Game of Life

Francesco Corti - 606922

April 2020

Abstract

The purpose of the assignment was to implement different versions of Game Of Life, a game devised by the British mathematician John Horton Conway in 1970. Three versions were developed: sequential, parallel using C++ threads version and parallel using OpenMP.

1 Code structure

The program has a single class *gameOfLife.h* that expose three public methods *Sequential*, *StandardThreads* and *OpenMP*. The user need to pass from the command line the following parameters: number of iteration, random number generator seed, dimension of the board and the parallelism degree. By default the program compute the sequential version followed by the Fork-Join and the OpenMP. For each version the time spent for computing is printed out on the screen.

In order to provide and efficient implementation the two-dimensional board is saved as one dimensional *bool** array, all the index are adjusted to make the computation correctly. After each iteration the update version of the board is swapped using the *std::swap* method. This operation prevent a total copy of the matrix and let the system scale well even with large matrix.

2 Measures

To understand the performance of the program two metrics related to performance and parallelism, *speedup* and *scalability*, were used as performance indicators. For these metrics the speedup asymptote is given by $f(n) = n$.

2.1 Speedup

The *speedup* is the ratio between the best known sequential execution time and the parallel execution time. This is obtained by the formula:

$$\text{s(n)} = \frac{T_{seq}}{T_{par}(n)} \tag{1}$$

and gives a measure of how good is our parallelization with respect to the best sequential computation.

2.2 Scalability

The *scalability* is the ratio between the parallel execution with parallelism degree equal to 1 and the parallel execution with parallelism degree equal to n. This is obtained by the formula:

$$\text{scalab(n)} = \frac{T_{par}(1)}{T_{par}(n)} \tag{2}$$

and measures how efficient is the parallel implementation in obtaining better performance on bigger parallelism degree.

3 Results

I initially developed a sequential implementation of the program as a baseline. This was done to save the sequential time T_{seq} that was used to compute the speedup curves. After that I developed the parallel version with C++ threads (only) and OpenMP, for each of them I saved $T_{par}(1)$ which is the parallel execution time with parallel degree equal to 1 that was used to compute the scalability curves.

To obtain good results the program was tested in the Xeon PHI with board's dimensions of 500x500, 5000x5000 and 10000x10000. For each of these dimensions the zoomed plot is shown on the right side.

3.1 Speedup curves

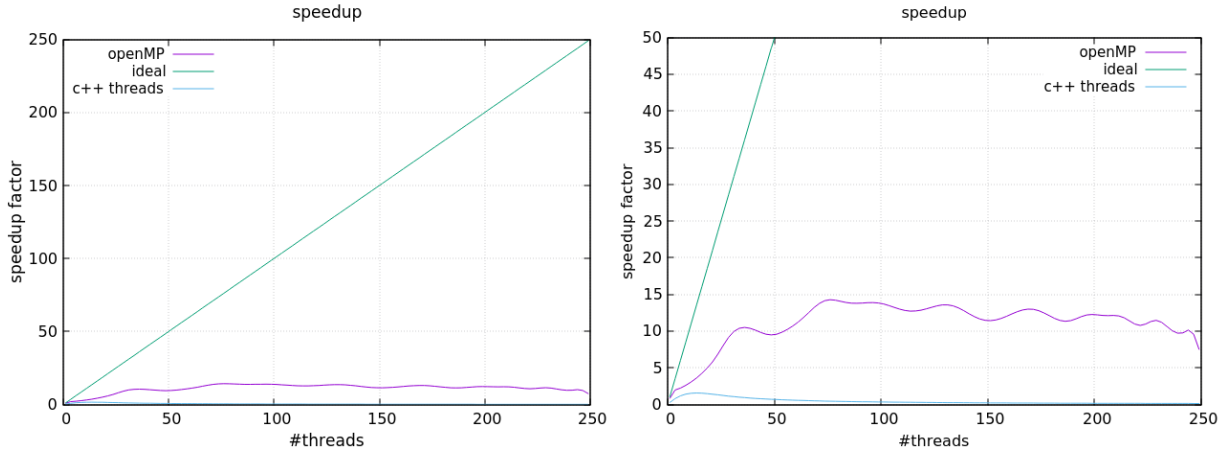

Figure 1: Speedup 500x500 boards.

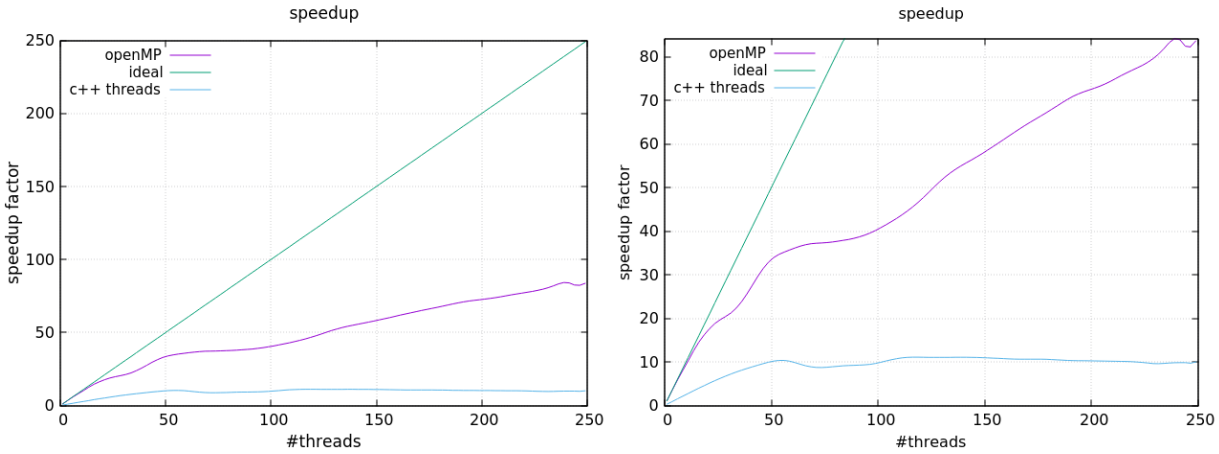

Figure 2: Speedup 5000x5000 boards.

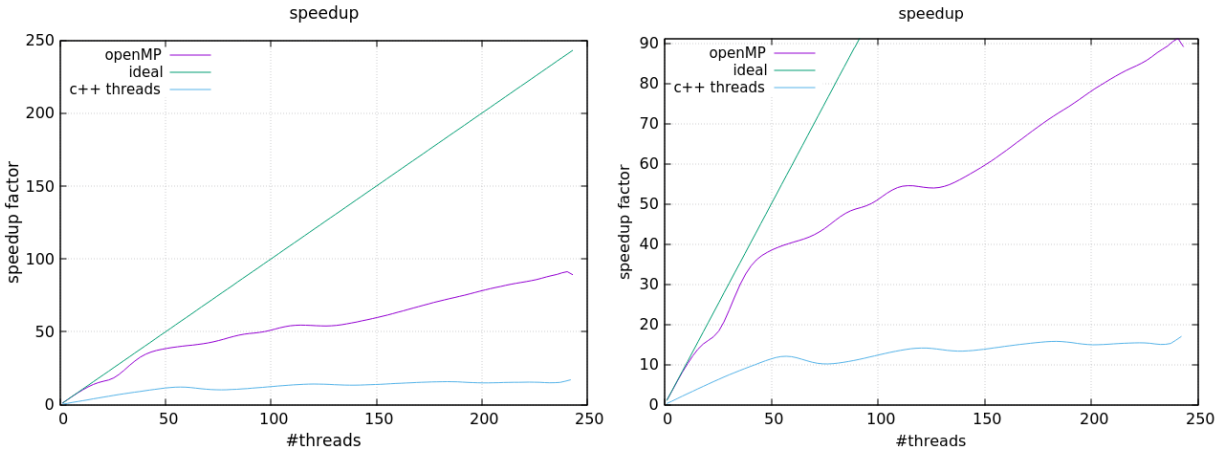

Figure 3: Speedup 10000x10000 boards.

3.2 Scalability curves

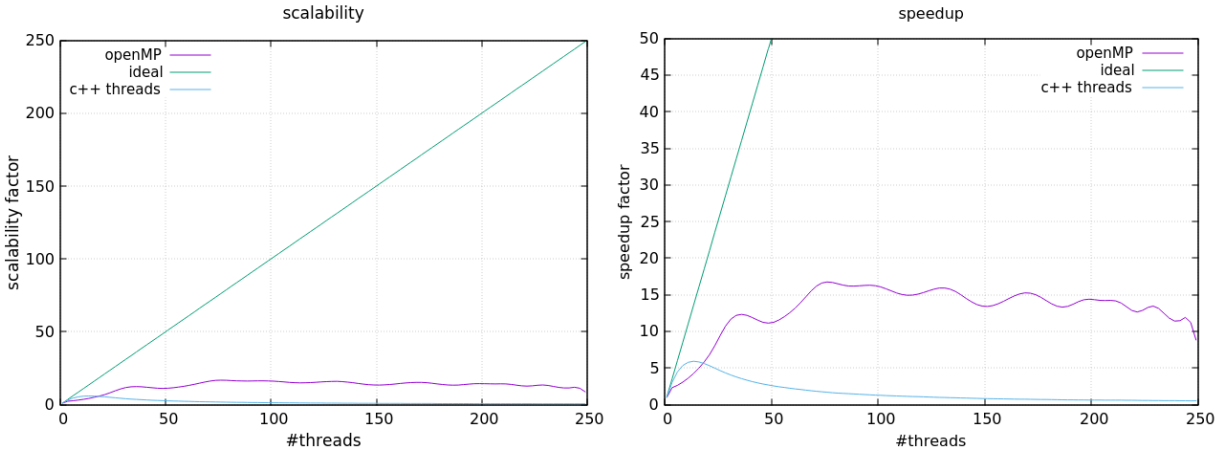

Figure 4: Scalability 500x500 boards.

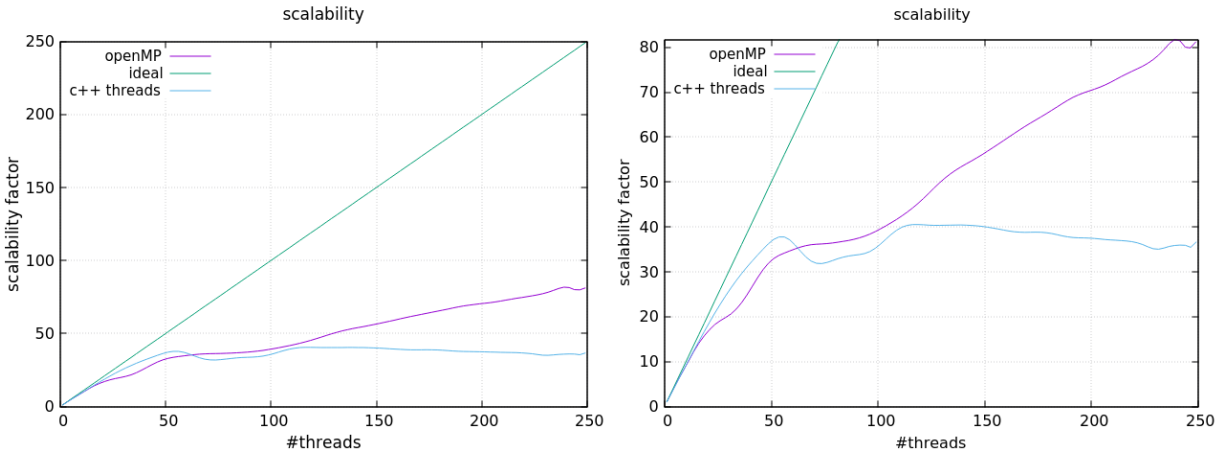

Figure 5: Scalability 5000x5000 boards.

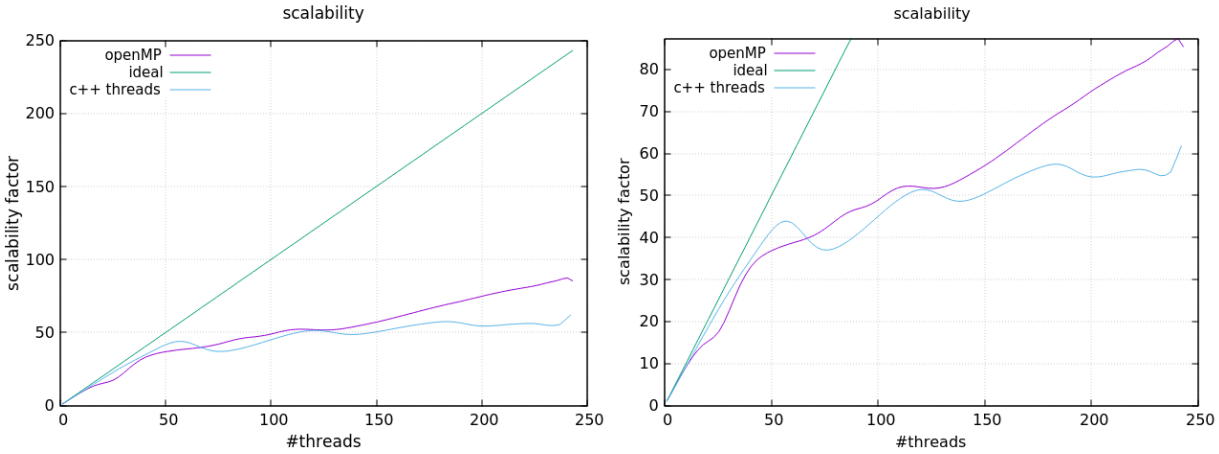

Figure 6: Scalability 10000x10000 boards.

4 Conclusion

As we can seen in figure 1 not all the computations can benefit of an high parallelism degree, with small boards and a high parallelism degree the benefits of adding threads doesn't speed up the computation time obtained to solve the problem. This because the *overhead* taken to setup the parallel computation can exceed the computing time taken to solve the problem and the amount of work given to the worker isn't enough to justify the time spent to spawn them.

Game of Life with little board dimensions isn't a heavy problem to compute, however if we expand the dimensions we can benefits from a high parallelism degree and obtain a good speedup (nearly sublinear with OpenMP) due to parallel computation, see figure 3.