ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

CMPE 300 - Analysis of Algorithms
Fall 2022

MPI Programming Project

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

INTRODUCTION

In this project we are trying to estimate the probability of occurrence of a word duo can given a dataset.

What we aim to do by finding the probability can be regarded as what is happening on Google search engine. When we type a word, it recommends the most likely word that we might want to type next.

We utilize mpi4py and sys framework. We use mpi4py to separate the tasks among different processors, sys to read input from command line.

Program Interface

Firstly, the user should install the mpi4py library to execute this program, which can be done with “pip install mpi4py” command.

The user communicates with the program via command line arguments. There are command line arguments for user to interact with.

-n -> represents number of processes

—input_file -> represents input file for the main.py

–merge_method -> represents merge method for the program. It can be “WORKERS” or “MASTER

–test_file -> represents the test file which contains the bigrams that are wanted to calculate their possibility

```
mpiexec -n 5 python main.py --input_file sample_text.txt --
merge_method MASTER --test_file test.txt
```

Program Execution

We can’t really claim that there is a user interface or a menu in this projec. On top of that, we do not aim to serve to an end-user. We can simply use this project to find probability distributions of bigram and unigrams of given text dataset. Thus, we can predict the most probable word after a specific word using given text dataset. The answer to question `How can we use it` is clearly explained in Program Interface section of the report.

Input and Output

There are two input files for this project. One that is taken with input file argument is a preprocessed text document with a single sentence per line made up of only words. The other input file that is taken with the test file argument contains a bigram per line. The probability calculations are requested for these bigrams.

There is no output file for this project. The program prints its output to the console. It prints the number of lines per process and the requested probabilities.

Program Structure

In the following code we retrieve inputfile’s name, merge method and test file’s name and assign them to variables:

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

```
if "--input_file" in sys.argv:
    # Get the index of the argument
    index = sys.argv.index("--input_file")
    # Get the value of the argument
    inputfile = sys.argv[index + 1]



if "--merge_method" in sys.argv:
    # Get the index of the argument
    index = sys.argv.index("--merge_method")
    # Get the value of the argument
    merge_method = sys.argv[index + 1]


if "--test_file" in sys.argv:
    # Get the index of the argument
    index = sys.argv.index("--test_file")
    # Get the value of the argument
    testfile = sys.argv[index + 1]
```

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

REQUIREMENT 1

```
#divides lines to processes
def divider(lines,noOfProcesses):

    length=len(lines)
    sendList=[[] for i in range(noOfProcesses)]

    for i in range(length):
        lines[i]=lines[i].rstrip()

        sendList[i%noOfProcesses].append(lines[i])
    return(sendList)
```

```
if rank == 0:
    with open(inputfile, "r", encoding="utf8") as f:
  # Read the contents of the file
        lines = f.readlines()

    # REQUIREMENT 1
    sendList = divider(lines,size-1)

    # sends data to workers
    for i in range(1,size):
        comm.send(sendList[i-1], dest=i, tag=11)
```

Divider function takes a list called lines and number of processes as an input. Creates a list called sendList which consists of noOfProcesses empty lists. For loop assigns lines to the empty lists inside of sendList in an approximately equal manner.

In the 2nd code block, inputfile is read. The variable lines is a list of lines in the input file, and size is the number of processes (Master+Workers).

They’re given to divider function as parameters. Then in a for loop, using comm.send function, they’re sent to the workers.

REQUIREMENT 2

```
# REQUIREMENT 2
#receive and merge dicts
if merge_method=="MASTER":
    bigramDict={}
    unigramDict={}
    for i in range(1,size):
        data = comm.recv(source=i, tag=11)
        unidictWorker =data[0]
        bigramDictWorker =data[1]
        for i in unidictWorker.keys():
            if(i in unigramDict):
                unigramDict[i] += unidictWorker[i]
            else:
                unigramDict[i] = unidictWorker[i]

        for i in bigramDictWorker.keys():
            if(i in bigramDict):
                bigramDict[i] += bigramDictWorker[i]
            else:
                bigramDict[i] = bigramDictWorker[i]
```

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

```
#Worker processes
else:
    #receives and counts
    data = comm.recv(source=0, tag=11)
    print("rank:", rank, "number of sentences:", len(data))

    unidictWork=countUni(data)
    bigramDictWork=splitBi(data)

    if merge_method=="MASTER":
        #sends dictionaries together
        comm.send([unidictWork, bigramDictWork], dest=0, tag=11)
        # comm.send(bigramDictWork,dest=0, tag=11)
```

In this requirement we run the code in MASTER mode as merge method.

1st picture: bigramDict and unigramDict are the main dictionaries of master process. Master process receives the returned data via comm.recv function. Then using 2 for loops, we kept the record of unigram and bigrams.

2nd picture: On the worker process side, workers get the data from master process. We can conclude this from source parameter of comm.recv function which is 0. And as it can be inferred from the code block inside if condition, the data is directly sent back to the master process. They are not sent in a manner like requirement 3.

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

REQUIREMENT 3

```
#Worker processes
else:
    #receives and counts
    data = comm.recv(source=0, tag=11)
    print("rank:", rank, "number of sentences:", len(data))

    unidictWork=countUni(data)
    bigramDictWork=splitBi(data)
```

```
elif merge_method=="WORKERS":
    if rank==1:
        comm.send([unidictWork,bigramDictWork],dest=rank+1,tag=11)

    for i in range(2,size):
        if rank==i:
            data=comm.recv(source=rank-1,tag=11)
            unidict=data[0]
            bigramDict=data[1]
            for i in unidict.keys():
                if(i in unidictWork):
                    unidictWork[i] += unidict[i]
                else:
                    unidictWork[i] = unidict[i]

            for i in bigramDict.keys():
                if(i in bigramDictWork):
                    bigramDictWork[i] += bigramDict[i]
                else:
                    bigramDictWork[i] = bigramDict[i]

            if rank!=(size-1):

                comm.send([unidictWork,bigramDictWork],dest=rank+1,tag=11)

            else:
                comm.send([unidictWork,bigramDictWork],dest=0,tag=11)
```

Here, the concept is sort of different. All the worker processes still get the data from Master process, however, the delivery system is a bit different.

Worker process of rank 1 finishes what it is doing* and sends the data to the next process, whose rank is one more.

There are 2 important things to notice, the last process in this queue sends the data to master process, not to another worker process.

*Using countUni and splitBi functions, unigrams and bigrams are counted and sent.

```
elif merge_method=="WORKERS":
    data = comm.recv(source=size-1,tag=11)
    unigramDict=data[0]
    bigramDict=data[1]
```

REQUIREMENT 4

```
#calculation of conditional probabilities
with open(testfile, "r", encoding="utf8") as f:
    tests=f.readlines()
for i in tests:
    i=i.rstrip()
    unigrams=i.split()
    print(i,":",bigramDict[i]/unigramDict[unigrams[0]])
```

And at the probability calculation stage, we read the lines of the testfile, which includes bigrams, and assign the result to a list variable called tests. Then we iterate over the tests. Loop variable i represents bigrams. In the print statement, bigramDict[i] represents number of occurrences of that specific bigram. UnigramDict is a dictionary that keeps the record of number of unigrams for each unigram. This calculation is nothing but numberofOccurrences of that bigram/ numberofOccurrences of the first word of that bigram.

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

Example

```
mpiexec -n 5 python main.py --input_file sample_text.txt --
merge_method MASTER --test_file test.txt
```

With this command, the program works with 5 processors (1 master, 4 workers), It counts unigrams and bigrams in the “sample_text.txt”. Then, It will use the MASTER merge method to merge the dictionaries from workers. Lastly, It will calculate the possibility of bigrams assuming the first word of the bigrams is given.

P(skills|new) for “new skill”

Improvements and Extensions

We could have written code in a more modular way but still it works fine.

Difficulties Encountered

Downloading and installing mpi4py was problematic but we dealt with it. Writing a detailed report is not something we’re used to.

Conclusion

As we concluded from the outputs, this is an effective approach to estimate which word is most likely to come next. Also, we concluded that parallel programming speeds up the compilation at a considerable amount.

APPENDICES

```
from mpi4py import MPI
import sys

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

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

```
if "--input_file" in sys.argv:
  # Get the index of the argument
  index = sys.argv.index("--input_file")
  # Get the value of the argument
  inputfile = sys.argv[index + 1]



if "--merge_method" in sys.argv:
  # Get the index of the argument
  index = sys.argv.index("--merge_method")
  # Get the value of the argument
  merge_method = sys.argv[index + 1]


if "--test_file" in sys.argv:
  # Get the index of the argument
  index = sys.argv.index("--test_file")
  # Get the value of the argument
  testfile = sys.argv[index + 1]


#counts unigrams for a worker
def countUni(linesOfWorker):
    unidictWorker={}

    for i in range(len(linesOfWorker)):
        liste =linesOfWorker[i].split()

        for i in liste:
            if(i in unidictWorker.keys()):
                unidictWorker[i]+= 1
            else:
                unidictWorker[i]=1
    return unidictWorker

#count bigrams
def splitBi(linesOfWorker):
    bigrams=[]
    bigramDict={}

    for i in range(len(linesOfWorker)):
        splitted = linesOfWorker[i].split()
        #print(splitted)
        for i in range(len(splitted)-1):
            bigram=splitted[i]+" "+splitted[i+1]
            #print(bigram)
            bigrams.append(bigram)
            if(bigram in bigramDict.keys()):
                bigramDict[bigram]+=1


            else:
                bigramDict[bigram]=1
    return bigramDict


#divides lines to processes
def divider(lines,noOfProcesses):
```

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

```
    length=len(lines)
    sendList=[[] for i in range(noOfProcesses)]

    for i in range(length):
        lines[i]=lines[i].rstrip()

        sendList[i%noOfProcesses].append(lines[i])
    return(sendList)



if rank == 0:
    with open(inputfile, "r", encoding="utf8") as f:
  # Read the contents of the file
        lines = f.readlines()

    # REQUIREMENT 1
    sendList = divider(lines,size-1)

    # sends data to workers
    for i in range(1,size):
        comm.send(sendList[i-1], dest=i, tag=11)


    # REQUIREMENT 2
    #receive and merge dicts
    if merge_method=="MASTER":
        bigramDict={}
        unigramDict={}
        for i in range(1,size):
            data = comm.recv(source=i, tag=11)
            unidictWorker =data[0]
            bigramDictWorker =data[1]
            for i in unidictWorker.keys():
                if(i in unigramDict):
                    unigramDict[i] += unidictWorker[i]
                else:
                    unigramDict[i] = unidictWorker[i]

            for i in bigramDictWorker.keys():
                if(i in bigramDict):
                    bigramDict[i] += bigramDictWorker[i]
                else:
                    bigramDict[i] = bigramDictWorker[i]

    elif merge_method=="WORKERS":
        data = comm.recv(source=size-1,tag=11)
        unigramDict=data[0]
        bigramDict=data[1]


    #calculation of conditional probabilities
    with open(testfile, "r", encoding="utf8") as f:
        tests=f.readlines()
    for i in tests:
        i=i.rstrip()
        unigrams=i.split()
        print(i,":",bigramDict[i]/unigramDict[unigrams[0]])
```

ONUR DİLSİZ - 2019400036
ANIL KÖSE – 2019400060

```
    #print(unigramDict)


#Worker processes
else:
    #receives and counts
    data = comm.recv(source=0, tag=11)
    print("rank:",rank, "number of sentences:",len(data))

    unidictWork=countUni(data)
    bigramDictWork=splitBi(data)

    if merge_method=="MASTER":
        #sends dictionaries together
        comm.send([unidictWork,bigramDictWork],dest=0, tag=11)
        # comm.send(bigramDictWork,dest=0, tag=11)



    elif merge_method=="WORKERS":
        if rank==1:
            comm.send([unidictWork,bigramDictWork],dest=rank+1,tag=11)

        for i in range(2,size):
            if rank==i:
                data=comm.recv(source=rank-1,tag=11)
                unidict=data[0]
                bigramDict=data[1]
                for i in unidict.keys():
                    if(i in unidictWork):
                        unidictWork[i] += unidict[i]
                    else:
                        unidictWork[i] = unidict[i]

                for i in bigramDict.keys():
                    if(i in bigramDictWork):
                        bigramDictWork[i] += bigramDict[i]
                    else:
                        bigramDictWork[i] = bigramDict[i]

                if rank!=(size-1):

                    comm.send([unidictWork,bigramDictWork],dest=rank+1,tag=11)

                else:
                    comm.send([unidictWork,bigramDictWork],dest=0,tag=11)
```