Mansoor Nabawi, 309498.

Tutorial 2, DDA 05.05.2022

Exercise 0

Name	Specification
Memory	4.7 GiB
Processor	Intel® Core™ i5-8250U CPU @ 1.60GHz × 4
Number of CPUs	4
GNOME	Version 3.32.2
OS type	64-bit
Virtualization	Oracle
Disk	110 GB
Python	3.8.8
IDLE	Jupyter-notebook/Spyder

I am using a CentOS8 on VirtualBox with above specification

Exercise 1

In all the exercises we divided datas into batches so each process takes it's own part and only master process reduce the tasks at the end.

1.A

In [44]:

```
%%writefile ex01.py

from mpi4py import MPI
import numpy as np

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

#stat = MPI.Status()


#size of array
N=10**8
#print(size)
#print(rank)

#how many elements per rank we can take?
Batch = round(N/size)

#sum using plus
def sum_vec(a,b):
  z = a+b
  return z

#sum by adding each element together
def sum_vec_(a,b):
  L = len(a)
  z = np.zeros(L)
  for i in range(L):
    z[i] = a[i] + b[i]
  return z

#timing
t1 = MPI.Wtime()

# master process
if rank == 0:

    #creating sample data
    data1 = np.random.random(size = (N,1))
    data2 = np.random.random(size = (N,1))

    #stacking data together to send to other processes
    data = np.hstack([data1, data1])
    # master process sends data to worker processes by
    # going through the ranks of all worker processes
    for i in range(1, size):

        #sending the matrices
        comm.Send(data, dest=i, tag=i)
        #print(f'From Process {rank} to -> process {i}\n Data: \n{subs[i]}')

    #computing the sum of the batch of data for process 0
    res = sum_vec(data[(Batch*rank):(Batch*(1+rank)),0], data[(Batch*rank):(Batch*(1+ra

    #receiving and doing final summation
    final_dt = [res]
    for i in range(1,size):#all processes except 0
        #shape of data
        shp_ = np.empty(1, dtype=int)#buffer
        comm.Recv(shp_, source=i, tag=i+size*2)

        #data itself
        data_f = np.empty((shp_[0],))#buffer
        comm.Recv(data_f, source=i, tag=i+size*3)
        final_dt.append(data_f)
    #concatenating to get the final result
    final_res = np.concatenate(final_dt, axis=0)
    #print("final result: ",final_res)
    #print("finallyyyyy....",final_res.shape)

    print("Time in rank 0 is:", MPI.Wtime()-t1)


# worker processes

else:
    #receicing data and performing actions
    data_ = np.empty((N,2))
    comm.Recv(data_, source=0, tag=rank)

    #summation
    res = sum_vec(data_[(Batch*rank):(Batch*(1+rank)),0], data_[(Batch*rank):(Batch*(1+
    #sending the shape of data
    shp = np.array(res.shape[0], dtype=int)
    comm.Send(shp, dest=0, tag=size*2+rank)
    #sending data itself
    comm.Send(res, dest=0, tag=rank+size*3)

    #print("Time in rank {} is:".format(rank), MPI.Wtime()-t1_)
    #print('Process {} received data:'.format(rank), data_.shape)
   # print(f'\nResult sent \n From Process {rank} to -> process {0}\n Data: \n{res}')
```

```
Overwriting ex01.py
```

In [2]:

```
#N=10**5
```

In [3]:

```
!mpiexec -np 1 python ex01.py
```

```
Time in rank 0 is: 0.005569934844970703
```

In [4]:

```
!mpiexec -np 2 python ex01.py
```

```
Time in rank 0 is: 0.03367304801940918
```

In [5]:

```
!mpiexec -np 3 python ex01.py
```

```
Time in rank 0 is: 0.007166147232055664
```

In [6]:

```
!mpiexec -np 4 python ex01.py
```

```
Time in rank 0 is: 0.009217977523803711
```

In [8]:

```
#N=10**6
```

In [13]:

```
!mpiexec -np 1 python ex01.py
```

```
Time in rank 0 is: 0.04259800910949707
```

In [14]:

```
!mpiexec -np 2 python ex01.py
```

```
Time in rank 0 is: 0.04929089546203613
```

In [15]:

```
!mpiexec -np 3 python ex01.py
```

```
Time in rank 0 is: 0.0678260326385498
```

In [16]:

```
!mpiexec -np 4 python ex01.py
```

```
Time in rank 0 is: 0.05819201469421387
```

In [18]:

```
#N=10**7
```

In [24]:

```
!mpiexec -np 1 python ex01.py
```

```
Time in rank 0 is: 0.4012110233306885
```

In [25]:

```
!mpiexec -np 2 python ex01.py
```

```
Time in rank 0 is: 0.43552303314208984
```

In [26]:

```
!mpiexec -np 3 python ex01.py
```

```
Time in rank 0 is: 0.48368406295776367
```

In [27]:

```
!mpiexec -np 4 python ex01.py
```

```
Time in rank 0 is: 0.5306990146636963
```

In [149…

```
#N=10**8
```

In [29]:

```
!mpiexec -np 1 python ex01.py
```

```
Time in rank 0 is: 14.720407009124756
```

In [30]:

```
!mpiexec -np 2 python ex01.py
```

```
Time in rank 0 is: 51.952938079833984
```

In [31]:

```
!mpiexec -np 3 python ex01.py
```

```
Traceback (most recent call last):
  File "ex01.py", line 58, in <module>
    final_res = np.concatenate(final_dt, axis=0)
  File "<__array_function__ internals>", line 5, in concatenate
numpy.core._exceptions.MemoryError: Unable to allocate 763. MiB for an array with shape
(99999999,) and data type float64
```

In [32]:

```
!mpiexec -np 4 python ex01.py
```

```
===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 14481 RUNNING AT mansoor
=   EXIT CODE: 9
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Killed (signal 9)
This typically refers to a problem with your application.
Please see the FAQ page for debugging suggestions
```

Report 1.A

N	P=1	P=2	P=3	P=4
10**5	0.0055	0.0336	0.00716	0.0092
10**6	0.04259	0.0492	0.0678	0.05819
10**7	0.4012	0.435	0.4836	0.5306
10**8	14.7204	51.952	MemoryError	EXIT CODE: 9

- As it is visible from the above table, by increasing the number of workers only in **N=10^5** with p=3 it had better performance than p=2 but p=1 is the fastest.
- Also in **N=10^6**, we see that when using 4 workers it has slightly better performance than using 3 workers, but still the p=1 is the fastest.
- in **N=10^8**, it takes longer to compute and with p=3 we get a MemoryError and with p=4 the program is terminated

---

Strategy

When Rank==0:

1. First We make a batch based on the data size and number of workers.
2. Rank0 always start the program by initializing the matrices.
3. Next we stack the data together to send it to other processes easily.
4. Rank0 send the stacked data to all other processes with a specific tag.
5. Rank0 itself receives the batch of the data and perform the summation.
6. Rank0 waits until other processes do their task and send the data to Rank0, as soon as it receives the data it first append to a list and at final stage it concatenate all members of the list to make the final data.

When Other Ranks:

1. Since we use Send/Recv we need to create a buffer before receiving the data.
2. The data is received from process 0 to other processes, the data is put in the buffer.
3. The summation is peformed and the result is going to be sent to the main process, but first the shape is sent to be used for size buffer for the data.
4. The result is sent to process 0

End:

- After all the processors did their task and sent their data back to master processor, the program will end

In [33]:

```
### Just checking with another way of matrix summation
```

In [38]:

```
%%writefile ex01_.py

from mpi4py import MPI
import numpy as np

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

stat = MPI.Status()


#size of array
N=10**7
#print(size)
#print(rank)

Batch = round(N/size)

def sum_vec(a,b):
  z = a+b
  return z

def sum_vec_(a,b):
  L = len(a)
  z = np.zeros(L)
  for i in range(L):
    z[i] = a[i] + b[i]
  return z

t1 = MPI.Wtime()
# master process
if rank == 0:

    data1 = np.random.random(size = (N,1))
    data2 = np.random.random(size = (N,1))

    data = np.hstack([data1, data1])
    # master process sends data to worker processes by
    # going through the ranks of all worker processes
    for i in range(1, size):

        #sending the matrices
        comm.Send(data, dest=i, tag=i)
        #print(f'From Process {rank} to -> process {i}\n Data: \n{subs[i]}')

    res = sum_vec_(data[(Batch*rank):(Batch*(1+rank)),0], data[(Batch*rank):(Batch*(1+r

    #receiving and doing final summation
    final_dt = [res]
    for i in range(1,size):
        shp_ = np.empty(1, dtype=int)
        comm.Recv(shp_, source=i, tag=i+size*2)

        data_f = np.empty((shp_[0],))
        comm.Recv(data_f, source=i, tag=i+size*3)
        final_dt.append(data_f)

    final_res = np.concatenate(final_dt, axis=0)
    #print("final result: ",final_res)
    #print("finallyyyyy....",final_res.shape)

    print("Time in rank 0 is:", MPI.Wtime()-t1)


# worker processes

else:
    data_ = np.empty((N,2))
    comm.Recv(data_, source=0, tag=rank)

    #for i in range(100):
    res = sum_vec_(data_[(Batch*rank):(Batch*(1+rank)),0], data_[(Batch*rank):(Batch*(1
    #data_r = sum_vec(data_[:,0], data_[:,1])

    shp = np.array(res.shape[0], dtype=int)
    comm.Send(shp, dest=0, tag=size*2+rank)
    comm.Send(res, dest=0, tag=rank+size*3)

    #print("Time in rank {} is:".format(rank), MPI.Wtime()-t1_)
    #print('Process {} received data:'.format(rank), data_.shape)
   # print(f'\nResult sent \n From Process {rank} to -> process {0}\n Data: \n{res}')
```

```
Overwriting ex01_.py
```

In [39]:

```
!mpiexec -np 1 python ex01_.py
```

```
Time in rank 0 is: 11.45203185081482
```

In [40]:

```
!mpiexec -np 2 python ex01_.py
```

```
Time in rank 0 is: 5.622430086135864
```

In [41]:

```
!mpiexec -np 3 python ex01_.py
```

```
Time in rank 0 is: 4.057527780532837
```

In [42]:

```
!mpiexec -np 4 python ex01_.py
```

```
Time in rank 0 is: 3.2704930305480957
```

Exercise 01_b

In [76]:

```
%%writefile ex01_b.py

from mpi4py import MPI
import numpy as np

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

stat = MPI.Status()


#size of array
N=10**8
#print(size)
#print(rank)

#Batch
Batch = round(N/size)

#find average

def find_avg(a):
    avg = np.sum(a)/len(a)
    return avg

t1 = MPI.Wtime()
# master process
if rank == 0:
    #data initialization
    data = np.random.random(size = (N,1))

    # master process sends data to worker processes by
    # going through the ranks of all worker processes
    for i in range(1, size):

        #sending the matrices
        comm.Send(data, dest=i, tag=i)
        #print(f'From Process {rank} to -> process {i}\n Data: \n{subs[i]}')

    res = find_avg(data[(Batch*rank):(Batch*(1+rank)),0])

    #receiving and doing final summation
    final_dt = [res]
    for i in range(1,size):

        data_f = np.empty((1,))
        comm.Recv(data_f, source=i, tag=i+size*3)
        final_dt.append(data_f)

    #final_res = np.concatenate(final_dt, axis=0)
    print("\naverage of all processes: ",sum(final_dt)/len(final_dt))
    print("\n Average in rank 0 (complete data): ",find_avg(data))

    print("Time in rank 0 is:", MPI.Wtime()-t1)


# worker processes

else:
    #Receiving data, performing action, sending back to rank0

    data_ = np.empty((N,1))
    comm.Recv(data_, source=0, tag=rank)

    res = find_avg(data_[(Batch*rank):(Batch*(1+rank)),0])

    comm.Send(res, dest=0, tag=rank+size*3)
    #print("Time in rank {} is:".format(rank), MPI.Wtime()-t1_)
    #print('Process {} received data:'.format(rank), data_.shape)
    print(f'\nResult sent \n From Process {rank} to -> process {0} -> average is: {res}
```

```
Overwriting ex01_b.py
```

In [47]:

```
#N=10**5
```

In [62]:

```
!mpiexec -np 1 python ex01_b.py
```

```
average of all processes:  0.5006752382005252

 Average in rank 0 (complete data):  0.5006752382005252
Time in rank 0 is: 0.0020799636840820312
```

In [63]:

```
!mpiexec -np 2 python ex01_b.py
```

```
average of all processes:  [0.49875728]

 Average in rank 0 (complete data):  0.49875727769334893
Time in rank 0 is: 0.0027980804443359375

Result sent
 From Process 1 to -> process 0 -> average is: 0.4978065801580282
```

In [64]:

```
!mpiexec -np 3 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.5018343789616073

Result sent
 From Process 2 to -> process 0 -> average is: 0.5006353741933652

average of all processes:  [0.5014581]

 Average in rank 0 (complete data):  0.5014576544755762
Time in rank 0 is: 0.022327899932861328
```

In [65]:

```
!mpiexec -np 4 python ex01_b.py
```

```
average of all processes:  [0.50063504]

 Average in rank 0 (complete data):  0.5006350397001725
Time in rank 0 is: 0.0039141178131103516

Result sent
 From Process 1 to -> process 0 -> average is: 0.4993707406654497

Result sent
 From Process 2 to -> process 0 -> average is: 0.5035168433173906

Result sent
 From Process 3 to -> process 0 -> average is: 0.4983482692397172
```

In [ ]:

```
#N=10**6
```

In [67]:

```
!mpiexec -np 1 python ex01_b.py
```

```
average of all processes:  0.49991940537174945

 Average in rank 0 (complete data):  0.49991940537174945
Time in rank 0 is: 0.02193284034729004
```

In [68]:

```
!mpiexec -np 2 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.5003881667265816

average of all processes:  [0.50026862]

 Average in rank 0 (complete data):  0.5002686186372196
Time in rank 0 is: 0.022748947143554688
```

In [69]:

```
!mpiexec -np 3 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.5011521048043635

Result sent
 From Process 2 to -> process 0 -> average is: 0.49941585619033146

average of all processes:  [0.50023216]

 Average in rank 0 (complete data):  0.5002326421876567
Time in rank 0 is: 0.02737593650817871
```

In [70]:

```
!mpiexec -np 4 python ex01_b.py
```

```
average of all processes:  [0.49969929]

 Average in rank 0 (complete data):  0.4996992910470642
Time in rank 0 is: 0.028100967407226562

Result sent
 From Process 1 to -> process 0 -> average is: 0.4998374459464813

Result sent
 From Process 2 to -> process 0 -> average is: 0.49978330678419985

Result sent
 From Process 3 to -> process 0 -> average is: 0.4998523433929824
```

In [ ]:

```
#N=10**7
```

In [72]:

```
!mpiexec -np 1 python ex01_b.py
```

```
average of all processes:  0.49991197947978944

 Average in rank 0 (complete data):  0.49991197947978944
Time in rank 0 is: 0.17930102348327637
```

In [73]:

```
!mpiexec -np 2 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.5000099661746804

average of all processes:  [0.50001165]

 Average in rank 0 (complete data):  0.5000116537529374
Time in rank 0 is: 0.19699716567993164
```

In [74]:

```
!mpiexec -np 3 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.49985877031539033

Result sent
```

```
 From Process 2 to -> process 0 -> average is: 0.5000189508274225

average of all processes:  [0.50001178]

 Average in rank 0 (complete data):  0.5000117479407509
Time in rank 0 is: 0.21705389022827148
```

In [75]:

```
!mpiexec -np 4 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.4997507859018565

Result sent
 From Process 2 to -> process 0 -> average is: 0.5000337200155665

average of all processes:  [0.49984501]

 Average in rank 0 (complete data):  0.4998450084167889
Time in rank 0 is: 0.2613978385925293

Result sent
 From Process 3 to -> process 0 -> average is: 0.4997723141586117
```

In [39]:

```
#N=10**8
```

In [77]:

```
!mpiexec -np 1 python ex01_b.py
```

```
average of all processes:  0.5000040859584217

 Average in rank 0 (complete data):  0.5000040859584217
Time in rank 0 is: 1.7917490005493164
```

In [78]:

```
!mpiexec -np 2 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.4999667910085194

average of all processes:  [0.49997988]

 Average in rank 0 (complete data):  0.499979877061593
Time in rank 0 is: 2.0573461055755615
```

In [79]:

```
!mpiexec -np 3 python ex01_b.py
```

```
Result sent
 From Process 1 to -> process 0 -> average is: 0.50004774949159

Result sent
 From Process 2 to -> process 0 -> average is: 0.4999918077500031

average of all processes:  [0.50001787]

 Average in rank 0 (complete data):  0.5000178713581599
Time in rank 0 is: 2.2844340801239014
```

In [80]:

```
!mpiexec -np 4 python ex01.py
```

```
===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 17022 RUNNING AT mansoor
=   EXIT CODE: 9
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Killed (signal 9)
```

```
This typically refers to a problem with your application.
```

Report

Table of Times for 1.B

N	P=1	P=2	P=3	P=4
10**5	002079	0.00279	0.02232	0.0039
10**6	0.0219	0.0227	0.02737	0.0281
10**7	0.179	0.196	0.2170	0.2613
10**8	1.7917	2.0573	2.2844	EXIT CODE: 9

- when N=10^5 we see by increasing the number of the workers, the computation speed is decreased.
- in all other cases we see a very small increase in the computation speed.

The problem that we don't see that much decrease speed might be because i am using a virutalmachine

Strategy

SAME AS PREVIOUS

When Rank==0:

1. First We make a batch based on the data size and number of workers.
2. Rank0 always start the program by initializing the matrices.
3. Next we stack the data together to send it to other processes easily.
4. Rank0 send the stacked data to all other processes with a specific tag.
5. Rank0 itself receives the batch of the data and perform the summation.
6. Rank0 waits until other processes do their task and send the data to Rank0, as soon as it receives the data it first append to a list and at final stage it concatenate all members of the list to make the final data.

When Other Ranks:

1. Since we use Send/Recv we need to create a buffer before receiving the data.
2. The data is received from process 0 to other processes, the data is put in the buffer.
3. The summation is peformed and the result is going to be sent to the main process, but first the shape is sent to be used for size buffer for the data.
4. The result is sent to process 0

End:

- After all the processors did their task and sent their data back to master processor, the program will end

Exercise 2

In [396…

```
%%writefile ex02_a.py

from mpi4py import MPI
import numpy as np

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

stat = MPI.Status()


#size of array
N=10**5
#print(size)
#print(rank)

Batch = round(N/size)



t1 = MPI.Wtime()
# master process
if rank == 0:

    data1 = np.random.random(size = (N,1))
    data2 = np.random.random(size = (N,1))
    matrix_a = np.zeros(shape=(N, N))

    # master process sends data to worker processes by
    # going through the ranks of all worker processes
    for i in range(1, size):

        #sending the matrices
        comm.Send(data1, dest=i, tag=i)
        comm.Send(data2, dest=i, tag=i*2)
        #print(f'From Process {rank} to -> process {i}\n Data: \n{subs[i]}')



    d1 = data1[(Batch*rank):(Batch*(1+rank)),:]


    vector_s = np.dot(d1, data2.T)

    print(f"shape in rank {rank}, is: {vector_s.shape}")

    #receiving and doing final summation
    final_dt = [vector_s]
    #receiving data
    for i in range(1,size):
        shp_ = np.empty(2, dtype=int)
        comm.Recv(shp_, source=i, tag=i+size*2)

        data_f = np.empty((shp_[0],shp_[1]))
        comm.Recv(data_f, source=i, tag=i+size*3)
        final_dt.append(data_f)
    #final result
    final_res = np.concatenate(final_dt, axis=0)
    print("final result: ",final_res.shape)
    #print("finallyyyyy....",vector_s.shape)

    print("Time in rank 0 is:", MPI.Wtime()-t1)


# worker processes

else:
    #creating buffer, recevining datas, peroforming tasks.
    #sending back to the master process.

    data_1 = np.empty((N,1))
    comm.Recv(data_1, source=0, tag=rank)

    data_1 = data_1[(Batch*rank):(Batch*(1+rank)),:]
    #print(data_1.shape)

    data_2 = np.empty((N,1))
    comm.Recv(data_2, source=0, tag=rank*2)


    vector_s = np.dot(data_1, data_2.T)
    print(f"shape in rank {rank}, is: {vector_s.shape}")


    shp = np.array(vector_s.shape, dtype=int)
    comm.Send(shp, dest=0, tag=size*2+rank)
    comm.Send(vector_s, dest=0, tag=rank+size*3)

    #print("Time in rank {} is:".format(rank), MPI.Wtime()-t1_)
    #print('Process {} received data:'.format(rank), data_.shape)
    #print(f'\nResult sent \n From Process {rank} to -> process {0}\n Data shape: \n{re
```

```
Overwriting ex02_a.py
```

In [381…

```
!mpiexec -np 1 python ex02_a.py
```

```
shape in rank 0, is: (100, 100)
final result:  (100, 100)
Time in rank 0 is: 0.00017714500427246094
```

In [382…

```
!mpiexec -np 2 python ex02_a.py
```

```
shape in rank 1, is: (50, 100)
shape in rank 0, is: (50, 100)
final result:  (100, 100)
Time in rank 0 is: 0.00027298927307128906
```

In [383…

```
!mpiexec -np 3 python ex02_a.py
```

```
shape in rank 1, is: (33, 100)
shape in rank 2, is: (33, 100)
shape in rank 0, is: (33, 100)
final result:  (99, 100)
Time in rank 0 is: 0.0022530555725097656
```

In [384…

```
!mpiexec -np 4 python ex02_a.py
```

```
shape in rank 1, is: (25, 100)
shape in rank 0, is: (25, 100)
final result:  (100, 100)
Time in rank 0 is: 0.011131048202514648
shape in rank 2, is: (25, 100)
shape in rank 3, is: (25, 100)
```

In [ ]:

```
#N=10**3
```

In [386…

```
!mpiexec -np 1 python ex02_a.py
```

```
shape in rank 0, is: (1000, 1000)
final result:  (1000, 1000)
Time in rank 0 is: 0.012784004211425781
```

In [387…

```
!mpiexec -np 2 python ex02_a.py
```

```
shape in rank 0, is: (500, 1000)
final result:  (1000, 1000)
Time in rank 0 is: 0.0314030647277832
shape in rank 1, is: (500, 1000)
```

In [388…

```
!mpiexec -np 3 python ex02_a.py
```

```
shape in rank 1, is: (333, 1000)
shape in rank 0, is: (333, 1000)
final result:  (999, 1000)
Time in rank 0 is: 0.06317996978759766
shape in rank 2, is: (333, 1000)
```

In [389…

```
!mpiexec -np 4 python ex02_a.py
```

```
shape in rank 0, is: (250, 1000)
final result:  (1000, 1000)
Time in rank 0 is: 0.049365997314453125
shape in rank 1, is: (250, 1000)
shape in rank 2, is: (250, 1000)
shape in rank 3, is: (250, 1000)
```

In [391…

```
#N=10**4
```

In [392…

```
!mpiexec -np 1 python ex02_a.py
```

```
shape in rank 0, is: (10000, 10000)
final result:  (10000, 10000)
Time in rank 0 is: 0.5667219161987305
```

In [393…

```
!mpiexec -np 2 python ex02_a.py
```

```
shape in rank 1, is: (5000, 10000)
shape in rank 0, is: (5000, 10000)
final result:  (10000, 10000)
Time in rank 0 is: 0.49054718017578125
```

In [394…

```
!mpiexec -np 3 python ex02_a.py
```

```
shape in rank 1, is: (3333, 10000)
shape in rank 2, is: (3333, 10000)
shape in rank 0, is: (3333, 10000)
final result:  (9999, 10000)
Time in rank 0 is: 0.4853520393371582
```

In [395…

```
!mpiexec -np 4 python ex02_a.py
```

```
shape in rank 1, is: (2500, 10000)
shape in rank 2, is: (2500, 10000)
shape in rank 3, is: (2500, 10000)
shape in rank 0, is: (2500, 10000)
final result:  (10000, 10000)
Time in rank 0 is: 0.49196314811706543
```

In [ ]:

```
#N=10**5
```

In [397…

```
!mpiexec -np 1 python ex02_a.py
```

```
Traceback (most recent call last):
  File "ex02_a.py", line 27, in <module>
    matrix_a = np.zeros(shape=(N, N))
numpy.core._exceptions.MemoryError: Unable to allocate 74.5 GiB for an array with shape
(100000, 100000) and data type float64
```

In [398…

```
!mpiexec -np 2 python ex02_a.py
```

```
Traceback (most recent call last):
  File "ex02_a.py", line 27, in <module>
    matrix_a = np.zeros(shape=(N, N))
numpy.core._exceptions.MemoryError: Unable to allocate 74.5 GiB for an array with shape
(100000, 100000) and data type float64
^C
[mpiexec@mansoor] Sending Ctrl-C to processes as requested
[mpiexec@mansoor] Press Ctrl-C again to force abort
```

In [399…

```
!mpiexec -np 3 python ex02_a.py
```

```
Traceback (most recent call last):
  File "ex02_a.py", line 27, in <module>
    matrix_a = np.zeros(shape=(N, N))
numpy.core._exceptions.MemoryError: Unable to allocate 74.5 GiB for an array with shape
(100000, 100000) and data type float64
^C
[mpiexec@mansoor] Sending Ctrl-C to processes as requested
[mpiexec@mansoor] Press Ctrl-C again to force abort
```

In [ ]:

```
!mpiexec -np 4 python ex02_a.py
```

Report 2.A

Table of Times for 2

N	P=1	P=2	P=3	P=4
10**2	0.000177	0.000272	0.00225	0.01113
10**3	0.01278	0.0314	0.0631	0.04936
10**4	0.56672	0.49054	0.48535	0.491963

- N=10^2 increase in number of workers don't have effect in the decrease of computation speed.
- N=10^3 only with p=4 we have a better performance than one worker less but overall p=1 has the best performance.
- N= 10^4 we see that by increasing the number of workers we see a slightly imporve in the computation spped .
- N= 10^5 is not doable due to memoryerror

Strategy

SAME AS PREVIOUS

When Rank==0:

1. First We make a batch based on the data size and number of workers.
2. Rank0 always start the program by initializing the matrices.
3. Next we stack the data together to send it to other processes easily.
4. Rank0 send the stacked data to all other processes with a specific tag.
5. Rank0 itself receives the batch of the data and perform the summation.

6. Rank0 waits until other processes do their task and send the data to Rank0, as soon as it receives the data it first append to a list and at final stage it concatenate all members of the list to make the final data.

When Other Ranks:

1. Since we use Send/Recv we need to create a buffer before receiving the data.
2. The data is received from process 0 to other processes, the data is put in the buffer.
3. The summation is peformed and the result is going to be sent to the main process, but first the shape is sent to be used for size buffer for the data.
4. The result is sent to process 0

End:

- After all the processors did their task and sent their data back to master processor, the program will end

Exercise 2_

In [231…

```
%%writefile ex02.py

from mpi4py import MPI
import numpy as np

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

stat = MPI.Status()


#size of array
N=10**4
#print(size)
#print(rank)

Batch = round(N/size)

def multiplication(a,b):
  z = np.dot(a,b)
  return z

t1 = MPI.Wtime()
# master process
if rank == 0:

    data1 = np.random.random(size = (N,N))
    data2 = np.random.random(size = (N,N))

    for i in range(1, size):

        #sending the matrices
        comm.Send(data1, dest=i, tag=i)
        comm.Send(data2, dest=i, tag=i*2)
        #print(f'From Process {rank} to -> process {i}\n Data: \n{subs[i]}')


    #choosing the batch for this process
    d1 = data1[(Batch*rank):(Batch*(1+rank)),:]
    d2 = data2[(Batch*rank):(Batch*(1+rank)),:]

    #Calculating product
    vector_s = np.zeros(shape=(d1.shape[0],d1.shape[1]))
    for row in range(0, d1.shape[0]):
        for colm in range(0, d2.shape[0]):
            vector_s[row,0] += (d1[row,colm] * d2[colm,0])
    #Sending back segment of output vector C

    print(f"shape in rank {rank}, is: {vector_s.shape}")

    #receiving and doing final summation
    final_dt = [vector_s]
    for i in range(1,size):
        shp_ = np.empty(2, dtype=int)
        comm.Recv(shp_, source=i, tag=i+size*2)

        data_f = np.empty((shp_[0],shp_[1]))
        comm.Recv(data_f, source=i, tag=i+size*3)
        final_dt.append(data_f)

    final_res = np.concatenate(final_dt, axis=0)
    #print("final result: ",final_res[0])
    print("finallyyyyy....",final_res.shape)

    print("Time in rank 0 is:", MPI.Wtime()-t1)
```

```python
# worker processes

else:
    #create buffer
    #receive data
    #peform action
    #send data back

    data_1 = np.empty((N,N))
    comm.Recv(data_1, source=0, tag=rank)

    data_1 = data_1[(Batch*rank):(Batch*(1+rank))]
    #print(data_1.shape)

    data_2 = np.empty((N,N))
    comm.Recv(data_2, source=0, tag=rank*2)
    data_2 = data_2[(Batch*rank):(Batch*(1+rank))]
    #print(data_2.shape)



    #Calculating product
    vector_s = np.zeros(shape=(data_1.shape[0],data_2.shape[1]))
    for row in range(0, data_1.shape[0]):
        for colm in range(0, data_2.shape[0]):
            vector_s[row,0] += (data_1[row,colm] * data_2[colm,0])


    print(f"shape in rank {rank}, is: {vector_s.shape}")


    shp = np.array(vector_s.shape, dtype=int)
    comm.Send(shp, dest=0, tag=size*2+rank)
    comm.Send(vector_s, dest=0, tag=rank+size*3)

    #print("Time in rank {} is:".format(rank), MPI.Wtime()-t1_)
    #print('Process {} received data:'.format(rank), data_.shape)
    #print(f'\nResult sent \n From Process {rank} to -> process {0}\n Data shape: \n{re
```

```
Overwriting ex02.py
```

In [222…

```
!mpiexec -np 1 python ex02.py
```

```
shape in rank 0, is: (100, 100)
finallyyyyy.... (100, 100)
Time in rank 0 is: 0.020988941192626953
```

In [223…

```
!mpiexec -np 2 python ex02.py
```

```
shape in rank 1, is: (50, 100)
shape in rank 0, is: (50, 100)
finallyyyyy.... (100, 100)
Time in rank 0 is: 0.007315874099731445
```

In [224…

```
!mpiexec -np 3 python ex02.py
```

```
shape in rank 1, is: (33, 100)
shape in rank 2, is: (33, 100)
shape in rank 0, is: (33, 100)
finallyyyyy.... (99, 100)
Time in rank 0 is: 0.003826141357421875
```

In [225…

```
!mpiexec -np 4 python ex02.py
```

```
shape in rank 0, is: (25, 100)
finallyyyyy.... (100, 100)
Time in rank 0 is: 0.002684116363525390б
```

```
shape in rank 1, is: (25, 100)
shape in rank 2, is: (25, 100)
shape in rank 3, is: (25, 100)
```

In [116…

```
#N=10**3
```

In [227…

```
!mpiexec -np 1 python ex02.py
```

```
shape in rank 0, is: (1000, 1000)
finallyyyyy.... (1000, 1000)
Time in rank 0 is: 2.043778896331787
```

In [228…

```
!mpiexec -np 2 python ex02.py
```

```
shape in rank 1, is: (500, 1000)
shape in rank 0, is: (500, 1000)
finallyyyyy.... (1000, 1000)
Time in rank 0 is: 0.5822610855102539
```

In [229…

```
!mpiexec -np 3 python ex02.py
```

```
shape in rank 1, is: (333, 1000)
shape in rank 2, is: (333, 1000)
shape in rank 0, is: (333, 1000)
finallyyyyy.... (999, 1000)
Time in rank 0 is: 0.3003368377685547
```

In [230…

```
!mpiexec -np 4 python ex02.py
```

```
shape in rank 1, is: (250, 1000)
shape in rank 2, is: (250, 1000)
shape in rank 0, is: (250, 1000)
finallyyyyy.... (1000, 1000)
Time in rank 0 is: 0.23574185371398926
shape in rank 3, is: (250, 1000)
```

In [199…

```
#N=10**4
```

In [232…

```
!mpiexec -np 1 python ex02.py
```

```
shape in rank 0, is: (10000, 10000)
finallyyyyy.... (10000, 10000)
Time in rank 0 is: 207.29367899894714
```

In [233…

```
!mpiexec -np 2 python ex02.py
```

```
shape in rank 1, is: (5000, 10000)
shape in rank 0, is: (5000, 10000)
finallyyyyy.... (10000, 10000)
Time in rank 0 is: 62.28562092781067
```

In [234…

```
!mpiexec -np 3 python ex02.py
```

```
shape in rank 1, is: (3333, 10000)
shape in rank 2, is: (3333, 10000)
shape in rank 0, is: (3333, 10000)
finallyyyyy.... (9999, 10000)
Time in rank 0 is: 62.819145917892456
```

In [236…

```
!mpiexec -np 4 python ex02.py
```

```
shape in rank 1, is: (2500, 10000)
shape in rank 2, is: (2500, 10000)
shape in rank 3, is: (2500, 10000)

===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 25456 RUNNING AT mansoor
=   EXIT CODE: 9
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Killed (signal 9)
This typically refers to a problem with your application.
Please see the FAQ page for debugging suggestions
```

Report 2_

Table of Times for 2

N	P=1	P=2	P=3	P=4
10**2	0.0209	0.00731	0.00382	0.0026
10**3	2.0437	0.5822	0.3003	0.2357
10**4	207.293	62.285	62.819	EXIT CODE: 9

- Here we can clearly see the speed up in all the sizes with more workers
- Only when N=10^4 the program exits. -**The reason why it has speed is the way we are performing our task which is one by one. when we have vector summation/multiplication the speed is overall faster than operational summation/multiplication but by adding extra workers we don't see imporvements.**

Strategy

SAME AS PREVIOUS

When Rank==0:

1. First We make a batch based on the data size and number of workers.
2. Rank0 always start the program by initializing the matrices.
3. Next we stack the data together to send it to other processes easily.
4. Rank0 send the stacked data to all other processes with a specific tag.
5. Rank0 itself receives the batch of the data and perform the summation.
6. Rank0 waits until other processes do their task and send the data to Rank0, as soon as it receives the data it first append to a list and at final stage it concatenate all members of the list to make the final data.

When Other Ranks:

1. Since we use Send/Recv we need to create a buffer before receiving the data.
2. The data is received from process 0 to other processes, the data is put in the buffer.
3. The summation is peformed and the result is going to be sent to the main process, but first the shape is sent to be used for size buffer for the data.
4. The result is sent to process 0

End:

- After all the processors did their task and sent their data back to master processor, the program will end

In [ ]: