Assignment Report: Game Of Life

Federico Finocchio

Contents

1 Introduction

The assignment was about implementing three different versions of Game Of Life to visualize speedup and scalability of the parallel versions.
All versions are implemented using a *matrix-like* data structure.

The versions implemented are:

- **gol-seq.cpp**: sequential implementation;
- **gol-par.cpp**: parallel implementation using only C++ threads from **thread** library;
- **gol-omp.cpp**: parallel implementation using OpenMP **parallel for** construct.

Next sections will walk through main implementation choices, execution/testing instructions and results about speedup and scalability.

2 Implementation

All implemented versions share the main board structure, however some changes were necessary to implement the parallel computation, in particular the thread-related implementation required the major changes.

2.1 Program versions

Main implementation choices are the following:

- **gol-seq.cpp**: sequential implementation using *matrix-like* data structure (actually *std::vector<std::vector<int>>*).
  At each iteration of the main loop, every cell in the board (i.e. every element in the innermost vector) is updated using values from surrounding cells (-1, 0, +1 horizontal/vertical).
  Parallel implementations are based on distributing work from the outermost vector (i.e. sub-sets of rows are distributed evenly among workers);

- **gol-par.cpp**: parallelism is implemented assigning ranges of values to each worker. Ranges are computed starting from the board size, this means that parallelism degree is bounded by the total number of rows in the board.
  At the end each worker will work on a sub-set of rows of the main board.;

- **gol-omp.cpp**: parallelism is implemented using **omp parallel for** construct and work assignment is delegated to omp using the **schedule(dynamic)** clause.

2.2 Command Line Parameters

All the parameters must be specified to run each version.
The parameters to be specified are:

- **nrows**: total number of rows the board must contain;
- **ncols**: total number of columns the board must contain;
- **iters**: how many times the status of the board will be updated;
- **seed**: seed to be provided to the random number generator;
- **wait_time**: used for debugging, specify how much time to wait before computing the next board update;
- **nworkers**: required only in parallel versions, specify the total number of workers.

3 Execution and Testing

This section is about testing and executing the different versions.
At this purpose a **Makefile** is provided, and it can be used to compile and test the code.
Various shell scripts are provided to run tests and save statistics about each execution, in particolar:

- **test.sh**: script used to run a fixed number of times the Game Of Life's **version** provided in input.
  It produces two auxiliary files, **test-<version>.txt** containing raw execution outputs and **out.txt** containing completion time of each execution.
  At the end a file **<version>-res.txt** is created (or updated) containing the results from the last execution.
- **stats.sh**: script used by **test.sh** to compute the average between all executions. It prints the resulting average.

To compile the code you can simply run:

```
$ make all
```

To run tests you have to specify which kind of version you want to use (using different make targets) and what are the parameters of the execution using the **ARGS** variable, like:

```
$ make test−par ARGS='100 200 50 70 0 32'
```

In this case we are requiring to run the thread-related version using a board with 100 rows, 200 columns, 50 total iterations, 70 as seed for random cell generation, 0 wait time (only for debugging) and 32 as parallelism degree.

4 Results

This section will provide various plots about speedup and scalability regarding the thread-related and omp implementation.
All tests were run using 50 board updates and with 70 as seed.

4.1 Speedup and Scalability: gol-par.cpp

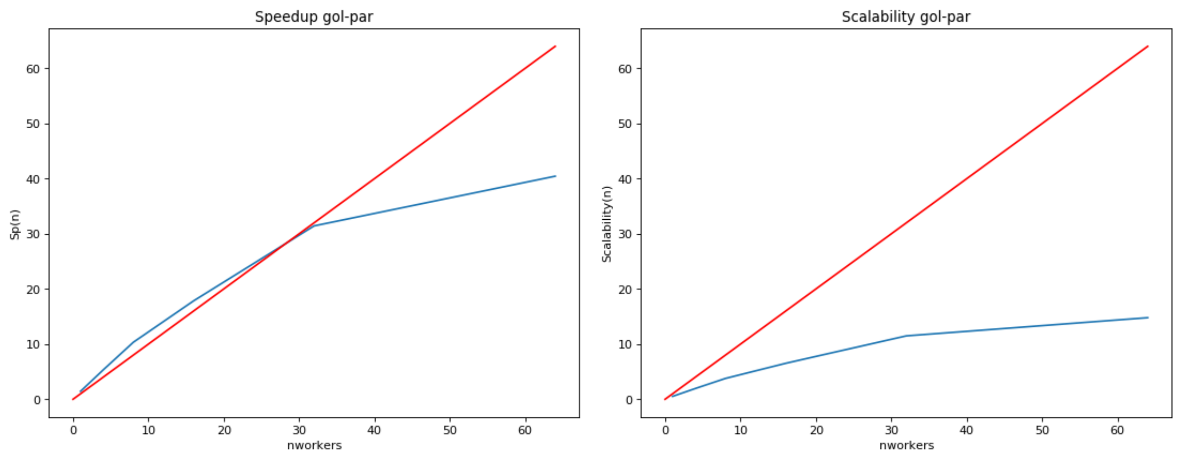

Figure 1: Speedup(left) and Scalability(right) gol-par on 200x200 board.

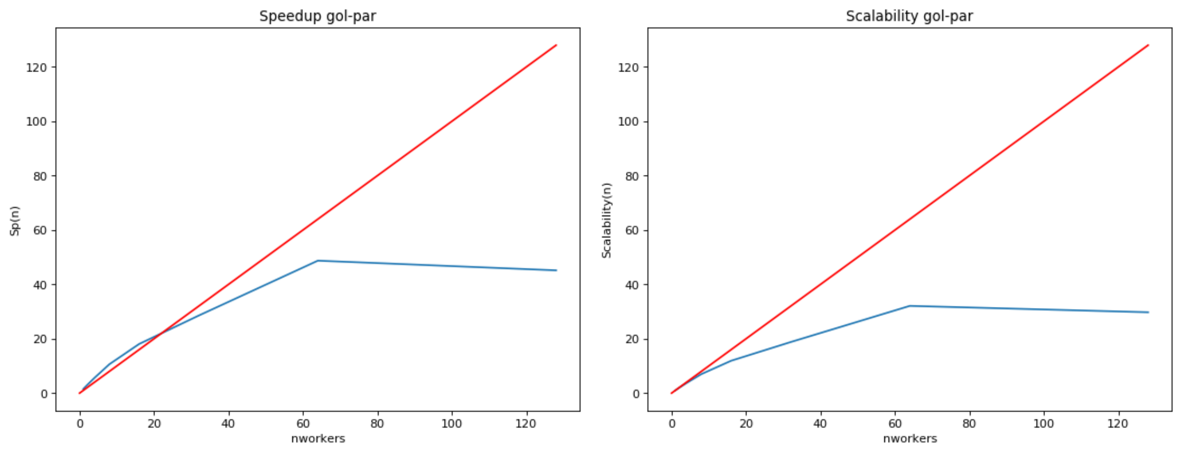

Figure 2: Speedup(left) and Scalability(right) gol-par on 300x200 board.

4.2 Speedup and Scalability: gol-omp.cpp

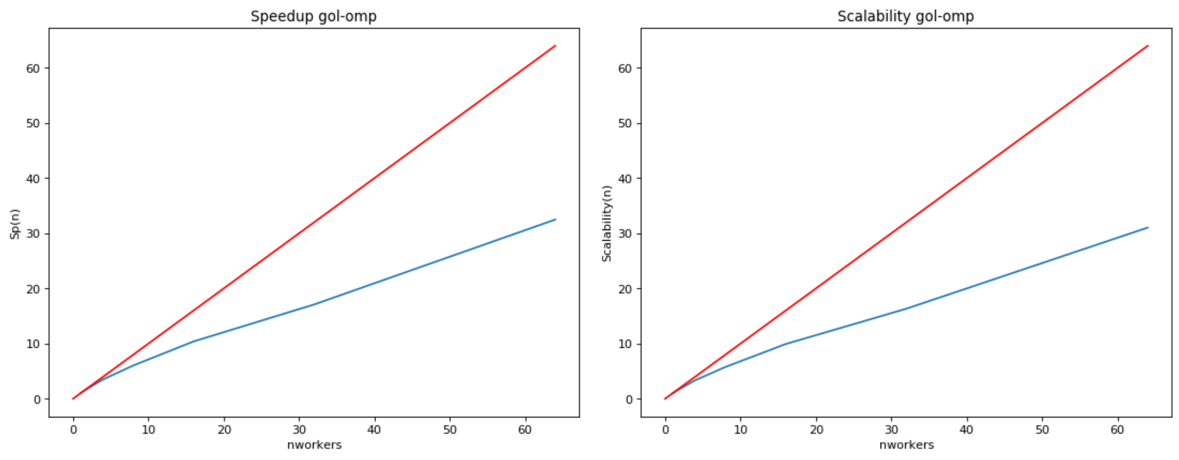

Figure 3: Speedup(left) and Scalability(right) gol-omp on 200x200 board.

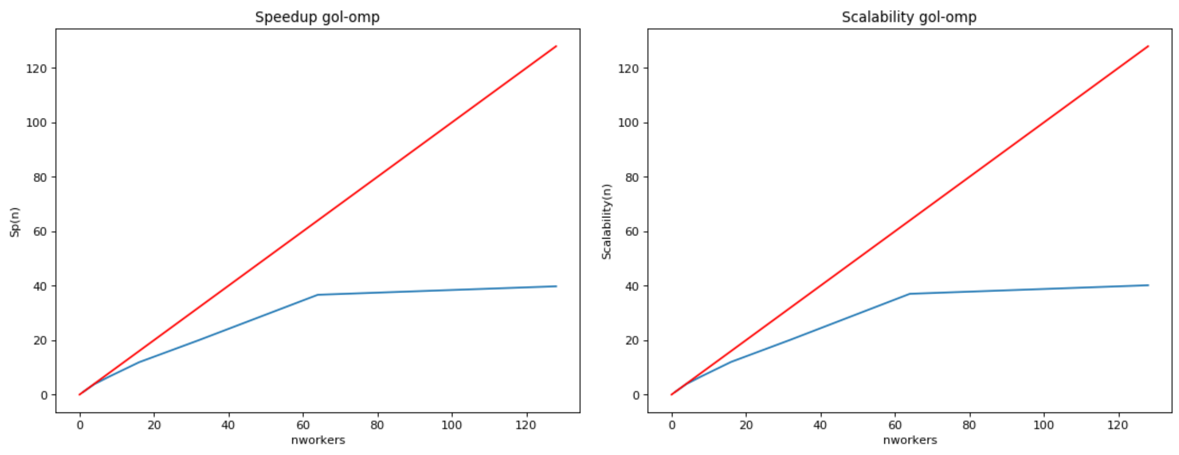

Figure 4: Speedup(left) and Scalability(right) gol-par on 300x200 board.