CS426

Fall 2017

Project 2

Due November 13th 2:59PM

1. Problem Definition

In this project, you will implement a parallel document search system. The algorithm that you will implement is similar to Supervised Search. However, it is a very simple version of it.

In our system, each document i is represented with a weight vector w_i. A weight vector w_i is composed of D number of elements such that each of the weight values $w_{i,j}$ corresponds to the relationship between the file i and word j in our dictionary, and D is the dictionary size. D is only an integer here, we do not have an actual dictionary.

Set of documents W which contains F documents can be represented as follows:

$$W = \{ w_i \mid w_i \ is\ a\ vector\ and\ w_{i,j} \geq 0\ and\ w_{i,j}\ is\ an\ integer \}\ where\ 0 \leq i \leq F\ and\ 0 \leq j \leq D$$

Then, we will insert some queries to the system. A query Q has same vector definition as a document vector w_i.

$$Q = \{ q_j \mid q_j \geq 0\ and\ q_j\ is\ an\ integer \}\ where\ 0 \leq j \leq D$$

Finally, your program will take $W, Q, D\ and\ K$ as input and will give top K related documents as output according to given *similarity function*.

Set of files will be given as an input file to your system. Each line of the input file contains id of document, consecutive D integers which are seperated by a single space. Format of a line is as follows:

Doc_id: weight_0 weight_1 ... weight_d

Example line for document with id 5 and a system with dictionary size 3

5: 0 5 3

And query will also be given with an input file. However, it does not contain an id field.

E.g. for a query with a dictionary size 3

2 0 7

Now, we will define our *similarity function*. Similarity of a document i to a query Q, s_i , is defined as follows:

$$s_i = \sum_{j=0}^{D} w_{i,j}^{q_j}$$

E.g. we can calculate similarity value for previous weight vector and query pair as follows:

$$0^2 + 5^0 + 3^7 = 2188$$

To summarize, you will read documents file and query file. You will find similarity values for each document and output most related K documents' ids.

2. Implementation Details

First you will implement a *kreduce* function with the following function prototype.

*void kreduce(int * topk, int * myids, int * myvals, int k, int world_size, int my_rank)*

kreduce function is a collective communication function like MPI_Bcast or MPI_Reduce. All processes will call it, however only the master process will have k ids that correspond to top k values in descending order according to their corresponding values. Meaning that *topk* is only used by master process. Each process pass two integer arrays to this function. *Myids* array keeps ids of documents of a process and *myvals* array keeps similarity values for these documents. *k* tells that how many top elements will be selected. Last two parameters of this function is optional and their names are self explanatory. Sizes of myids and myvals arrays are equal to k. And the values in myvals array is sorted in descending order.

Let say, we have 2 processes running and their inputs to this function as follows:

k = 3

Process 0 (Master):

My_ids = [5, 6, 7]

My_vals = [10, 8, 3]

Process 1 (Slave):

My_ids = [1, 2, 11]

My_vals = [17, 9, 8]

And as an output of this function, process 0 should have the following values in topk array:

Topk = [1, 5, 2]

You are not allowed to use any collective communication operations outside of this kreduce function. The only collective communication operation that you can use in main part of your program is kreduce

function and you have to use kreduce function in your main program. Better implementation of kreduce function will get better grades.

3. Output

You will give outputs for three things. First you will printout the computation time for sequential part of your program e.g. time for reading files. Secondly, you will printout the computation time for parallel part of your program e.g. calculating top k documents for the query. Lastly, you will printout top k ids. Output format should be as follows:

```
Sequential Part: 1.50 ms

Parallel Part: 5.22 ms

Total Time: 7.12 ms

Top k = 3 ids:

5

2

9
```

4. Submission

Your submission will include:

- Your code:
  - utils.h, utils.c files which are implemented for reading input files.
  - main.c implementation of the program.
  - Your code should be compiled with the following command:
    - mpicc –o documentSearch main.c utils.h utils.c –lm
  - Your executable should be able run with the following command:
    - Mpirun –n X ./documentSearch dictionarySize kValue documents.txt query.txt
  - Your code should include explanatory comments.
- Put several example outputs of your program. Place all output examples in directory named as **outputs**. Each output example should include some information about input parameters of your program.
- Run your code with various parameters. Decide which parameters affects the performance of your program. Disccuss your reasoning for choosing these parameters in your report.
- Your report:
  - **Very detailed explanation of implementation of kreduce function.**
  - Explanation of your main program.
  - **Plot several graphs** for choosen parameters. (**Hint:** There are at least three different parameters that will affect your programs performance.)

    - Discussion of selected parameters and and performance results according to these parameters. Remember that you have also taken sequential part execution and parallel part execution times. How can you interpret those values?
- Email to kaan.akyol@bilkent.edu.tr
    - A single zip file **yourname_lastname_p2.zip**
    - Email subject is **CS426_Project2**
    - **Any submission with wrong email subject or file name will not be graded .**
- **Bonus (+5 points):** Write your report with LaTeX.
    - In this case you should include .tex files in your submission.