TÉCNICO LISBOA

Parallel and Distributed Computing

2017/2018

Project Assignment

Parallel Sudoku

v1.0 (2018/03/15)

The purpose of this project is to enable you to gain experience in parallel programming on an SMP and a multicomputer, using OpenMP and MPI, respectively. For this assignment you are to write a serial and two parallel implementations of a Sudoku solver.

Problem Description

You are to write an algorithm to solve the popular puzzle Sudoku. For those unaware of this game (*e.g.*, students coming from Mars...), the puzzle consists of a partially filled matrix $n \times n$. Your algorithm needs to fill the blank positions with values 1 through n such that no number is repeated on each of the n rows, n columns, or the n squares of $\sqrt{n} \times \sqrt{n}$ cells that split the original matrix.

	9	3	6	2	8	1	4	
	6						5	
	3						9	
	5						7	
	4						6	
	8						3	
	1	7	5	9	3	4	2	

2	7	1	9	5	4	6	8	3
5	9	3	6	2	8	1	4	7
4	6	8	1	3	7	2	5	9
7	3	6	4	1	5	8	9	2
1	5	9	8	6	2	3	7	4
8	4	2	3	7	9	5	6	1
9	8	5	2	4	1	7	3	6
6	1	7	5	9	3	4	2	8
3	2	4	7	8	6	9	1	5

Figure 1: Sudoku puzzle and corresponding solution, with $n = 9$.

Important note: please do not spend time developing sophisticated heuristics for this problem, the objective is to implement a simple, clean search algorithm. What we are looking for are clever parallelization strategies. To allow a fair evaluation of these strategies, the three implementations (serial, OpenMP and MPI) should be based on the same search algorithm.

Part 1 - Serial implementation

Write a serial implementation of the above algorithm in C. Name the source file `sudoku-serial.c`.

Input

Your programs should accept one command line argument which will be the name of a file with the Sudoku instance matrix. The format of this file is:

- one line with one integer, $l = \sqrt{n}$, $2 \leq l \leq 9$ (specifying l avoids the square root in the code...).
- n lines, each with n integers, separated by a space, with values in the interval $[0, n]$, where 0s represent blank positions in the matrix.

Output

The program should output (to the `stdout`) a matrix in the same format as the input:

- n lines, each with n integers, separated by a space, with values in the interval $[1, n]$.

The non-zero values of the input should be kept untouched, no zero values should be present and the matrix should conform with the rules of Sudoku.

In case no solution exists, the program should output "`No solution`".

Example

Sample input:

```
2
0 0 3 1
3 0 0 0
4 2 0 0
1 3 0 2
```

Expected output:

```
2 4 3 1
3 1 2 4
4 2 1 3
1 3 4 2
```

Part 2 - OpenMP implementation

Write an OpenMP implementation of this Sudoku algorithm, with the same rules and input/output descriptions. Name this source code `sudoku-omp.c`.

Be careful about synchronization and load balancing.

Part 3 - MPI implementation

Write an MPI implementation of the Sudoku algorithm as for OpenMP, and address with the same issues. Name this source code `sudoku-mpi.c`.

Besides synchronization and load balancing, you will need to take into account the minimization of the impact of communication costs. If you see fit, you can also consider an implementation that combines both OpenMP and MPI.

What to Turn in, and When

The project will be submitted in two parts, and each part is due on the dates indicated below. For each part, a single ZIP file should be submitted, containing the source code for the serial and the corresponding parallel version, as well as a short report. Please be aware that the deadlines are firm and that any projects submitted after the due date will not be accepted. There will be no exceptions.

1st due date (serial + OMP + report): **April 6th**, until 17:00.

2nd due date (serial + MPI + report): **May 18th**, until 17:00.

The ZIP file to be submitted should contain: the two source code files, named as indicated above, as well as a short, objective, report (1-2 pages maximum) in PDF format, named `report-omp.pdf` (1st submission) or `report-mpi.pdf` (2nd submission). The reports should cover the following topics:

- present experimental times to run the parallel versions on input data that will be made available (for 1, 2, 4 and 8 parallel tasks), the speedups, as well as other performance metrics that you see fit;
- explain what decomposition was used;
- discuss what were the synchronization concerns and why;
- describe how was load balancing performed;
- discuss the performance results, and if they are what you expected.

Ideally, the serial version should be the same on both submissions. However, if after the 1st submission, you find major errors or inefficiencies in it, you can correct them and submit a different serial version with the MPI parallel version. Obviously, the MPI parallel version should be based on that new serial version. That should be clearly stated in the report.

Important note: Your project will be tested in the first lab class just after the due date of each submission, therefore you should attend such classes, otherwise your submission will not be graded. If, for a valid reason, it would be completely impossible for you to attend that lab class, you should contact your instructor in advance and try to arrange a different slot in another lab class for that purpose.

Grading

The following aspects will be taken into consideration when grading your project, not necessarily in this order:

- correctness, i.e. if your program always produces a correct solution;
- performance, both in terms of time (speedup), memory footprint and scalability;
- source code organization, simplicity and clarity;
- originality.

The project will only be graded in the end of the semester, during a discussion with all the students involved. Each of the students involved in a given project can end up with a different grade, according to their demonstrated knowledge about the code that was submitted.