Assignment 3

Introduction to Parallel Programming

due **Tuesday 15 October 2019, 23:59**
(time from which "late days" start counting)

Instructions

As in the previous assignments, you will need to register in a group on Studentportalen. The group can be different than your previous ones, but, once again, we suggest that you find another student to work together and form a group. In case of trouble, please contact the assistant (nima.ghoroubi@it.uu.se).

Submission checklist:

- Submissions must clearly show your name(s).
- Submit a **single** PDF report, as well as a **.zip** file with all source code for your solutions to the exercises in Studentportalen.
- Solutions must be in C or C++ with OpenMP, as specified in the exercises.
- All source code must compile and run on the **Linux lab machines running Ubuntu**. For the list of the Linux servers that you can use, refer to https://www.it.uu.se/datordrift/maskinpark/linux.
- **Provide instructions for compilation and running, preferably by including Makefile(s).**
- No source code modifications should be required for reproducing your results.
- Your report must describe the theoretical concepts used as well as all relevant details of your solution.

In case you do not reach a working solution, describe the main challenges and proposals to address them. Please keep your answers short and concise, but clear and complete.
Also, note that some of the exercises ask you to do benchmarking on the Lab machines and this requires 1) using the machine at a time when it is lightly loaded, and 2) taking multiple measurements to see if there is any variation. Thus, it's not a good idea to leave this part of the assignment for too close to the deadline!

Some OpenMP Tutorials

The Web contains a myriad of tutorials for OpenMP, in many languages and for many languages! Some of them in English for C and C++ are:

- The "Hands-on Introduction to OpenMP" video tutorial in 27 parts (www.openmp.org/resources/) by Tim Mattson, which also comes with exercises (www.openmp.org/resources/tutorials-articles/).
- OpenMP (https://computing.llnl.gov/tutorials/openMP/).
- Guide into OpenMP: Easy multithreading programming for C++ (bisqwit.iki.fi/story/howto/openmp/).
- OpenMP FAQ (http://www.openmp.org/about/openmp-faq/).

There are many more. Be aware that these resources contain much more information than you will need for this assignment. Still, we suggest you study them a bit and experiment with some OpenMP annotations and runtime functions before you start on the exercises below.

Exercise 1: Sieve of Eratosthenes (2 points)

The previous assignment asked you to use Posix threads to implement a parallel version of the Sieve of Eratosthenes algorithm for finding prime numbers and also suggested a strategy to parallelize your program. This exercise asks you to use OpenMP instead of Posix threads for the parallelization of your solution. If you choose to work in the same (one or two person) group as before, you need to use the program of your previous submission as a basis for this one. If you decide to form a *new* group with another student for this assignment, you can choose *one* of your previous submissions as basis (state which one you used in your report). If you have *not* submitted a program for this exercise of assignment 2, you can of course write an OpenMP solution from scratch.
Besides your code, you need to provide a short section in your report that explains how you modified your solution and reports the speedup curve you get as the number of cores is increased. Was the OpenMP version easier or more difficult to write? Are the speedups you get the same better or worse (and why)? Refer to the second assignment for more information about how to benchmark your program.

Exercise 2: Conway's Game of Life (4 points in total)

Conway's Game of Life takes place in a two dimensional array of cells. Each cell can be empty (dead) or full (alive) representing the existence of a living organism in it, that can switch from one state to the other one *once* during some particular time interval. In every such time period (called a *generation* or a *step*), each cell examines its own state and that of its neighbours (right, left, up, down and the neighbouring cells in its two diagonals) and follows the following rules to update its state:

- If a cell has fewer than two live neighbours it dies of loneliness.
- If a cell has two or three live neighbours it lives on to the next generation.
- If a cell has more than three live neighbours it dies of overpopulation.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The initial array constitutes the *seed* of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed—births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a *tick*. The rules continue to be applied repeatedly to create further generations.
File `Game_of_Life.c` contains a sequential implementation of the game; it takes as arguments the size of the array and the number of generations (steps) in the game.
The exercise asks you to convert this program into a parallel one using OpenMP and conduct experiments to measure the performance of your program as the number of cores increases for array sizes 64×64, 1024×1024, and 4096×4096 using 1000 and 2000 steps.
Submit your code, your speedup curves (x axis should be the number of cores/threads, y axis the speedup you get) for your measurements and a brief report with your findings and comments.
As you can easily discover, the Web contains plenty of OpenMP implementations of Conway's Game of Life. We have many of them, but *we want your own!*

Exercise 3: Matrix Multiplication in OpenMP (3 points in total)

Consider the matrix-matrix product code given in slide 16 of Lecture 7 (`07-OpenMP.pdf`). Your task is to implement it and test it. Use the `OMP_NUM_THREADS` environment variable to control the number of threads and plot the performance with varying numbers of threads. Consider three cases in which (i) only the outermost loop is parallelized; (ii) the outer two loops are parallelized; and (iii) all three loops are parallelized. What is the observed result from these three cases? Submit your code and a brief report with your experiments and comments.

Exercise 4: Sparse Matrix Product in OpenMP (3 points in total)

Consider a sparse matrix stored in the *compressed row format* (you may find a description of this format on the web or any suitable text on sparse linear algebra).
Write an OpenMP program for computing the product of this matrix with a vector.
Download sample matrices from the Matrix Market (`http://math.nist.gov/MatrixMarket/`) and test the performance of your implementation as a function of matrix size and number of threads. As in the previous exercises, submit your code and a brief report with your experiments and comments.

Good luck!