NOVA SCHOOL OF SCIENCE & TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

Concurrency and Parallelism 2022-23

Lab 04: An OpenMP Implementation of the *Game of Life*

Hervé Paulino & Alex Davidson

March 2023

Abstract

In this lab class you will aim to write a concurrent implementation of the "Game of Life". The aim of this lab is to improve your knowledge and skills on C programming, and on the development of parallel programs using OpenMP.

1 Introduction

The Game of Life[1][2] is a *zero-player game*, that was invented in 1970 by the British mathematician John Horton Conway. Conway developed an interest in a problem which was made evident in the 1940s by mathematician John von Neumann, who aimed to find a hypothetical machine that had the ability to create copies of itself, and was successful when he discovered a mathematical model for such a machine with very complicated rules on a rectangular grid. Thus, the Game of Life was Conway's way of simplifying von Neumann's ideas. It is the best-known example of a cellular automaton, in which rules are applied to cells and their neighbours in a regular grid. Martin Gardner popularised the Game of Life by writing two articles for his column "Mathematical Games" in the journal Scientific American in 1970 and 1971.

You worked with OpenMP in the previous lab. In this lab you are expected to take a larger and more complex program and parallelise it with OpenMP.

2 Task: Implementation of Game of Life

2.1 Sequential version of the *Game of Life*

Clone the given version of a sequential implementation of the Game of Life on your device by following the link

`https://classroom.github.com/a/rJHKLBd0`

and compile it in the terminal using the command `make`. `make` is a command that builds a project by following the rules given in a project specification text file named `Makefile`. Open

[1] `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`
[2] `http://web.stanford.edu/~cdebs/GameOfLife/`

the text file `Makefile` with your favourite text editor. You can search the Internet to learn more about `make` and `Makefile`, you'll need it again in the near future.

This project depends on the `libpcre` library. In Linux you can install it with

```
apt intall libpcre3-dev
```

and in macOS with

```
brew install pcre
```

and in Windows you may have to download it directly from GitHub using this link: `https://github.com/PCRE2Project/pcre2`.

The given `Makefile` is rather versatile and should work for both Linux and macOS. If you adapt it for Windows, please share your adaptations in Piazza.

2.2 OpenMP version of the *Game of Life*

The given version of the Game of Life is sequential. **Your job is to make a new parallel version of this program using OpenMP!**

Please follow these steps:

1. Compile and experiment with the given version. Carefully study (and make sure you understand) the given source code.
2. Change the program to include a new optional flag -p (from *pause*), which receives an integer as argument, and will make a pause for the given number of milliseconds. This will slow down the simulation.
3. Change the program to include a new optional flag -f (from *final*) so that only the last board (final state) is printed. This is useful to compare the final board with colleagues.
4. Change the program to include a new optional flag -s (from *silent*) that will only print the board configuration and the time it took for the program to execute, separated by tabs (`\t`) to be easily imported as a CSV file.
5. Create a parallel version of the program by using OpenMP. This version must accept a new optional flag -t (from threads), which receives an integer as argument, to control the number of OpenMP threads to use.
6. Experiment your parallel version with different boards, board sizes and number of processors.

Please remember to use git appropriately. Learn to work with branches and make a branch when you start a new phase from the list above, and merge your branch to your main version when the development is finished and appropriately tested/validated.

3 Task: Performance Questions

As you have done previously, you should create plots of your programme's performance across a number of runs, when altering the size of the initial board, and the number of generations. Furthermore, you can also use the tools explained previously such as `perf`, `gprof`, and `pprof` to produce profiles of your implementation. You can then use appropriate visualisations (such as Call Graphs and Flame Graphs) to display which parts of your programme are the most performance-intensive.

3.1 Some questions to think about

Once you have created your performance visualisations, you can answer the following questions.

- What is the achieved speed-up from parallelisation?
- Is there a relation between the board size and the speed-up achieved?
- What is the impact of using the `-O3` flag in the speed-up?
- Which functions take the longest to execute?
- What alternative methods produce more efficient results?

Acknowledgments

The text from the Introduction is an adaptation from the text in `http://web.stanford.edu/~cdebs/GameOfLife/`.