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. |
| **-a** | 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. |
| **-n <nthreads>** | Number of threads — Execute the program with <nthreads> threads. |
| **-t** | Timing — Measure the run time of your program. |

# Workplan

## Clone the source GIT repository and crate a branch

Start by using the web interface of your favourite git hosting site to import the original repository for Home Work 1 to your own account, naming your new private copy as below:

**cp-2017-18-gXX-hw1**

where "gXX" is the group number, available at:

https://docs.google.com/spreadsheets/d/114aBEtMMUFCiUZDKoTLhntoM-whJicoKO9sEN2BGl_Og/edit#gid=0

To import the repository you can use the following links for GitHub and BitBucket:

https://help.github.com/articles/importing-a-repository-with-github-importer/

https://confluence.atlassian.com/get-started-with-bitbucket/import-a-repository-861178561.html

Next, clone your new repository to create a local copy in your machine.

```
git clone URL_OF_YOUR_REPOSITORY
```

You are strongly advised to work on a branch instead of your base repository:

```
git branch gXX
git checkout gXX
```

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

Then compile and run the example program with:

```
make
./othello -c -d 100 -b 7
```

## 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.

Remember to "#include <cilk/cilk.h>" in your source files that use cilk services.

To limit the number of cilk workers to 4 threads (for example), call:

```
__cilkrts_set_param("nworkers", "4");
```

## 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!)

## Evaluation

Your submission will be evaluated considering:

— The quality of the work as described in the submitted report (80%)
— The quality of the submitted code (15%)
— The implementation of a multi-level min_max algorithm (15%). *The current implementation of the AI uses the algorithm min_max with only one level of depth.*

Please note that the highest grade possible is 110%, which corresponds to 22 points. So if your grade is higher than 20 it will be lowered to 20.

## Report

You are expected to submit an original written report obeying to the following specifications:

1) Use a4paper size with all margins at 3cm;
2) Use font Times New Roman or similar at size 11pt;
3) Have at most 5 pages, including:
   - Up to 3 pages with text;
   - Up to 2 pages with charts/graphics;
   - Up to 1 page with Acknowledgments and Bibliography.
4) The report shall follow the following general specification, which should be adapted to better fit your needs and mind set;
   a) Abstract — A very short summary of your problem, your solution and your results.
   b) Introduction — A short section describing what is the context? What is the problem? What is it interesting and challenging in the problem? What is the general approach/strategy you followed to have the program executing correctly multi-threaded?
   c) Approach — A longer section describing your solution. How did you split the work among threads? What parallel pattern(s) did you use? What problems did you identify when parallelizing the given code? How did you address/solve them? IMPORTANT: by reading this section it must be clear how your solution works, making it also clear why it works. After reading this section, another programmer should be able to re-implement your solution without looking at your source code.
   d) Validation — A short section describing which techniques did you use to validate your solution. How did you test your program? Why did the tests convince you that your solution was correct and well implemented? What would happen otherwise?
   e) Evaluation — A longer section describing which correctness and performance tests did you make (avoid tables and use charts/plots, unless the precise numeric values are important). You do not need many tests neither many charts/plots. You need to interpret/explain well the results you got (i.e., the behaviour you observed).
   f) Conclusions — A sort sections answering the following questions (or others you may find interesting). Did you got the results you were expecting? How does your validation and evaluation sections confirm that?
   g) Acknowledgments — Did you use some code snippet available on the web? Cite the source. Did get any help from someone other than your group mate? Please give him/her credit for it (the name, number, and what kind of help was given.)

h) Bibliography — Which references did you use for preparing your work and the report. Bibliographic references must be cited somewhere in the text!

5) It is recommended that you write your report in LATEX (recommended means recommended, not obligatory!)

## Submission

If you submit a buggy version (e.g., that does not compile, that stops abnormally with a run-time error, that does not respect the game rules, etc) will be penalized. So…

**Before submitting remember to check carefully that your program is working correctly and following all the rules of the game (and of the heuristic used).**

### How to submit your project?

One of the group members created a project in BitBucket or GitHub by replicating the original repository. This repository must be *shared with both teaching staff members*, whose IDs are:

BitBucket — joaomlourenco, bernymac

GitHub — joaomlourenco, bernymac

Your repository must contain both the source code of your solution and the project report.

### What to submit?

The repository must have the following structure:

```
cp-2017-18-gXX-hw1 -+-> README.txt
                    +-> report.pdf
                    +-> src/
```

Where:

- `cp-2017-18-gXX-hw1` — is your repository name (gXX = group number, e.g., “g27”).
- `README.txt` — is a text file containing the name and number of both elements in the group.
- `report.pdf` — a PDF with your original written report.
- `src` — a directory (folder) with your source code.