**Cmpe300: Analysis of Algorithms Fall’22 Project2**

**Student Names:**

* **Abdullah Susuz**
* **Umut Demir**

**Submitted Person: Umut Demir**

**Title of the Project: MPI Programming Project**

**Submission Date: 20/12/2022**

**Introduction**

In this project, we are expected to calculate the data for a bigram and unigram language model. For calculation, we use MPI framework and mpi4py python library for our calculations in a parallel way.

MPI:“MPI is a standardized and portable message-passing standard designed to function on parallel computing architectures.”, (from [wikipedia](https://en.wikipedia.org/wiki/Message_Passing_Interface))

**Program Interface**

In order to run this program, one has to have a computer that runs python3 including mpi4py library and mpi framework. This program can be run with mpiexec command. Also there are some parameter like the level ol parallelism, name of the input files and name of the merge method. If there is any error, the program ends itself. In addition, command+c or ctrl+c can be used to terminate the program in terminal.

**Program Execution**

As stated above, this program can be run with command line interface. The execution is the program is as follows:

mpiexec -n <num> python3 main.py --input\_file <input-file> --merge\_method <method> --test\_file <test-file>

For example:

Text

Description automatically generated

<num>: It is the number of ranks minus one. That is, <num> states the number of the parallel processes.

<input-file>: This file includes sentences. It is used to calculate the number of unigrams and bigrams.

<method>: There are two different methods for merging the operations. This part states the name of the method, which is etiher WORKERS or MASTER.

<test-file>: This file includes some bigrams. These bigrams are used for probability calculations.

**Input and Output**

As I stated above, one has to have a number for stating the number of parallel processes, a method name that is etiher WORKERS or MASTER, an input file for calculating bigrams and unigrams, finally a test file for probability calculations.

An example for executions is as follows:

Text

Description automatically generated

It means that I have 4 parallel “worker”s for calculating unigrams and bigrams, sample\_text2.txt file for data file for unigrams and bigrams, use MASTER method for merging the results, and test.txt for calculating probabilities of bigrams from this file.

The result of this execution is as follows:

Text

Description automatically generated

First, sentences from sample\_text2.txt is distirbuted to 4 workers. Then number of unigrams and bigrams are printed to the terminal. Other outputs is probability of bigrams from test.txt file.

**Program Structure**

The structure of the program changes according to the method name taken as an argument. In the MASTER method, all unigrams and bigrams are calculated when the data distributed by the master is processed by the workers and returned to the master. If the method is WORKERS, each worker processes the data received from the master and the data from the previous worker and sends it to the next worker. The last worker sends all the processed data to the master and the master does not need to process any data. When these processes are finished, the master saves unigrams and bigrams in the dictiyonary structure. In this dictionary, the key value is unigram or bigram, and value is the number of times it appears in the input\_file. Unigrams and bigrams are communicated in the same way but with different tags. Then, the conditional probabilty of the bigrams given in test\_file is found with the formula of the total number of the given bigram / the total number of the first word in the given bigram.

**Examples**

The results obtained with different merge methods for bigrams in test\_file after training with a large data:

Text

Description automatically generated

**Improvements and Extensions**

Seperating operations for unigrams and bigrams may cause the program to slow down. If these processes can be collected at once or under a single data, the program will provide a significant increase in speed. While we were only transferring unigram and bigram numbers to meet the initial requirements at the beginning of the project, we were using list structure instead of using dictionary structure, then we used dictionary since we need conditional probability later on. The program's loyalty to its object oriented structure enabled significant code readability and work sharing.

**Difficulties Encountered**

First of all, working with mpi4py library on linux systems caused certain problems. The first problem we encounter may be the installation of the library and the problems due to environment. Then, obtaining all unigrams and bigrams together with their number and sending them to a different node both made it difficult to track large data and the parallel computation structure made it difficult to debug. Apart from these, the project generally went as we planned and our plan ended properly.

**Conclusion**

We have created a model that converges to a language processing model by providing a parallel computing structure and data exchange between nodes. This model predicts the conditional probability of new bigrams that are trained with the given data. We used mpi4py library to set up data exchange and parallel computing structure. We have set up this structure in two different ways according to the user's preference. The common feature of these structures is that they have a master node that collects and distributes the data evenly, then calculates the probability, and they have worker nodes that process this data and send the unigram and bigram numbers to the required place.

**Appendices**

<https://mpi4py.readthedocs.io/en/stable/tutorial.html>

<https://web.stanford.edu/~jurafsky/slp3/3.pdf>

<https://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiintro/ppframe.htm>