

Parallel and Distributed Computing

**Report 1st Project**

**Parallel Sudoku using OpenMP**

Professor Luís

Semester – 2017/2018

05/04/2018

Group 9:

81731 – Catarina Aleixo

81074 – Manuel Reis

- Bernardo Amaral

1. **Introduction**

The main goal of this project is to develop both a serial and a parallel program of a Sudoku solver in C, using OpenMP.

Both implementations are based on a simple search algorithm, that uses backtracking in order to find the solution for a given Sudoku.

1. **Serial implementation**

The program that was developed receives a file as input and first fills a game board that corresponds to the Sudoku given. There is a structure called Board that contains a unidimensional array, the square size of n and the size n, where the square size is given by the first line that is read from the given file. The game board is represented by the unidimensional array of type Cell that is another structure that represents each cell of the board and contains three integers: value, fixed and minPoss. The value is a number between 1 and n for the fixed cells and 0 for the unfixed ones, fixed is a flag that states if a certain cell has a fixed value (for instance, the values different from 0 that are read from the file are fixed) and minPoss represents the first value that a certain cell can have.

After assigning each cell of the game board with the value read from the file, it will be done an update on each unfixed cell where the minPoss value is defined.

Then is called a function solveSudoku() that is where the implementation of the Sudoku solver is done. In order to take advantage of the openMP tool it was implemented a solution based on a brute force algorithm, using backtracking, which means that the program will try to assign a possible value for each cell (between 1 and n), and then it will be tested if there’s no conflict in the corresponding row, column and box calling the checkValidity() function.

First, it’s called the bruteforce() function. There it will be constructed a stack that will hold game boards. The stack has a certain limit number of boards that it is established to be n. Once it reaches n boards, the program exits from that function and returns to the solveSudoku().

* **bruteforce():** in that function, the program looks for unfixed cells and when it founds one, tries to assign all values from minPoss to n.

In each try, checks if that value is valid in the game board calling the function checkValidity(). If it’s a possible value, then pushes that board into the stack, which corresponds to the original board with that value added on the actual unfixed cell.

When the value of that cell reaches n, then the program checks if the stack is empty or not. If the stack isn’t empty, then pops a board from it, once the next unfixed value will be searched into the board that is popped from the stack.

This is repeatedly done until one of the following cases happen: there’s no board on the stack or the number of boards in the stack reached the limit.

There’s also another particular case that happens when the program reaches the last cell, which means that it was already found a solution for the given Sudoku.

After returning from the bruteforce() function, the Sudoku will be solved using backtracking. The algorithm is simple: until the stack isn’t empty or the solution isn’t found, pops a board from the stack and tries to fill the unfixed cells with valid numbers between minPoss and n. If there’s no valid number for a certain cell, then it will be done a backtracking to the last unfixed cell. The program pops another board when there’s no possible value for the first unfixed cell of the corresponding board.

1. **Parallel implementation using OpenMP**

In the parallel implementation, the solveSudoku() function is parallelized. The main idea is to assign each board in the stack to a different thread. Each thread will apply the brute force algorithm with backtracking to the respective board and the first one that finds a solution will abort the other threads.

Finally the solution is returned to the main that will call the printBoard() function that will print the Sudoku solved.