Topic: Distributed Computing with Message Passing Interface (MPI)

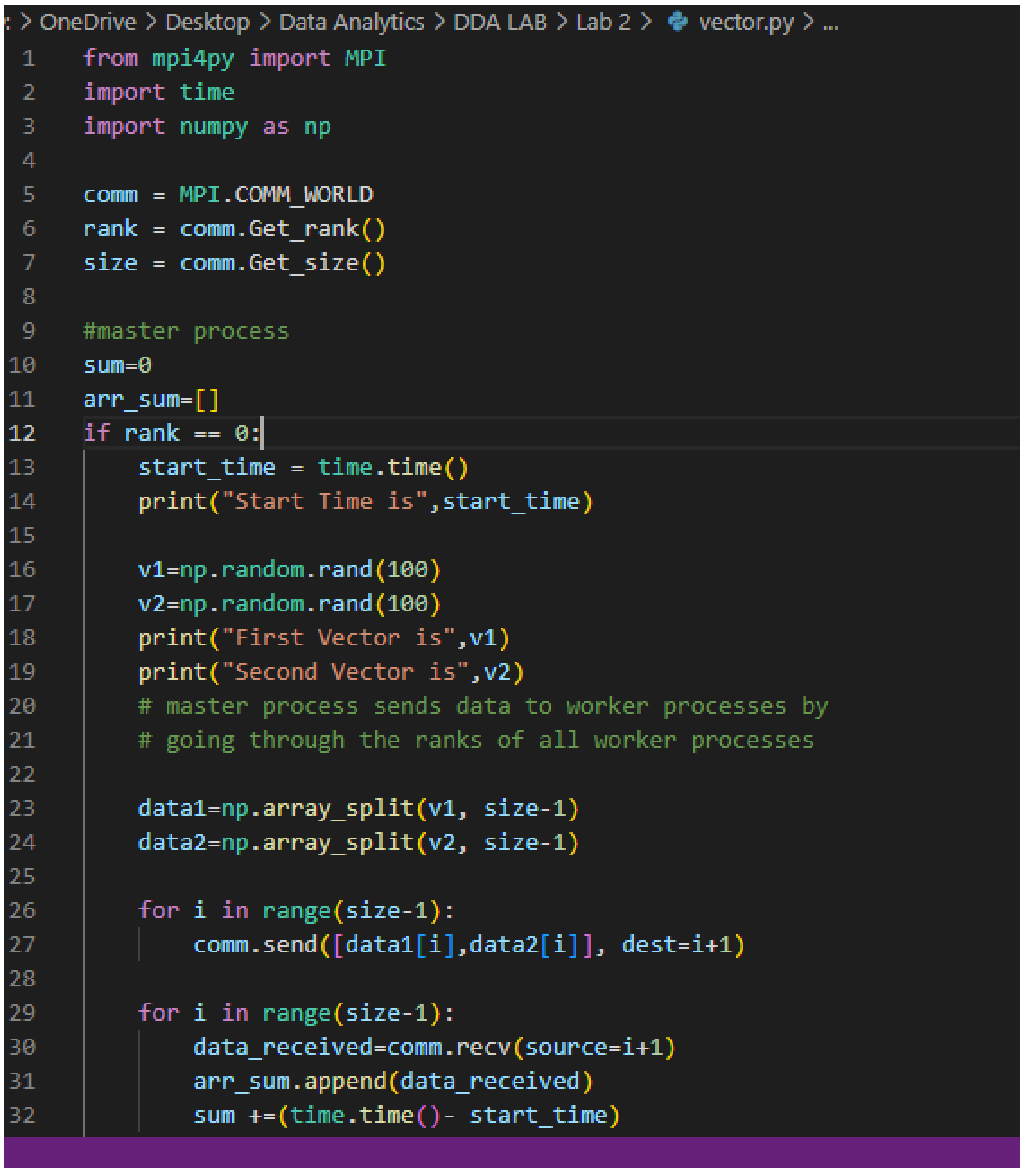
Exercise 0: Explain your system

Exercise 1: Basic Parallel Vector Operations with MPI

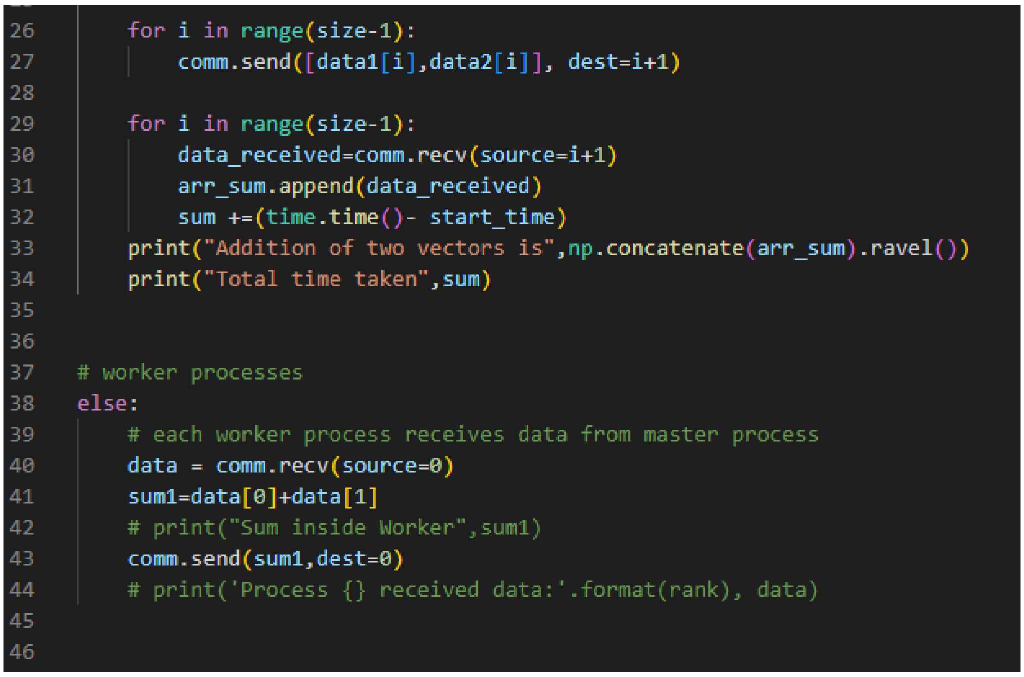
This is done using MPI Point-to-Point Communication. Send and Recv functions.

a) **Add two vectors and store results in a third vector.**
   Parallelization Strategy:
   - When the rank is zero (i.e master process), it splits data based on the size into chunks. Here size refers to the number of workers.
   - This data is sent in the form of an array to the Worker processes.
   - Data is received by the Worker process and sum is performed here. For the sum operation, the data[0] is a chunk from the first vector and data[1] is a chunk from the second vector.
   - Worker process send the result to the master process where it is appended in an array and displayed.

Program:

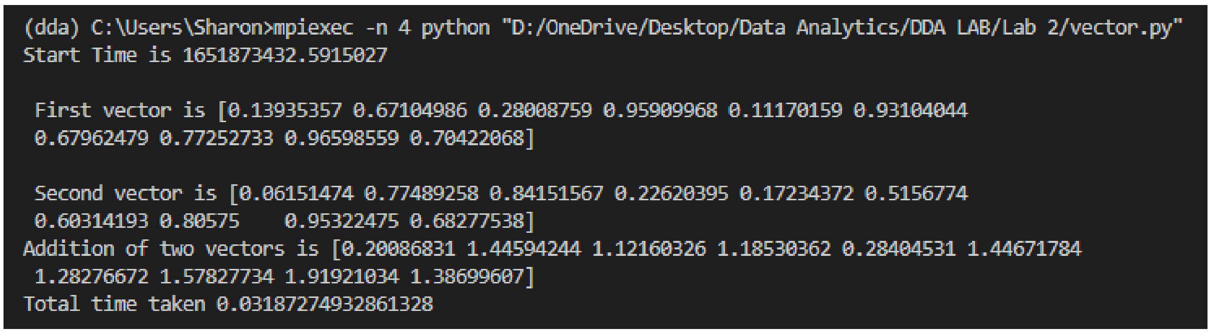

: > OneDrive > Desktop > Data Analytics > DDA LAB > Lab 2 > vector.py > ...

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

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

#master process
sum=0
arr_sum=[]
if rank == 0:
    start_time = time.time()
    print("Start Time is",start_time)

    v1=np.random.rand(100)
    v2=np.random.rand(100)
    print("First Vector is",v1)
    print("Second Vector is",v2)
    # master process sends data to worker processes by
    # going through the ranks of all worker processes

    data1=np.array_split(v1, size-1)
    data2=np.array_split(v2, size-1)

    for i in range(size-1):
        comm.send([data1[i],data2[i]], dest=i+1)

    for i in range(size-1):
        data_received=comm.recv(source=i+1)
        arr_sum.append(data_received)
        sum +=(time.time()- start_time)
```



```python
    for i in range(size-1):
        comm.send([data1[i],data2[i]], dest=i+1)

    for i in range(size-1):
        data_received=comm.recv(source=i+1)
        arr_sum.append(data_received)
        sum +=(time.time()- start_time)
    print("Addition of two vectors is",np.concatenate(arr_sum).ravel())
    print("Total time taken",sum)


# worker processes
else:
    # each worker process receives data from master process
    data = comm.recv(source=0)
    sum1=data[0]+data[1]
    # print("Sum inside Worker",sum1)
    comm.send(sum1,dest=0)
    # print('Process {} received data:'.format(rank), data)
```

Results are shown by taking a smaller array of size 10 to verify them.



```
(dda) C:\Users\Sharon>mpiexec -n 4 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 2/vector.py"
Start Time is 1651873432.5915027

 First vector is [0.13935357 0.67104986 0.28008759 0.95909968 0.11170159 0.93104044
 0.67962479 0.77252733 0.96598559 0.70422068]

 Second vector is [0.06151474 0.77489258 0.84151567 0.22620395 0.17234372 0.5156774
 0.60314193 0.80575    0.95322475 0.68277538]
Addition of two vectors is [0.20086831 1.44594244 1.12160326 1.18530362 0.28404531 1.44671784
 1.28276672 1.57827734 1.91921034 1.38699607]
Total time taken 0.03187274932861328
```

Table 1: Relation between different Array Size(N) and Number of Workers (P)

Number of Workers / Array Size	P=2	P=3	P=4
10^2	0.00103 sec	0.0 sec	0.01003 sec
10^3	0.02676 sec	0.08284 sec	0.19359 sec
10^4	0.00099 sec	0.00199 sec	0.0 sec

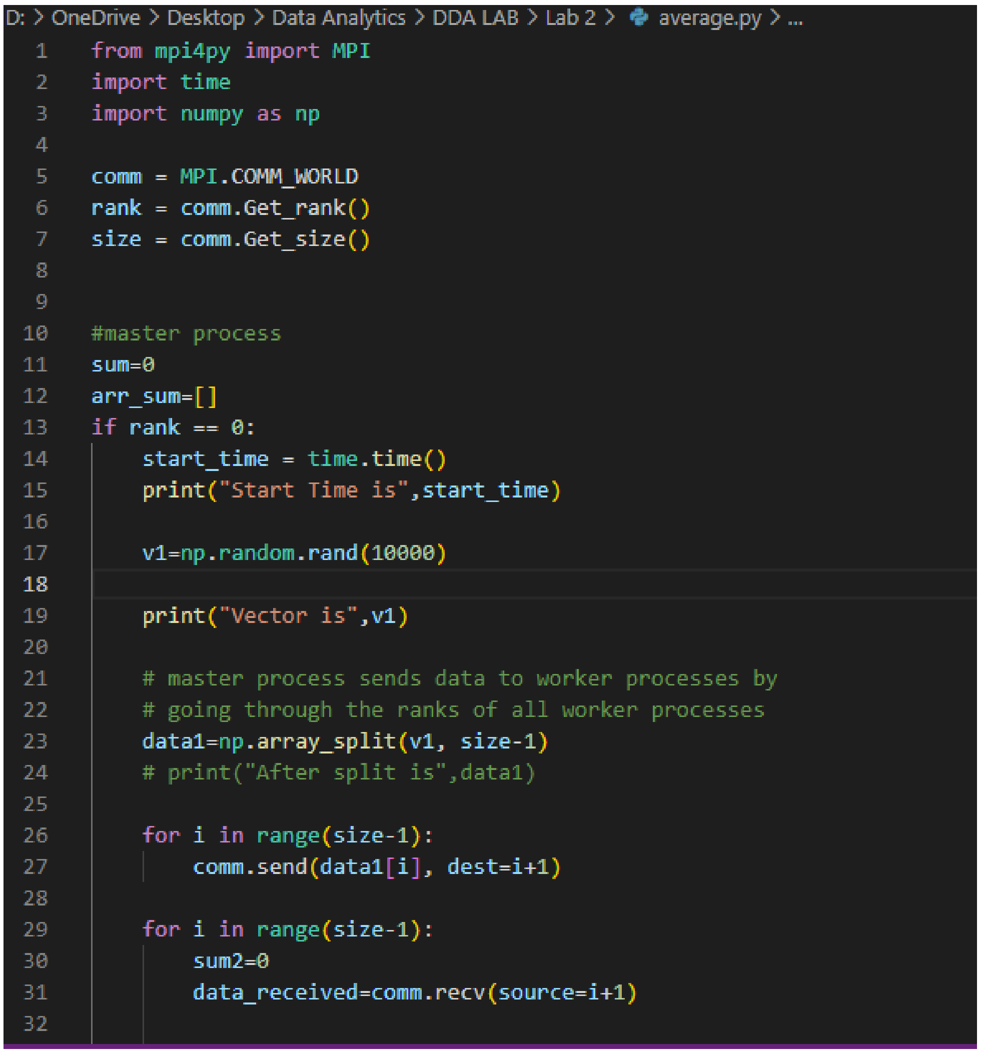
We can see from the Table with an increase in the size of matrices the time decreases. With increase in the number of workers time increase for size 2 and 3 as this creates an overhead for parallelization. But for larger size time decreases due to parallel execution. Time is in sec.

b) Find an average of numbers in a vector.

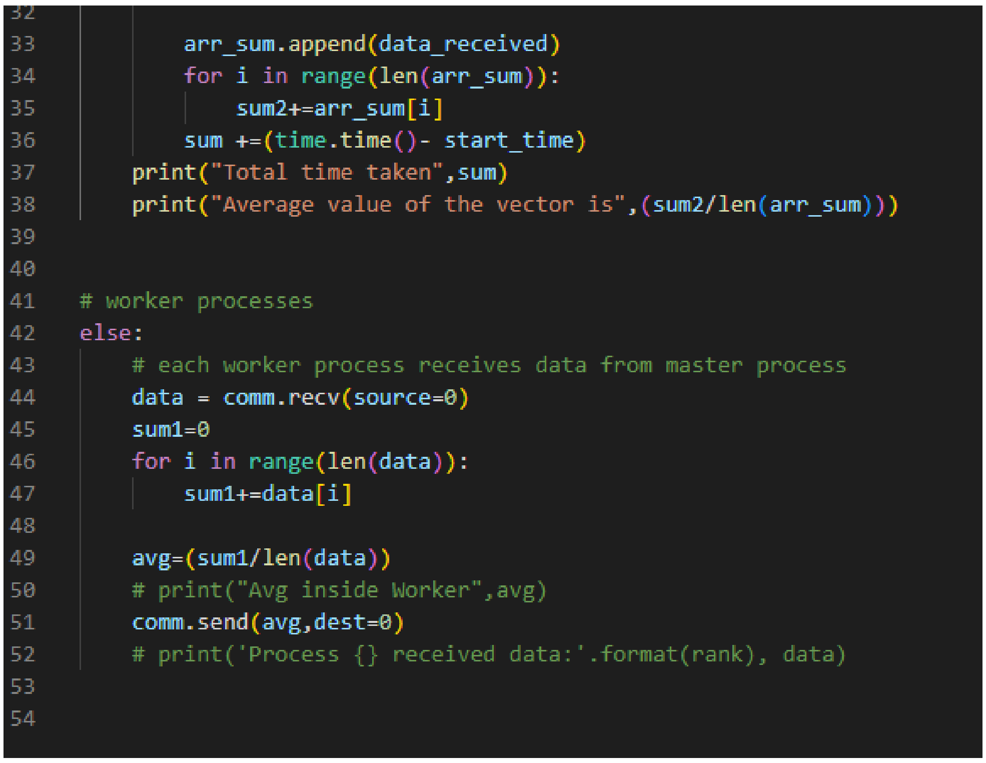
Parallelization Strategy:

- When the rank is zero (i.e master process), it splits data based on the size into chunks. Here size refers to the number of workers.
- This data is sent in the form of an array to the Worker processes.
- Data is received by the Worker process and average of particular chunk is performed there.
- Worker process send the result to the master process where it is appended in an array. Final average of the array is again taken and displayed.

Program:

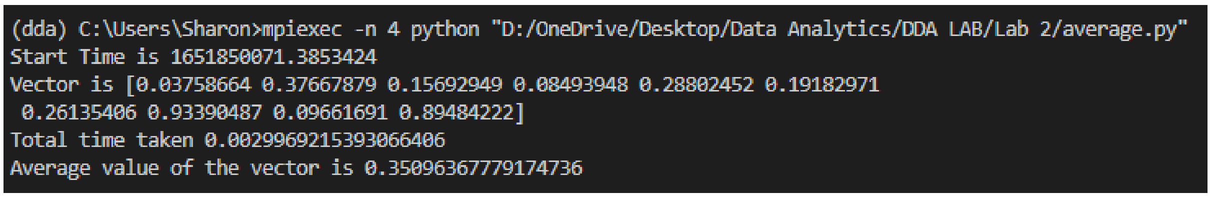

D: > OneDrive > Desktop > Data Analytics > DDA LAB > Lab 2 > average.py > ...

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

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


#master process
sum=0
arr_sum=[]
if rank == 0:
    start_time = time.time()
    print("Start Time is",start_time)

    v1=np.random.rand(10000)

    print("Vector is",v1)

    # master process sends data to worker processes by
    # going through the ranks of all worker processes
    data1=np.array_split(v1, size-1)
    # print("After split is",data1)

    for i in range(size-1):
        comm.send(data1[i], dest=i+1)

    for i in range(size-1):
        sum2=0
        data_received=comm.recv(source=i+1)

```



```python

        arr_sum.append(data_received)
        for i in range(len(arr_sum)):
            sum2+=arr_sum[i]
        sum +=(time.time()- start_time)
    print("Total time taken",sum)
    print("Average value of the vector is",(sum2/len(arr_sum)))


# worker processes
else:
    # each worker process receives data from master process
    data = comm.recv(source=0)
    sum1=0
    for i in range(len(data)):
        sum1+=data[i]

    avg=(sum1/len(data))
    # print("Avg inside Worker",avg)
    comm.send(avg,dest=0)
    # print('Process {} received data:'.format(rank), data)


```

Results are shown by taking a smaller array of size 10 to verify them.



```
(dda) C:\Users\Sharon>mpiexec -n 4 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 2/average.py"
Start Time is 1651850071.3853424
Vector is [0.03758664 0.37667879 0.15692949 0.08493948 0.28802452 0.19182971
 0.26135406 0.93390487 0.09661691 0.89484222]
Total time taken 0.0029969215393066406
Average value of the vector is 0.35096367779174736
```

Table 2: Relation between different Array Size(N) and Number of Workers (P)

Number of Worker (P) / Array Size	P=2	P=3	P=4
10^2	0.00199 sec	0.00206 sec	0.00599 sec
10^3	0.00890 sec	0.02096 sec	0.03777 sec
10^4	0.00099 sec	0.00188 sec	0.00299 sec

We can see from the Table with an increase in the size of matrices the time increases. With increase in the number of workers time increase as this creates an overhead for parallelization. Time is in sec.

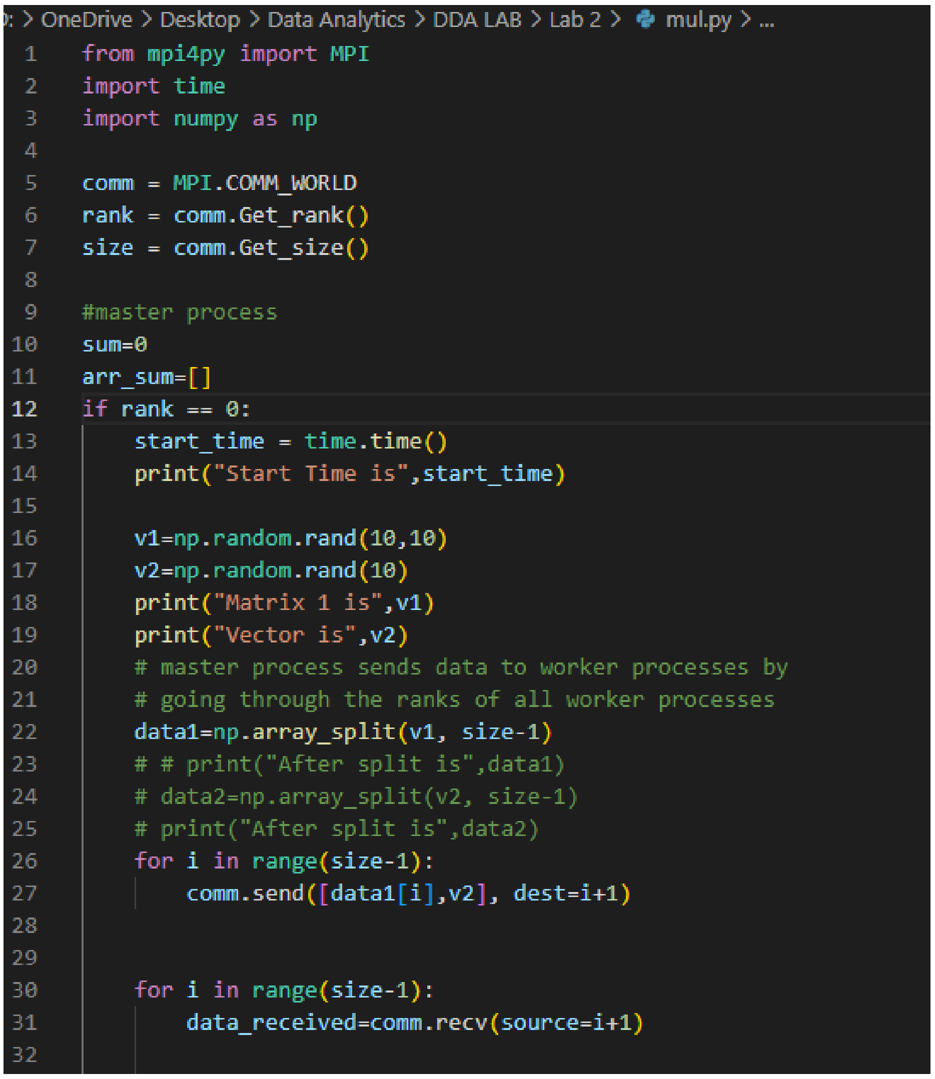
Exercise 2: Parallel Matrix-Vector multiplication using MPI

This is done using MPI Point-to-Point Communication. Send and Recv functions.

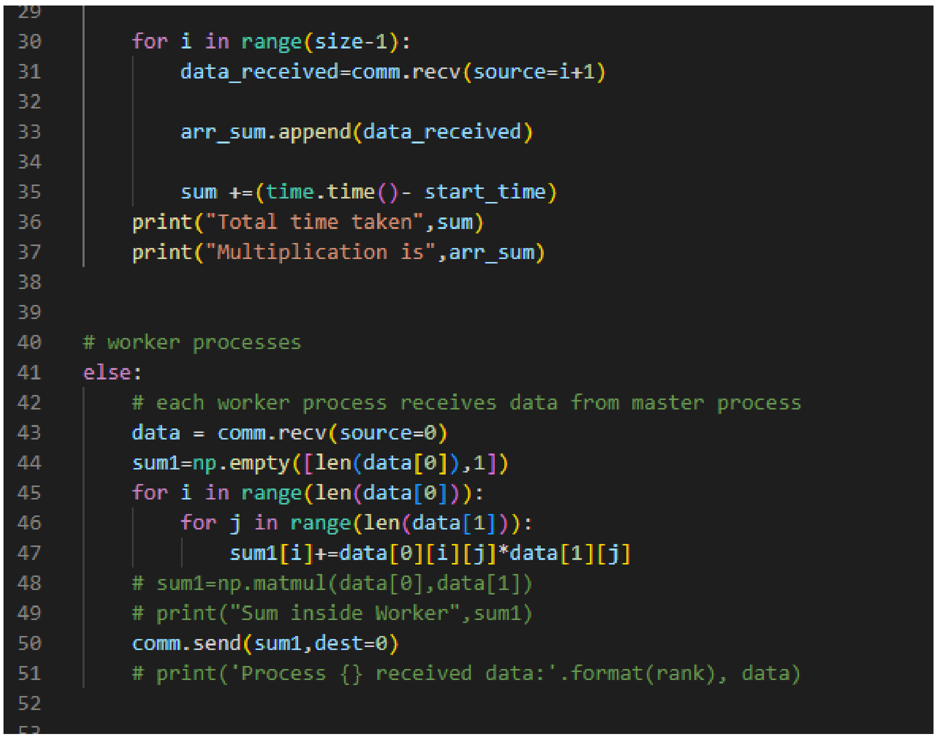
Parallelization Strategy:

- When the rank is zero (i.e master process), it splits the matrix based on the size into chunks. Here "**v1**" rows are split using **np.array_split**() according to the number of workers.
- Both the split matrix and entire vector is sent in the form of an array to the Worker processes.
- Data is received by the Worker process and multiplication of the small chunk of a matrix is performed with the vector.
- Worker processes send the result of the multiplication to the master process. It is appended in an array **arr_sum**() and displayed.

Program:

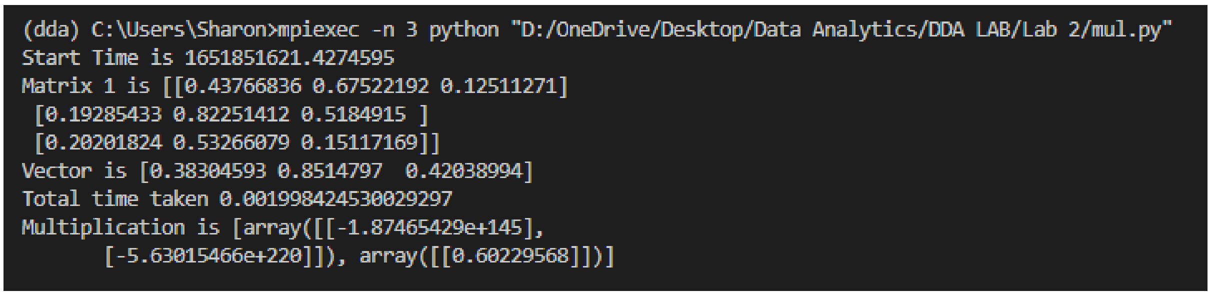

: > OneDrive > Desktop > Data Analytics > DDA LAB > Lab 2 > mul.py > ...

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

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

#master process
sum=0
arr_sum=[]
if rank == 0:
    start_time = time.time()
    print("Start Time is",start_time)

    v1=np.random.rand(10,10)
    v2=np.random.rand(10)
    print("Matrix 1 is",v1)
    print("Vector is",v2)
    # master process sends data to worker processes by
    # going through the ranks of all worker processes
    data1=np.array_split(v1, size-1)
    # # print("After split is",data1)
    # data2=np.array_split(v2, size-1)
    # print("After split is",data2)
    for i in range(size-1):
        comm.send([data1[i],v2], dest=i+1)


    for i in range(size-1):
        data_received=comm.recv(source=i+1)

```



```python

    for i in range(size-1):
        data_received=comm.recv(source=i+1)

        arr_sum.append(data_received)

        sum +=(time.time()- start_time)
    print("Total time taken",sum)
    print("Multiplication is",arr_sum)


# worker processes
else:
    # each worker process receives data from master process
    data = comm.recv(source=0)
    sum1=np.empty([len(data[0]),1])
    for i in range(len(data[0])):
        for j in range(len(data[1])):
            sum1[i]+=data[0][i][j]*data[1][j]
    # sum1=np.matmul(data[0],data[1])
    # print("Sum inside Worker",sum1)
    comm.send(sum1,dest=0)
    # print('Process {} received data:'.format(rank), data)

```

Results are shown by taking a smaller matrix of size 3X3 multiplied with vector of size 3 to verify them.



```
(dda) C:\Users\Sharon>mpiexec -n 3 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 2/mul.py"
Start Time is 1651851621.4274595
Matrix 1 is [[0.43766836 0.67522192 0.12511271]
 [0.19285433 0.82251412 0.5184915 ]
 [0.20201824 0.53266079 0.15117169]]
Vector is [0.38304593 0.8514797  0.42038994]
Total time taken 0.001998424530029297
Multiplication is [array([[-1.87465429e+145],
       [-5.63015466e+220]]), array([[0.60229568]])]
```

Table 3: Relation between different Array Size(N) and Number of Workers (P)

Number of Workers(P) / Array Size	P=2	P=3	P=4
$10^{1\times1}$ X 10	0.00348 sec	0.00398 sec	0.00299 sec
$10^{2\times2}$ X 10^2	0.03337 sec	0.03577 sec	0.01529 sec
$10^{3\times3}$ X 10^3	3.01739 sec	3.13207 sec	3.25048 sec

We can see from the Table with an increase in the size of matrices the time increases. With an increase in the number of workers time decreases. But for size (3X3) time increases as this creates an overhead for parallelization. Time is in sec.

I have tried with size 10^4 , but due to RAM it is not able to compute. So I have displayed the results for the rest of the values of n.

Exercise 3: Parallel Matrix Operation using MPI.

This is done using MPI Collective Communication. Bcast, scatter and gather functions

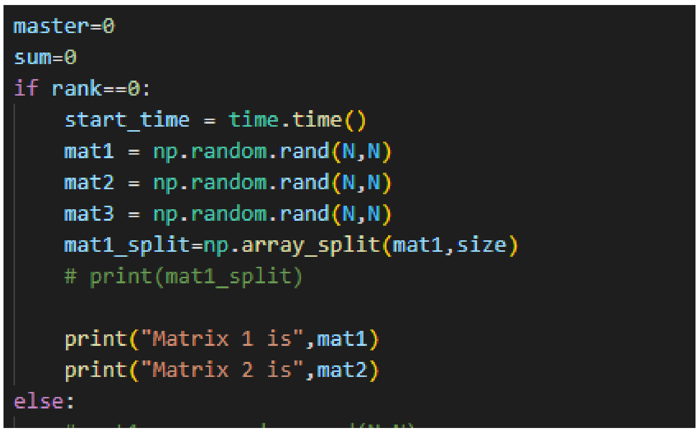
Broadcast data from one member to all members of a group.
Gather: Gather data from all members to one member of a group.
Scatter: Scatter data from one member to all members of a group.

Parallelization Strategy:

- When the rank is zero (i.e master process), 3 matrices are initialized.
- For rank==0, the matrix is split into chunks based on its size.
- Matrix 1 is scattered to worker processes and the result is stored in data.
- Matrix 2 is broadcasted to all members of the group.
- Gather performs matrix multiplication from all processes. And the result is stored in mat3.
- Finally, the rank is checked if it is zero and the result is displayed along with the total time taken.

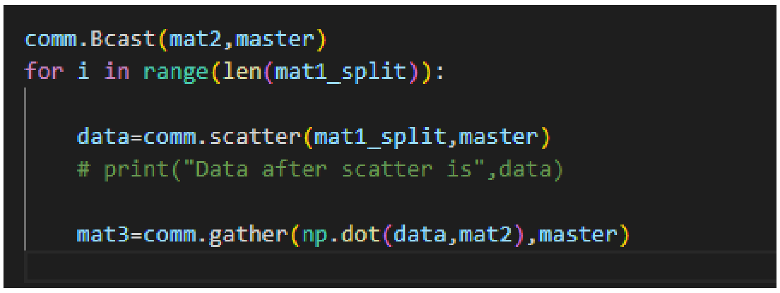
Program:

Mat1 and mat2 are the two matrices to be multiplied and the result gets stored in mat3.



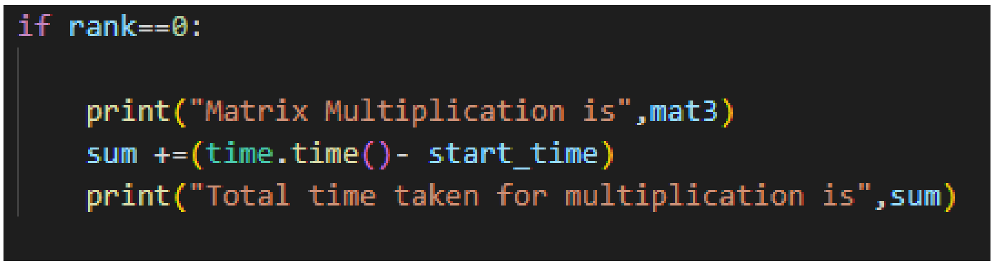
```
master=0
sum=0
if rank==0:
    start_time = time.time()
    mat1 = np.random.rand(N,N)
    mat2 = np.random.rand(N,N)
    mat3 = np.random.rand(N,N)
    mat1_split=np.array_split(mat1,size)
    # print(mat1_split)

    print("Matrix 1 is",mat1)
    print("Matrix 2 is",mat2)
else:
```

Mat2 is broadcasted to all workers. During the broadcast, the same information is sent to all the workers.

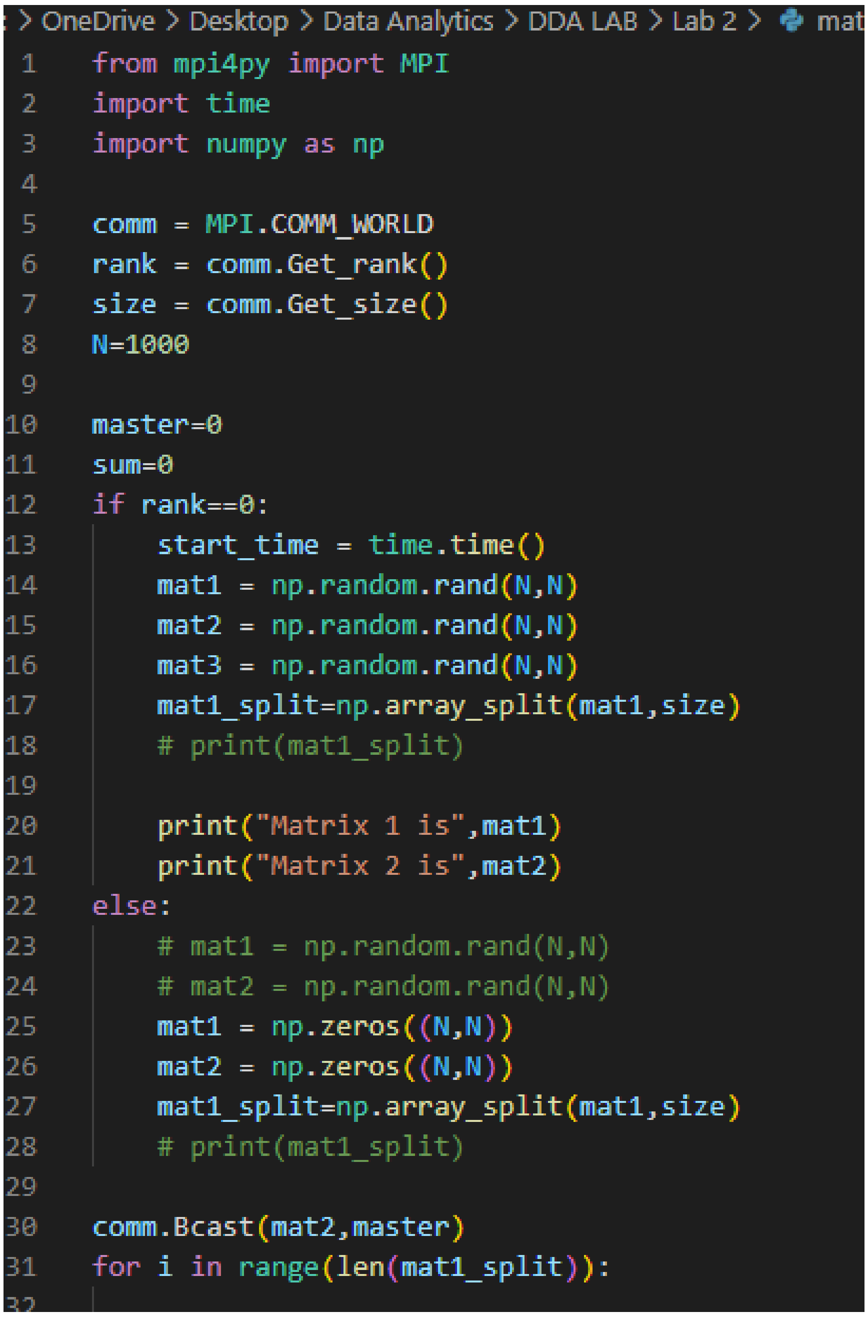


```
comm.Bcast(mat2,master)
for i in range(len(mat1_split)):

    data=comm.scatter(mat1_split,master)
    # print("Data after scatter is",data)

    mat3=comm.gather(np.dot(data,mat2),master)
```

After the results are gathered, we check if the rank is one. And then print the final matrix multiplication and the time taken for the process.

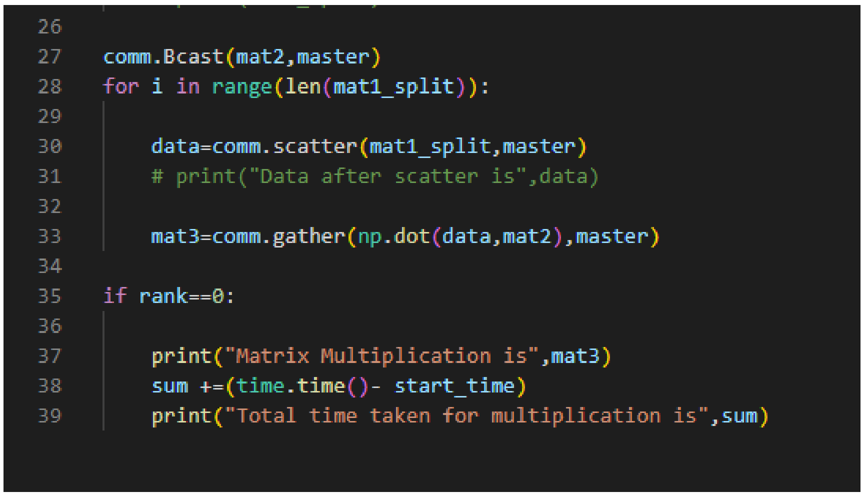


```
if rank==0:

    print("Matrix Multiplication is",mat3)
    sum +=(time.time()- start_time)
    print("Total time taken for multiplication is",sum)
```



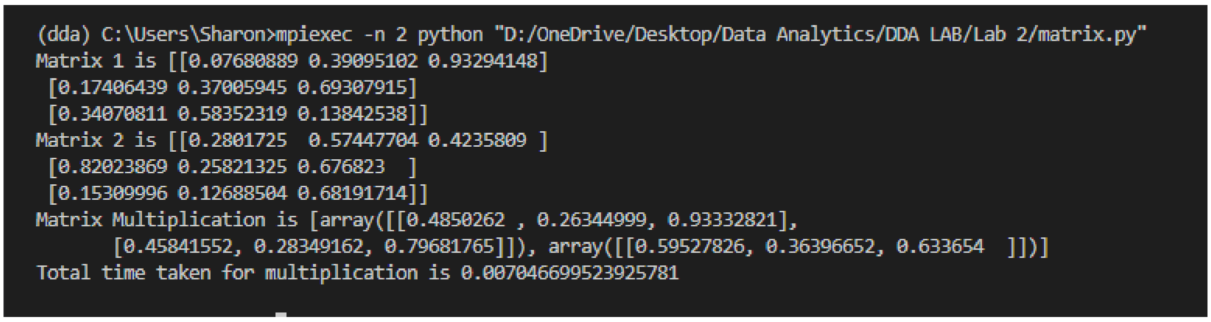
: > OneDrive > Desktop > Data Analytics > DDA LAB > Lab 2 > mat

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

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
N=1000

master=0
sum=0
if rank==0:
    start_time = time.time()
    mat1 = np.random.rand(N,N)
    mat2 = np.random.rand(N,N)
    mat3 = np.random.rand(N,N)
    mat1_split=np.array_split(mat1,size)
    # print(mat1_split)

    print("Matrix 1 is",mat1)
    print("Matrix 2 is",mat2)
else:
    # mat1 = np.random.rand(N,N)
    # mat2 = np.random.rand(N,N)
    mat1 = np.zeros((N,N))
    mat2 = np.zeros((N,N))
    mat1_split=np.array_split(mat1,size)
    # print(mat1_split)

comm.Bcast(mat2,master)
for i in range(len(mat1_split)):
```



```python

comm.Bcast(mat2,master)
for i in range(len(mat1_split)):

    data=comm.scatter(mat1_split,master)
    # print("Data after scatter is",data)

    mat3=comm.gather(np.dot(data,mat2),master)

if rank==0:

    print("Matrix Multiplication is",mat3)
    sum +=(time.time()- start_time)
    print("Total time taken for multiplication is",sum)
```

Results are shown by taking a smaller matrix of size 3X3 multiplied with matrix of size 3X3 to verify them.



```
(dda) C:\Users\Sharon>mpiexec -n 2 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 2/matrix.py"
Matrix 1 is [[0.07680889 0.39095102 0.93294148]
 [0.17406439 0.37005945 0.69307915]
 [0.34070811 0.58352319 0.13842538]]
Matrix 2 is [[0.2801725  0.57447704 0.4235809 ]
 [0.82023869 0.25821325 0.676823  ]
 [0.15309996 0.12688504 0.68191714]]
Matrix Multiplication is [array([[0.4850262 , 0.26344999, 0.93332821],
       [0.45841552, 0.28349162, 0.79681765]]), array([[0.59527826, 0.36396652, 0.633654  ]])]
Total time taken for multiplication is 0.007046699523925781
```

Table 4: Relation between different Array Size(N) and Number of Workers (P)

Number of Workers / Array Size	P=2	P=3	P=4
$10^{1 \times 1}$ X 10	0.00958 sec	0.00918	0.01144
$10^{2 \times 2}$ X 10^2	0.00864	0.00834	0.01138
$10^{3 \times 3}$ X 10^3	0.13293	0.18316	0.24202

We can see from the Table with an increase in the size of matrices the time increases. With increase in the number of workers time decreases. But for size (3X3) time increase as this creates an overhead for parallelization. Time is in sec.

I have tried with size 10^4 , but due to RAM it is not able to compute. So I have displayed the results for rest of the values of n.

References:

1. https://mpi4py.readthedocs.io/en/stable/tutorial.html
2. https://pdc-support.github.io/introduction-to-mpi/aio/index.html
3. https://www.christianbaun.de/CGC1718/Skript/CloudPresentation_Shamima_Akhter.pdf