Parallel Programing

Assignment 2

deadline: 25.02.2024 23:59

Submission Guidelines

To reduce likelihood of misunderstandings, please follow these guidelines:

1. Work can either be done individually or in pairs.
2. Submission is through the moodle (lamda) system.
3. Make sure that your solution (for part III) compiles and runs without any errors and warnings on BIU servers.
4. In the first line of every file you submit, write in a comment your id and full name. For example: "/* 123456789 Israela Israeli */".
5. Not using parallel computation (i.e <thread>/<pthread.h> libraries for part III, and parallel algorithms for parts I,II) in even one task will result in an automatic 0 for the grade. In other words, **sequential code is unacceptable.**

Bonuses

You may add a question suggestion for HW for next semester. The question should be interesting to solve. There is no need to solve the question, but it needs to be reasonably solvable for the homework #2 of the course. Top 3 ideas will get a point in the final grade of course.

General Background

The goal of this exercise is to practice both parallel thinking, and parallel programming (using c/c++ and <pthread.h>/<thread>)

This exercise is composed of 3 unrelated parts. The first 2 parts are about theoretical parallel algorithms, and the third part is about parallel coding in c\c++ (particularly for graphs).

Environment & Submission

As we will discuss, makefile examples for part III will be given under makefiles_III.zip in the moodle (lamda).
As we stated before, each part of this assignment uses different source files.
We will specify which files you need to submit.

For parts I, II:
You will need to submit a pdf file named parts_I_II.pdf with the solution for parts I, II.
For parts III:
You will need to submit your code for part 3 in a zip file named part_III.zip
This zip file must include all your code, and your makefile for the code (as explained in part III itself, you can use one of the 2 given makefiles, or modify them to suit your needs). Please note that your makefile has to compile your code to a file named "part_III.out".

Your zip file must be organized in such a way, that when using the unzip command (in the terminal of the biu planet servers), all the files must be located directly inside the same folder that contains the zip file. In other words, after opening a terminal in the folder that contains your zip file, and then running:

```
$ unzip part_III.zip
$ ls
```

The ls command has to print all your files, and not just list a directory where all your files are located.

Make sure that after unzipping your file (as explained above) on the biu planet servers, then your file compiles with the "make" command with no errors, and after compiling, make sure that your code runs (on the biu planet servers).

Part I - QuickSort

In this part, we'll think about a faster QuickSort algorithm, using parallel computation. We already talked about O(n) (average time complexity) QuickSort, and the limiting factor was the partition algorithm's O(n) running time. So in this part you will need to suggest a partition algorithm that utilizes parallel computing, and computes the partition of an array for a given pivot in O(log(n)) time.

We require you to provide a theoretical parallel algorithm (i.e pseudo code), so you don't have to submit an implementation. We also require a time complexity analysis for your algorithm, and an explanation for the correctness of the algorithm (this explanation doesn't have to be a formal proof).

Part II - 15 puzzle game

The rules of the game

The game consists of a 4 by 4 grid of numbers, where one of the squares is empty. The rest of the squares are filled with numbers from {1,2,...,15} where each number shows up exactly once.
To solve the game, you can slide the squares that are next to the empty slot, to the empty slot. And to solve the game, you need to reach the following state:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	empty square

Here is a link to the wikipedia page about the game (so you can make sure you understand the rules):
https://en.wikipedia.org/wiki/15_Puzzle

The assignment

In this part, we'll think about how to solve the puzzle using parallel computation.
We require you to provide a theoretical parallel algorithm (i.e pseudo code) that solves the puzzle. Please note that you don't have to submit an implementation for the algorithm. We also require a time complexity analysis for your algorithm, and an explanation for the correctness of the algorithm (this explanation doesn't have to be a formal proof).

Part III - BFS implementation

Now, we require you to implement a parallel BFS algorithm for undirected graphs, which calculates the shortest path matrix for the given graph (note: to represent infinity, mark the distance as -1).

The graph will be inputted exactly like the DFS algorithm from practical session number 5. We recommend you to test your code using input redirection from a file, so you won't have to input a graph each time.

The signature for the function should be as follows:

```
void bfs(Graph *graph, int **m);
```

Where graph is the given graph, and m is the shortest path matrix you need to update its values.

For this, you can write your code in either c or cpp. And you will need to submit a makefile. We will provide 2 makefiles that you could use (although you could also provide your own makefile), one for c, and the other for cpp.

Please note that you must use either <pthread.h> or <thread> as your thread library. Please also use a thread pool, you are permitted to use the code from the 5th practical session.

Note: you may change the SyncGraph implementation to suit your needs better, as long as the program still compiles and runs fine. However, you cannot change the implementation of main.c