departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Concurrency and Parallelism 2017-18
Home Work 1

Objectives

By doing this homework you will learn how to design and implement parallel programs using the Cilk+ programming language, and how to evaluate the quality and impact of your design options and implementation in the performance of your program.

Introduction

In this homework assignment, you are asked to parallelise the implementation of a C program that plays the Othello (aka Reversi) game. You may check an online version of this game at: https://www.mathsisfun.com/games/reversi.html

You are given a full C program that compiles and works, and when it runs prints the sequence of board from the initial to the states. But the implementation you are given is sequential. You are expected to study that sequential version and, once you understand how it works, develop a parallel version using the Cilk+ programming language.

After the parallelisation, you are expected to run some tests and understand how and when your parallel version performs better than the sequential version.

Executing the program

The program has no mandatory command line arguments, but accepts multiple options that control the execution mode. Some of the options take arguments. The general syntax for executing the program is:

```
othello [options]
```

where options can be any of:

Option	Semantics
-c	Colorize — Use colours to differentiate between Red and Blue when printing the board.
-s	Silent — Do not print the board, just the final results.
-h	Animation — Play in animation mode.
-d <milisec>	Delay — Pause <milisec>after each player turn.
-b <bsize>	Board size — Define the bard size as a square with <bsize>X<bsize> cells.
-t <nthreads>	Number of threads — Execute the program with <nthreads> threads.

Workplan

Clone the source GIT repository and crate a branch

Start by cloning the source GIT repository and then compiling and executing the given program by using the following commands (below, "gXX" represents your group number, e.g., "g07"):

```
git clone https://joaomlourenco@bitbucket.org/cp201718team/hw1.git
git branch gXX
git checkout gXX
make
./othello -c -d 100 -b 7
```

Note: if you want to quickly learn the basic concepts of Git branching, check the following link: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Check the effect of each of the flags

Try to use different combination of flags, like:

```
./othello -c -d 10
./othello -c -b 10
./othello -s -b 75
./othello -b 10
./othello -b 10 -t 5
```

As expected and you probably noticed, the flag '-t' is taking no effect in the behaviour of the program.

Inspecting the source code

Open the source code in your favourite editor (e.g., eclipse) and study it. Almost for sure you will find some things which are new for you or, at least, you don't understand them. Ask the teacher about them.

Decide how to parallelise the program

The program can be parallelised in a few different ways. Once you decide how you want to parallelise the program, discuss your idea with your teacher in the classroom. Explain what you plan to do and what you think you will have to change / re-implement for it to work.

Implement the parallel version using cilk+

Do your implementation.

Re-run your program

Now that you have a parallel implementation of the Othello game, try again the examples from above and confirm your code is correct.

Add the option "-x" to for timing the program execution

With this option active, the only output of the program should be a line with the following format (use a TAB as separator):

```
boardSize nThreads timeMiliseconds finalRed  finalBlue
```

Get some results for the execution time with different number of threads

Once you confirmed your implementation of the parallel version is correct, make a set of tests to collect the raw performance of the parallel version, using only the "-x" flag active and varying the number of threads.

To reduce the impact of "noise" when measuring the execution time (i.e., the processor(s) got busy doing something else and not executing the program), each test should be executed multiple times. If you are using linux, you can generate the multiple execution of the program from the command line as (this is just an example):

```
for n in 1 2 4 8; do
    for i in 1 2 3 4 5; do
        ./othello -s –b 100 –t $n;
    done;
done | tee results.txt
```

The above command will test the transpose program with 1, 2, 4 and 8 threads, and each test will be executed 5 times before stepping into the next test.

If you find it interesting, you may add yet another "for" to vary the board size.

Think at least twice on which tests do you want to make:

- Which tests (how many threads) will you try? 1, 2, 4 and 8 as in the example? Or something else? Do so experiments before deciding and collecting the "final" results.
- How many times should each test be executed? How will you process the obtained results? (Hint… think about the Olympics and what happened in gymnastic contests!)

Submission

Instructions on what and how to submit you project will be given soon in a new version of this document.