2. DISTRIBUTED COMPUTING WITH MESSAGE PASSING INTERFACE

Gökce SUCU

246112

EXERCISE 0: EXPLAINING THE SYSTEM

$$\begin{array}{rl} \textbf{Processor} & 1.1\ GHz\ dual-core\ Intel\ Core\ i3 \\ \textbf{Number of Cores} & 8 \\ \textbf{RAM} & 8\ GB \\ \textbf{Operating System} & macOS\ Monterey \\ \textbf{Programmin Language} & Python\ 3.8.13 \end{array}$$

EXERCISE 1: BASIC PARALLEL VECTOR OPERATIONS WITH MPI

This file just for report! I run my all codes on .py file in command line. .py codes attached seperately.

EX1. a. ADDING TWO VECTORS

EX1 a.0.Importing the Necessary Libraries

In [2]:
```
from mpi4py import MPI
import mpi4py
import numpy as np
```

EX1 a.1.Creating a Communicator

In [10]:
```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
name = MPI. Get_processor_name()

print("Name=",name)
print("Rank=",rank)
print("Size=",size)
```

```
Name= Gokces-MacBook-Air.local
Rank= 0
Size= 1
```

To be able to run more than 1 processor (for different size values) , I use command line and I run the code "mpirun -n P python file_name.py".

EX1 a.1. Defining the Necessary Functions

a.1.1 Creating n Dimensional Vector

I will create function that creates $10 * *n$ dimensional function. It will consists of integers between 0 to 10.

In [3]:
```
def n_dim_vector(n):
    v = np.random.randint(0,10,size=(10**n,1))
    return v
```

a.1.2 Splitting The Vectors into Pieces Equally For Each Process

Now, I will create a function for splitting equally the data into P (process) pieces to able to use comm.scatter() function.

In [4]:
```
def pieces_of_vector(v,process):
    list_of_pieces=list(range(process))
    vector_size= int(v.shape[0])
    numbers_in_pieces= int(vector_size/process)
    for i in range(0,process-1):
        list_of_pieces[i]=v[i*numbers_in_pieces:(i+1)*numbers_in_pieces]
    list_of_pieces[process-1]=v[(process-1)*numbers_in_pieces:]
    return list_of_pieces
```

a.1.4 Gathering Pieces from Each Process After Parallel Computing

At the beginning, I splitted the into P pieces to send each processes. After paralell calculating I need a function that gathers all calculations from each process.

In [5]:
```
def vector_from_list_pieces(list_of_pieces):
    gathered_data = list_of_pieces[0]
    process= len(list_of_pieces)
    for i in range(1,process):
        gathered_data = np.concatenate((gathered_data,list_of_pieces[i]))
    return gathered_data
```

EX1 a.3. Point to Point Communication (Scatter)

Basically, the idea is that I splitted data into P pieces (P is the value of process). Then I send this pieces orderly to each process by using comm.scattter() function. This step show how to send the each process respectively.

In [ ]:
```
if rank == 1:
    shared_data=pieces_of_vector(data,size)

else:
    shared_data = None


received = comm.scatter(shared_data, root=1) # received data from rank=1
```

EX1 a.4. Calculaton of Vector Adding and Time

Here, I coded the adding operations by using if loop in each different process. Also, I used MPI.Wtime() for calculating time differences.

In [ ]:
```
for i in range(0,size):
    if rank==i:
        start = MPI.Wtime() #timing starts
        x = received[:,0]+received[:,1] #adding the two vectors
        end = MPI.Wtime() #timing ends
        y = end-start # time difference
```

EX1 a.5. Gathering of All Calculations on Each Process

In this part, I gathered all results of each process. For this reason. Iused comm.gather() function. I also gathered all timing for each process. At the end, I have gathered_one that shows the all result of each process and it is list. I also have timing list for each process and it is also listed.

In [ ]:
```
Gathered_one = comm.gather(x,root=1)
Gathered_two=comm.gather(y,root=1)
```

EX1 a.6. Arranging The Gathered Results

In the previous code, I gathered information but I need exact solution. For this reason by using these lists and the function that I created function vector_from_list_pieces. Basically, this function creats vector from list.

In [ ]:
```
print("Gathered Addings",vector_from_list_pieces(Gathered_one))
print("Gathered Times",sum(Gathered_two),"\n")
#. mpirun -n 12 python untitled3.py
```

Also, I use sum() function to calculate total timing.

EX1 a.7 Results and Tables For Adding Two vectors

$TIMING$	$dim = 10^3$	$dim = 10^7$	$dim = 10^8$
$P = 2$	0.00001	0.0097	0.0106
$P = 4$	0.00003	0.0313	0.0322
$P = 8$	0.00006	0.04051	0.0450
$P = 12$	0.0001	0.04860	0.0552

EX1 b. AVERAGE NUMBER OF A VECTOR

Find an average of numbers in a vector I just changed the function in calculation part. I used np.mean instead of vector addition operation.

In [ ]:
```
for i in range(0,size):
    if rank==i:
        start = MPI.Wtime()
        x = np.mean(received) #changed part
        end = MPI.Wtime()
        y = end-start
```

I also had to change last part to gather all solutions.

In [ ]:
```
#i added np.mean function
print("Gathered Addings",np.mean(Gathered_one))

print("Gathered Times",sum(Gathered_two),"\n")
#.
```

EX 1 b. Results and Tables Average Numbers in the Vectors

$TIMING$	$dim = 10^3$	$dim = 10^7$	$dim = 10^8$
$P = 2$	0.00005	0.0083	0.1655
$P = 4$	0.0001	0.0085	0.2524
$P = 8$	0.0003	0.01452	0.3702
$P = 12$	0.0006	0.0226	0.4653

EXERCISE 2: PARALLEL MATRIX VECTOR MULTIPLICATION

In this example, I used same previous code. I did some minor changes. First of all, because of generating a matrix, I created a function for creating a matrix randomly.

EX2: 1. Creating a nxn Dimensional Matrix A

In [51]:
```
def n_dim_matrix(n):
    A= np.random.randint(0,10,size=(10**n,10**n))
    return A
```

EX2: 2. Creating A Function to Split A into P pieces

I also need P equally splitted small pieces of A matrix. Dimension of Each small matrix would be N/P x N. P is process value N is dimension of A. I wont send this small matrix to each process now. After adding vector v, I would send to each process.

In [52]:
```
def pieces_of_matrix(A,process):
    list_of_pieces=list(range(process))
    vector_size= int(A.shape[0])
    numbers_in_pieces= int(vector_size/process)
    for i in range(0,process-1):
        list_of_pieces[i]=A[i*numbers_in_pieces:(i+1)*numbers_in_pieces,:
    list_of_pieces[process-1]=A[(process-1)*numbers_in_pieces:,:]
    return list_of_pieces
```

EX 2. 3. Creating a Function for Creating Real Result From Gathered

After gathering all results from P process, every result would be listed on gathered_one list. The solution must be Nx1 dimension. For this reason, I need a new function to create solution from gathered results.

In [53]:
```
def concat_gathered(gathered_list):
    length  = len(gathered_list)
    concated_last= gathered_list[0]
    for i in range(1,length):
        concated_last = np.concatenate((concated_last,gathered_list[i]))
    return concated_last
```

EX 2. 4. Creating A Function For List of Shared Data

I already created P small N/P x N dimension matrix . I want to add vector v at the end of small matrices horizantally. For this reason, I need one loof for creating list of shared data.

In [54]:
```
data = []
for i in range(0,size):
    data.append(np.concatenate((pieces_of_A[i],v_one.T),axis=0))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [54], in <cell line: 2>()
      1 data = []
----> 2 for i in range(0,size):
      3     data.append(np.concatenate((pieces_of_A[i],v_one.T),axis=0))

NameError: name 'size' is not defined
```

Here, as you can see, I added v vector at the end of small A pieces by usin np.concatenate() function. Now, my list of data is ready to send P process. In my shared data list there are P list to be sended to each process.

EX 2. 5. Sending List of Shared Data

In the data list there are P element to be sent to P process orderly. Now, I would do this.

In [ ]:
```
if rank == 1:
    shared_data=data

else:
    shared_data = None

#received is the each processes received data by rank=1
received = comm.scatter(shared_data, root=1)
```

I am using comm.scatter() because it seperates orderly.

EX2. 6. Defining a Operation

Now, it is time for calculation. Each process received N/P xN dimension orderly splitted small matrices and vector v that is attacted to at the end of small matrices. Basically, I applied matrix multiplication of small matrixes and vector v.

In [ ]:
```
for i in range(0,size):
    if rank==i:
        start = MPI.Wtime()
        #received[:-1,:] I seperated small matrix from vector v
        #received[-1,:] is the vector v (last row of received matrix)
        x = np.matmul(received[:-1,:] ,received[-1,:].T)
        end = MPI.Wtime()
        y = end-start
```

EX2. 7. Gathering all Results From Each Process

Now, it is time to gather all results from each process to create gathered list.

In [ ]:
```
Gathered_one = comm.gather(x,root=1)
Gathered_two=comm.gather(y,root=1)
```

EX2. 8. Creating Exact Result

After we applied the operations, we received results and it will be listed in the gathering list. We need to tranform into Nx1 dimension vector.

In [ ]:
```
print("Gathered List",Gathered_one)
print("Result of Multiplication",concat_gathered(Gathered_one))
print("Gathered Times",sum(Gathered_two),"\n")
print("\n")

# 2
```

EX2. 9. Results and Tables Matirx Vector Multiplication

$TIMING$	$dim = 10^2$	$dim = 10^3$	$dim = 10^4$
$P = 2$	0.00002	0.0004	0.1761
$P = 4$	0.00006	0.0005	0.2297
$P = 8$	0.0001	0.0009	0.2598
$P = 12$	0.0004	0.0015	0.3020

In [62]:
```
a = np.array(range(36)).reshape((6,6))
```

In [ ]:

```

```

EXERCISE 3: MATRIX MATRIX MULTIPLICATION

in this exercise, I would create N x N dimension two matrices A and B. I would split B matrices columns into P pieces. Each peac would include N/P columns. Then I would add A matrix and column pieces of B matrix as a tuple to new shared data list. In this list, there would be P tuples. (A, B matrix columns).

Then I would send this P tuples to P process respectively. Then I would use np.matmul() function for multiplication of matrix A and B matrices pieces. Then I would ise np.cocatanate function for gathering solutions horizantilly.

EX3 1. Importing Necessary Libraries

In [1]:

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

EX3 2. Defining a Function For Creating $10^n x 10^n$ Matrix A

Now, I need a function to splitt right side matrix into columns.

In [ ]:

```
def pieces_of_matrix_columns(A,process):
    list_of_pieces=list(range(process))
    matrix_size= int(A.shape[0])
    amount_of_column_piece= int(matrix_size/process)
    for i in range(0,process-1):
        list_of_pieces[i]=A[:,i*amount_of_column_piece:(i+1)*amount_of_co
    list_of_pieces[process-1]=A[:,(process-1)*amount_of_column_piece:]
    return list_of_pieces
```

EX3 3. Creating Exact Solution By Using Gathered List

After Calculations are finished on the each process, I would gather the solutions but I need a N xN dimension matrix solution. For this reason, to create exac solution I need a function.

In [ ]:

```
def concat_gather(gathered):
    length = len(gathered)
    solution= gathered[0]
    for i in range(1,length):
        solution = np.concatenate((solution,gathered[i]),axis=1)
    return solution
```

EX3 4. Calling the Communication

In [2]:
```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
name = MPI. Get_processor_name()
```

EX3 5. Generating Matrix A and B

In [ ]:
```
N=3
A_matrix  = n_dim_matrix(N)
B_matrix  = n_dim_matrix(N)
```

EX3 6. Splitting the Matrix B into Pieces

Now, I would use the function that created for splitting the matrix into column pieces. I am doing this to send each process equall data.

In [ ]:
```
#i splitted the matrix a by rows so there are still 10**N columns.
set_of_columns_B=pieces_of_matrix_columns(B_matrix,size)
```

EX3 7. Creating a Shared Data List

Now, I would create list that has P element to send eah elements to processess respectively. This list consists of tuples and the first element of tuples are matrix A and second elements of tuples consist of EQUAL piece of B columns.

In [ ]:
```
data = []
for i in range(0,size):
    data.append([A_matrix,set_of_columns_B[i]])
```

EX3 8. Sending the Data to Processes

Here, I would send the shared data list that consists of tuples to each procesess. Each process receives one tuple data.

In [ ]:
```
if rank == 1:
    shared_data=data
else:
    shared_data = None
```

EX3 9. Received Data

Every process received tuples by usin comm.scatter() function. I used this functio because it sends data orderly to the process.

In [ ]:
```
received = comm.scatter(shared_data, root=1)
```

EX3 10. Applying the Matrix Multiplication

In this part , I would apply matrix multiplication on each processess. Combining each processess result vertially would give me the complete solution.

In [ ]:
```
for i in range(0,size):
    if rank==i:
        start = MPI.Wtime()
        x = np.matmul(received[0] ,received[1])
        end = MPI.Wtime()
        y = end-start
```

EX3 11. Gathering The Solution

Because, I calculated part of solutions on different processes, now I need to gather all results together.

In [ ]:
```
Gathered_one = comm.gather(x,root=1)
Gathered_two=comm.gather(y,root=1)
```

EX3 12. Transforming the Gathered Solution into Complete Solution

In [ ]:
```
print("Gathered List",Gathered_one)
print("Solution",concat_gather(Gathered_one))
print("Timing",sum(Gathered_two),"\n")
print("\n")
# mpirun -n 2 python untitled4.py
```

EX 13.Results and Tables Matirx Matrix Multiplication

$TIMING$	$dim = 10^2$	$dim = 10^3$
$P = 2$	0.00001	0.6623
$P = 4$	0.00003	0.6046
$P = 8$	0.0001	0.7791
$P = 12$	0.0002	0.8943

my computer couldnot calculate while 10^4.

In [ ]:

```
```