Faculty of Computers and Artificial Intelligence
Cairo University
High Performance Computing

Assignment 3 – MPI & OpenMPI

Deadline & Submission:

1. Teams: Form a team of 4 students from the same group or the same TA.
2. Upload it on Classroom with file named: A3_ID1_ID2_ID3_GroupName.zip
eg. A3_20130003_20130002_20130001_S1_S2.zip
3. Attach a screenshot from the console output for each problem in the zip folder.
4. Cheating could lead to serious consequences.

Problem 1: Election to choose a new president (using MPI)

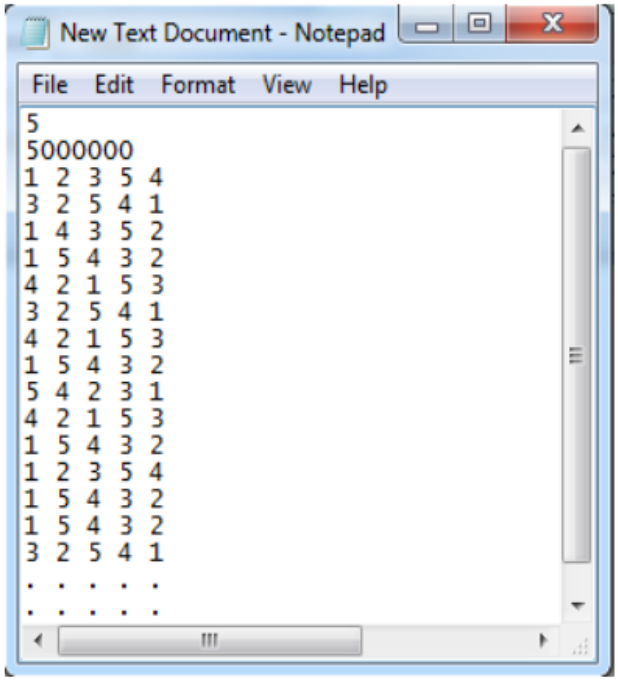
Here are the rules for the election process:
1. There are C candidates (numbered from 1 to C), and V voters.
2. The election process consists of up to 2 rounds. All candidates compete in the first round. If a candidate receives more than 50% of the votes, he wins, otherwise another round takes place, in which only the top 2 candidates compete for the presidency, the candidate who receives more votes than his opponent wins and becomes the new president.
3. The voters' preferences are the same in both rounds, and each voter must vote exactly once in each round for a currently competing candidate according to his preferences.
Given the preferences lists, you need to write a program to announce which candidate will win and in which round.

For example:
If the input
3 5 // number of candidates & number of voters
1 2 3 // voter 1 preference list
1 2 3 // voter 2 preference list
2 1 3 // voter 3 preference list
2 3 1 // voter 4 preference list
3 2 1 // voter 5 preference list
Then the output will be 2 2 // candidate 2 wins in round 2

And if input
2 3 // number of candidates & number of voters
2 1 // voter 1 preference list
1 2 // voter 2 preference list
2 1 // voter 3 preference list
Then the output will be 2 1 // candidate 2 wins in round 1

You must use files so firstly generate by the code file which contains the voters' preferences, and the format of the file must be number of candidates in the first line and number of voters in the second line, and then followed by voters' preferences equal to number of voters.

Note: when running the program the file must not be loaded by one process but every running process should loads its part.



Run Steps you must follow: When run the program prompt the user to generate file like above of voter's preferences or to calculate the result and if the user choose the second option enter the filename as input.

The Output: Print to the console the steps happening in every process and print which candidate will win the elections and in which round. And if there are 2 rounds identity that also and show the percentage of every candidate per each round.

Problem 2: search in array (using OpenMP)

Write a C program to search for a key in a 2D array. Matrix dimensions and the key we will search for will be taken as input, initialize the matrix with random numbers, share the matrix with all processes and return the indices of the key, if key is not found return -1.

Note: Must use dynamic allocation

Input: row, column, key
Output: array containing the indices the key appeared at.

Parallelization Process:

-Use " #pragma omp parallel for " directive to divide the search loops iterations among processes.
-every time the key is found display which process found it at what index and add that index to an array let's say "indxArr" and print that array.

Problem 3:Calculating Standard Deviation (using OpenMP)

Write a parallel c program to calculate standard deviation using

Given:

An integer n (number of elements per process).

Output:

Standard deviation of randomly generated (n * numberOfProcesses) elements.

How to Calculate Standard Deviation:

1. Calculate the Mean. (sum of n elements/ n)

2. For each number, subtract the mean and square the result. ((element - mean)2)

3. Calculate the mean of the squared differences. ($sum(\ (element\ -\ mean)2)\ /\ n$)

4. Take the square root of step three.

Parallelization Scenario:

- Get **n** from user and make it shared.

In the parallel part of the code will be the following

- Generate n random elements.

- Calculate local sum of the generated n elements.

- Calculate local mean of the generated n elements. (local sum / n)

- Calculate local sum of squared differences from the mean. Sum((element - mean)^2)

- Add local sum of squared differences to global sum. globalSum += localSumOSqD

- Calculate the global mean. (global sum / n * numOfProcesses).

- Calculate the square root of the global mean.