	execution time 0.008270448

		execution time 0.000422851 execution time 0.00097586
10**4	2	execution time 0.000890659 execution time 0.000841728
10**4	3	execution time 0.002161591 execution time 0.009986187 execution time 0.000248974
10**4	4	execution time 0.004355857 execution time 0.000188356 execution time 0.002359464 execution time 0.006301219

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
size = comm.size
rank = comm.rank
root = 0
n=10**4
st = MPI.Wtime()
def generate_vec(n):
    np.random.seed(0) # seed set to have a determinable solutiohn
    matA = np.random.randint(5, size=(n)) # nxn matrix
    return matA
if rank == root:
    result = np.array([]) # create an empty array
    A = generate_vec(n) # generate matrix
    # print('avg should be', np.mean(A)) # print average
    Asplit = np.array_split(A, size) # split matrix A into chunks
    # print(Asplit)
    for i in range(1, size):
        comm.send(Asplit[i], dest=i) # send A to each process
    resultChunks = np.mean(Asplit[0]) # calculate the first resultant
    result = np.append(result, resultChunks) # append result
    for z in range(1, size):
            received_resultChunks = comm.recv(source=z) # receive result from process
            result = np.append(result, received_resultChunks) # append result
    # print('average is',np.mean(result)) # print average


else:
    Achunk = comm.recv(source=root) # receive A from root
    avg = np.mean(Achunk) # calculate average
    comm.send(avg, dest=root) # send average to root
et = MPI.Wtime()
actual_time = et-st
print('execution time', actual_time) # print execution time
```

Question 2:

In this question, the main point to note here is the splitting strategy. The A vector is generated and B matrix is generated. The matrix B is transposed and then vsplit is used to break the matrix into row vectors, by the factor of size i.e the total processors initiated.
Next, A and Bsplit are now sent to all the other processors, while the first result is calculated at the root node and appended to the resultant vector.
Moreover, a loop runs to compile the results, it receives data from all nodes except the root which is itself, and compiles the result into the resultant array.
In the else, the conditional all processors other than the root processors are received and the product for the A vector with Bchunk is calculated, this result is sent back to the root which is explained above.

n	Processors	time
10**2	2	execution time 0.001072892 execution time 0.001024605
10**2	3	Array cannot be divided error. Limitation for this algorithm
10**2	4	execution time 0.000273401 execution time 0.000239441 execution time 0.006349515 execution time 0.035223852
10**3	2	Array cannot be divided error. Limitation for this algorithm
10**3	3	execution time 0.043481569 execution time 0.04352726699999998
10**3	4	execution time 0.028988338 execution time 0.032920386 execution time 0.043380851 execution time 0.028663998
10**4	2	execution time 2.660715582 execution time 2.661513705
10**4	3	Array cannot be divided error. Limitation for this algorithm
10**4	4	execution time 2.981859523 execution time 2.971693618 execution time 3.023419923 execution time 3.025136451

```
def multiplyV(A,B):
    result = np.dot(B,A)
    return result
def generate_mat(n):
    np.random.seed(0) # seed set to have a determinable solutiohn
    matA = np.random.randint(6, size=(n))
    matB = np.random.randint(5, size=(n,n))
    C = np.dot(matA, matB)
    # print('Result should be', C)
    return matA, matB
if rank == root:    # root process
    result = np.array([]) # create an empty array
    out =[] # create an empty array
    A, B = generate_mat(n) # generate matrix
    transpose= np.transpose(B) # transpose matrix
    Bsplit = np.vsplit(transpose, size) # split matrix B into chunks
    C = np.dot(A, B) # calculate matrix C
    for i in range(size): # send A to each process
        if i != root:  # except root
            comm.send(A, i) # send A to each process
            comm.send(Bsplit[i], dest=i)    # send B to each process
    resultChunks = multiplyV(A, Bsplit[0]) # calculate the first resultant
    # print(resultChunks)
    result = np.append(result, resultChunks) # append result
    for z in range(size): # receive result from process
        if z != root: # except root
            received_resultChunks = comm.recv(source=z) # receive result from process
            result = np.append(result, received_resultChunks) # append result
    # print('Output',result)
else:
    Achunk = comm.recv(source=root) # receive A from root
    Bchunk = comm.recv(source=root) # receive B from root
    product = multiplyV(Achunk, Bchunk) # calculate product
    # print(product)
    comm.send(product, dest=root) # send result to root

et = MPI.Wtime()
actual_time = et-st
print('execution time', actual_time) # print execution time
```

Question 3:
I discussed this question with Areeba Rauf and Aqsa Manzoor as I was facing difficulty in solving this.
In this case, we generate a matrix A and matrix B. A is broadcasted to all processors, I.e it is sent as it is, while B is scattered, that means the scatter method break the matrix and send it to other ranks for processing. The product is calculated and the ouput is generated by gathering data from all nodes.

It is 10 mins in submission deadline I cant attach the table.

```
from mpi4py import MPI
import numpy as np
comm=MPI .COMM_WORLD
rank = comm.rank
size= comm.Get_size()
n=10**2
st=MPI.Wtime()

def generate_mat(n):
    np.random.seed(0) # seed set to have a determinable solutioh
    matA = np.random.randint(6, size=(n,n)) # nxn matrix
    matB = np.random.randint(5, size=(n,n)) # nxn matrix
    return matA, matB

if(rank==0):
  A=np.random.randint(10,size=(n,n))
  B=np.random.randint(10,size=(n,n))
  A, B =  generate_mat(n)
  A_Chunks = np.vsplit(A, size)

else:
  A=None
  B=None
  C=None
  A_Chunks=None
recvA= comm.scatter(A_Chunks,root=0)
bcast_matB= comm.bcast(B,root=0)
product=np.dot(recvA,bcast_matB)
output = comm.gather(product , root=0)

# if rank==0:
#     # print('The result is',output)


et = MPI.Wtime()
actual_time = et-st
print('execution time', actual_time) # print execution time
```