CMPE300 - Analysis of Algorithms
Spring 2020

MPI Programming Project
(Finding Armstrong Numbers)

Student Name: ALİ BATIR
Student ID: 2015400261

Submitted Person: Zeynep Yirmibeşoğlu
Submission Date: 05.06.2020

Introduction

In this project, the goal to achieve is the parallel programming with Python using MPI library. It is expected to implement a parallel algorithm for finding Armstrong numbers.
An Armstrong number is a number that is equal to the sum of its own digits each raised to the power of the number of digits. For example 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407, 1634, 8208, 9474, ... are Armstrong numbers.

Program Interface&Execution

In this project, the program finds Armstrong numbers from 1 to ARRAYSIZE(A). Therefore we need an input value which is the ARRAYSIZE.

The program has no special interface. It is used by command line and the program is executed by using mpiexec command.

The command line to execute the program on the command line should be in the format:

>> mpiexec -n <NUM_PROCESSORS> --hostfile <hostfile_text_file> python mpi.py <input-value>

Input and Output

-n <NUM_PROCESSORS>

tells MPI to use <NUM_PROCESSORS> processes. It is total number of processors, (<NUM_PROCESSORS> - 1) worker processors and 1 master processor.

--hostfile <hostfile_text_file>

by using a hostfile with the --hostfile option. The hostfile is a text file that contains the names of hosts, the number of available slots on each host, and the maximum slots on each host.

<input-value>

the value for ARRAYSIZE (A)

python mpi.py

the name of MPI Python script

Output;

armstrong.txt that contains Armstrong numbers sequentially. This is the output file which is generated after the execution of program.

Program Structure

In the main function, after taking the arguments from terminal, MPI is initialized. MPI.COMM WORLD is most commonly used communicator. Size keeps the number of total processors and rank is kind of an id for each processors.

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

The number of worker processes is (size-1).

```
workers=size-1
#get array size (A)
ARRAYSIZE=int(sys.argv[1])
#find the size that divides the array
chunksize=ARRAYSIZE/workers
```

The structure contains two main parts which are master and worker. Program is design to work with one master processor and multiple workers.

The master process will create an array of numbers, from 1 to ARRAYSIZE. The array is divided and elements are added to sub arrays. Then, sub arrays are sent to workers.

```
#master process
if rank == 0:
    #create an array of numbers from 1 to ARRAYSIZE
    array=[]
    for i in range(ARRAYSIZE):
        array.append(i+1)
    #move array elements into a different order
    random.shuffle(array)
```

```
    for i in range(workers):
        #divide the array and create subarrays
        subarray=[]
        for j in range(chunksize):
            subarray.append(array[j])
        #send subarrays to worker processes
        comm.send(subarray, dest=(i+1), tag=1)
        #remove first chunksize elements from array
        array=array[chunksize:len(array)]

    #create an array for keeping armstrong numbers
    Armstrong=[]
    for i in range(workers):
        #receive the array of Armstrong numbers from each worker
        get_armstrong_numbers=comm.recv(tag=2)
        #append all elements of get_armstrong_numbers to Armstrong
array
        Armstrong.extend(get_armstrong_numbers)

    #receive sum of Armstrong numbers from last worker
    collective_sum_result=comm.recv(source=workers,tag=5)
    print('MASTER: Sum of all Armstrong numbers:
'+str(collective_sum_result))

    #sort the array of armstrong numbers
    Armstrong.sort()
    #print armstrong numbers to armstrong.txt
    with open('armstrong.txt', 'w') as f:
        for number in Armstrong:
            f.write("%s\n" % number)
```

The worker processes receive the first partition of the sub array and find the Armstrong numbers in their partition. When they have finished processing, they will send their results to the master process, which will sort and print these results into armstrong.txt.

```
#worker process
else:
    #receive the partition of the array
    receive=comm.recv(source=0,tag=1)
    #Finding Armstrong Numbers from the array that is received
    Armstrong_numbers=[]
    sum_armstrong=0
    #control all numbers from 0 to chunksize
    #chunksize:the size of the array elements for each worker
process
    for i in range(chunksize):
```

```
        digit=0
        #take a number from subarray
        number=receive[i]
        #find out what how many digits are there in our number
        while number!=0:
            #get the number except the last digit
            number=number/10
            digit=digit+1 #increase the number of digit
        count=digit
        #take the same number from the array again
        number=receive[i]
        multiply=1
        sum=0
        #calculate the sum of the Armstrong numbers
        while number!=0:
            #compute the remainder that results from performing
integer division
            remainder=number%10
            #multiply every digit 'digit' times and sum them
            while count!=0:
                multiply=multiply*remainder
                count=count-1 #decrease the number of digit
            sum = sum + multiply
            count=digit
            #get the number except the last digit
            number=number/10
            multiply=1
        #if the number is Armstrong number
        if sum==receive[i]:
            #add the armstrong number to array
            Armstrong_numbers.append(receive[i])
            sum_armstrong=sum_armstrong+receive[i]

    print("Sum of Armstrong numbers in Process " +  str(rank) + " =
" + str(sum_armstrong))

    #send the found Armstrong numbers in its partition to Master
process
    comm.send(Armstrong_numbers,dest=0,tag=2)
```

After calculation of the sums by each process, each process should send their sum to the next process, each of which will add the received sum to their own sum, resulting in a collective sum of Armstrong numbers. The last worker sends the final sum to master, which will print the sum of the Armstrong numbers.

```
         receive_sum_armstrong=0
```

```python
    if rank!=1:
        #receive the sum of the armstrong numbers from previous
process
        receive_sum_armstrong=comm.recv(source=rank-1,tag=4)
        #add sum of armstrong numbers to collective sum
        collective_sum=sum_armstrong+receive_sum_armstrong
        #print("Collective sum of Armstrong numbers in Process " +
str(rank) + " = " + str(collective_sum))
    #if rank is not equal to rank of the last worker
    if rank!=workers:
        #each process should send their sum to the next process
        comm.send(collective_sum,dest=rank+1,tag=4)
    else:
        #The last worker sends the final sum to master
        comm.send(collective_sum,dest=0,tag=5)
```

Examples

1. A = 1000, 5 processors (n = 4)

2. A = 10000, 5 processors (n = 4)

3. A = 100000, 5 processors (n = 4)

```
alibatir@alibatir-UX330UAK: ~/Desktop/mpiPROJECT/python
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$ mpiexec -n 11 --hostfile hostfile python mpi.py 1000
Sum of Armstrong numbers in Process 2 = 370
Sum of Armstrong numbers in Process 1 = 12
Sum of Armstrong numbers in Process 3 = 0
Sum of Armstrong numbers in Process 5 = 0
Sum of Armstrong numbers in Process 4 = 0
Sum of Armstrong numbers in Process 6 = 2
Sum of Armstrong numbers in Process 7 = 943
Sum of Armstrong numbers in Process 9 = 13
Sum of Armstrong numbers in Process 10 = 0
Sum of Armstrong numbers in Process 8 = 6
MASTER: Sum of all Armstrong numbers: 1346
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$ mpiexec -n 11 --hostfile hostfile python mpi.py 10000
Sum of Armstrong numbers in Process 1 = 0
Sum of Armstrong numbers in Process 2 = 9475
Sum of Armstrong numbers in Process 3 = 8585
Sum of Armstrong numbers in Process 5 = 389
Sum of Armstrong numbers in Process 6 = 8
Sum of Armstrong numbers in Process 4 = 407
Sum of Armstrong numbers in Process 7 = 1791
Sum of Armstrong numbers in Process 8 = 0
Sum of Armstrong numbers in Process 9 = 7
Sum of Armstrong numbers in Process 10 = 0
MASTER: Sum of all Armstrong numbers: 20662
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$ mpiexec -n 11 --hostfile hostfile python mpi.py 100000
Sum of Armstrong numbers in Process 1 = 0
Sum of Armstrong numbers in Process 2 = 6
Sum of Armstrong numbers in Process 8 = 12
Sum of Armstrong numbers in Process 4 = 755
Sum of Armstrong numbers in Process 3 = 1634
Sum of Armstrong numbers in Process 7 = 55155
Sum of Armstrong numbers in Process 5 = 3
Sum of Armstrong numbers in Process 6 = 10
Sum of Armstrong numbers in Process 9 = 92880
Sum of Armstrong numbers in Process 10 = 110766
MASTER: Sum of all Armstrong numbers: 261221
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$
```

4. A = 1000, 11 processors (n = 10)

5. A = 10000, 11 processors (n = 10)

6. A = 100000, 11 processors (n = 10)

```
alibatir@alibatir-UX330UAK: ~/Desktop/mpiPROJECT/python
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$ mpiexec -n 5 --hostfile hostfile python mpi.py 1000
Sum of Armstrong numbers in Process 1 = 8
Sum of Armstrong numbers in Process 2 = 14
Sum of Armstrong numbers in Process 4 = 415
Sum of Armstrong numbers in Process 3 = 909
MASTER: Sum of all Armstrong numbers: 1346
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$ mpiexec -n 5 --hostfile hostfile python mpi.py 10000
Sum of Armstrong numbers in Process 3 = 8583
Sum of Armstrong numbers in Process 4 = 2050
Sum of Armstrong numbers in Process 1 = 177
Sum of Armstrong numbers in Process 2 = 9852
MASTER: Sum of all Armstrong numbers: 20662
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$ mpiexec -n 5 --hostfile hostfile python mpi.py 100000
Sum of Armstrong numbers in Process 1 = 416
Sum of Armstrong numbers in Process 4 = 102952
Sum of Armstrong numbers in Process 2 = 148215
Sum of Armstrong numbers in Process 3 = 9638
MASTER: Sum of all Armstrong numbers: 261221
alibatir@alibatir-UX330UAK:~/Desktop/mpiPROJECT/python$
```

armstong.txt

The Armstrong numbers between 1 and 100000

```
armstrong.txt
Open
1
2
3
4
5
6
7
8
9
153
370
371
407
1634
8208
9474
54748
92727
93084
```

Conclusion

In conclusion, this project was a good exercise for parallel programming and it has taught me about how to use MPI technologies. I experienced how to use MPI library, send and receive functions with their parameters.