LAB DISTRIBUTED DATA ANALYTICS I
TUTORIAL I

Juan Fernando Espinosa Reinoso
303158
Group 1: Monday Tutorial

Note: Increasing the number of cores increases the time of execution because internally it is working with just 2 processes. Hence, oversubscribing them just split the actions between and wait until the previous action is executed.

EXERCISE 0

Processor	2,5 GHz Intel Core i7
Operating system	macOS High Sierra
Number of cores	2
RAM	8 GB 2133 MHz LPDDR3
Graphics	Intel Iris Plus Graphics 640 1536 MB
Programming Language Version	Python 3.7
Storage	250,69gb

EXERCISE 1 - BASIC PARALLEL VECTOR OPERATION WITH MPI

a. Add to vectors and store them in a third one

Size of vectors	Workers	Time	Observations
10**7	1	2.553081	
	2	2.640388	
	4	5.236301	
	6	5.838849	Memory error appears. (MPI perf. degradation)
	8	----	Help messages from processes. Memory error.

For the remaining vector sizes (10 **12 and 10**15) no result is printed due to RAM.

Structure of the code:

There are 3 main keys in the code:

1. *Create_vectors*: It is a function that takes a N number chosen and automatically originate a random vector of size N.
2. *Add_vectors:* Function that receives the vectors and apply the element-wise summation of the datapoints in each vector. It returns the **time** it takes to do the operation as well as a **final vector** of datapoints.
3. *Main MPI structure*: First, my approach considers worker 0 as a worker able to make operations. Decision made after checking the limited number of cores my laptop has. Therefore, the strategy is as follows:
    - Split the data in equal chunks. In consequence, getting a number of datapoints inside each chunk is optimal. In my approach it is called **p_workers.**
    - Splits both vector A and B according to the number of datapoints in each chunk. Worker 0 will take the first chunk of data and all the remaining data will be processed by the other workers.
    - To send the data to each worker in chunks a for loop is created from the first worker (exclude 0) to the last one and splitting the data considering the ith worker and the value of **p_workers.**

b. Take the average from vectors

Size of vectors	**Workers**	**Time**	**Observations**
10**7	1	6.506804	
	2	4.483055	
	4	9.084105	
	6	10.09127	Memory error appears. (MPI perf. degradation)

The structure of the code is exactly the same as exercise 1a with one major difference: Instead of the add_vector function it is replaced by the average_vector function and the main action inside it. Where the average of each vector value is taken and storage in a new vector.

EXERCISE 2 - PARALLEL MATRIX-VECTOR MULTIPLICATION USING MPI

Size	**Workers**	**Time**	**Observations**
10**2	1	0.003559	
	2	0.00367	

	3	0.0056430	
	4	0.005938	
	6	0.0065320	Memory error appears. (MPI perf. degradation)
10**3	1	0.321667	
	2	0.37157700	
	3	0.562415	
	4	0.741119	
	6	0.7617590	Memory error appears. (MPI perf. degradation)
10**4	1	33.978607	
	2	36.316707	
	3	52.626838	
	4	71.190176	

Structure of the code:

The code is presented in a similar manner as the previous one. It contains 3 pillars from which the model works around.

- *Create_data:* A function from which the matrix $A \in R^{NxN}$ and vector b of $A \in R^{N}$ are created with random values.
- *Matrix_vector:* Function that receives as inputs matrix A and vector b. The matrix-vector multiplication is executed and it returns the time it takes to execute the operation as well as a list with the outputs of each datapoint.
- *Main MPI structure:* The process is quite similar regarding the previous exercise with one important difference: Here the matrix will be splitted in several chunks (depending of the number of workers) but the vector must be send to all the chunks (broadcast). This is because each row of the matrix depends on all the data points in the vector.
  The splitting of Matrix A through the workers is applied the same way as in exercise 1.

EXERCISE 3 - PARALLEL MATRIX OPERATION USING MPI

Size of vectors	**Workers**	**Time**	**Observations**
10**2	1	0.534843	

	2	0.549064	
	3	0.8226229 9	
	4	1.084759	
	6	1.530485	Memory error appears. (MPI perf. degradation)

For the remaining matrix sizes (10 **3 and 10**4) no result is printed due to RAM/Memory.

Structure of the code:

As I found useful to create functions the process is repeated:

- *Create_data:* A function from which two matrices $A \in R^{NxN}$ and $B \in R^{NxN}$ are created with random values.
- *MatrixMult:* Function that receives as inputs matrix A and B. The matrix opeation is executed and it returns the time it takes to execute the multiplication as well as a a matrix $C \in R^{NxN}$ with the outputs of each operation.
- *Main MPI structure:* worker 0 creates the matrices calling the function above mentioned. Matrix A will be splitted in chunks in a row manner while matrix B will be completely shared with all workers: Each row of matrix A needs all columns of matrix B.
  The splitting of Matrix A through the workers is applied the same way as in exercise 1 and 2.