Game of Life

Giovanni Sorice - 606915

April 2020

Abstract

The assignment required to implement different solutions for Conway's Game of Life. I initially developed a *sequential* version, needed as a baseline and performance comparison for further implementations. The parallel versions developed are several.

1 Measures

We need some measures to understand the real advantages of using a parallel implementation. The common measures used in this case are speedup and scalability.

1.1 Speedup

The speedup is used to compare the time needed to execute a specific task sequentially with the time needed to do the same task in parallel. The ratio among the time spent by the sequential and the time spent by the parallel is called Speedup. We hope that it was linearly proportional to the number of parallel degree used and for this reason the time spent doing the task decrease as 1/k where k is the parallelism degree. Unfortunately, there are more things to keep in mind (overhead) and usually the rate of speedup is not the expected. The speedup is computed as follows.

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

1.2 Scalability

The scalability is the ration between the parallel execution time with parallelism degree equal to 1 and the parallel execution time with parallelism degree equal to n.

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

2 Implementation structure

I defined four classes: *GoLSeq*, *GoLThread*, *GoLOMP* and *GoLPool* in which there are respectively the implementation of the sequential code, parallel code with c++ standard thread used as fork-join, parallel code with OpenMP and parallel code with c++ standard thread used as a pool of worker and queue of tasks. The results of pool version are really good when I use small matrices but very bad in big matrices. I think that this behavior is influenced by the bottleneck in the queue of tasks in which the matrix is decomposed cell by cell. For this reason, the graph with big amount of rows and columns are not displayed with the pool implementation. All the code could be found at https://github.com/GiovanniSorice/GameOfLife.

3 Results

The results are shown in graphs, all the execution are made on square matrices. The size tested are 500x500, 1000x1000, 5000x5000 and 1000x1000 with 15 iterations. The test are made on Xeon Phi.

3.1 Speedup graph

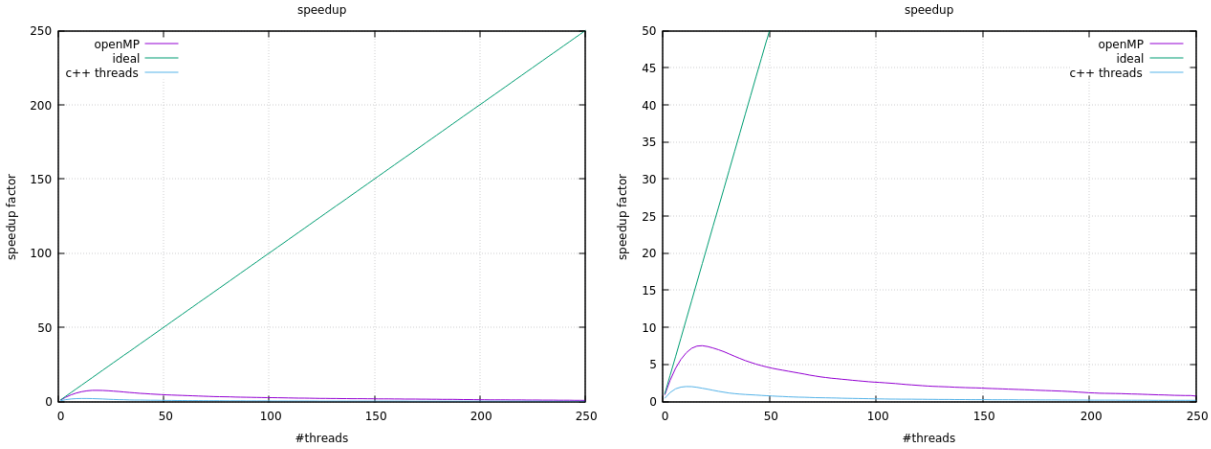

Figure 1: Speedup curves for 500x500 boards.

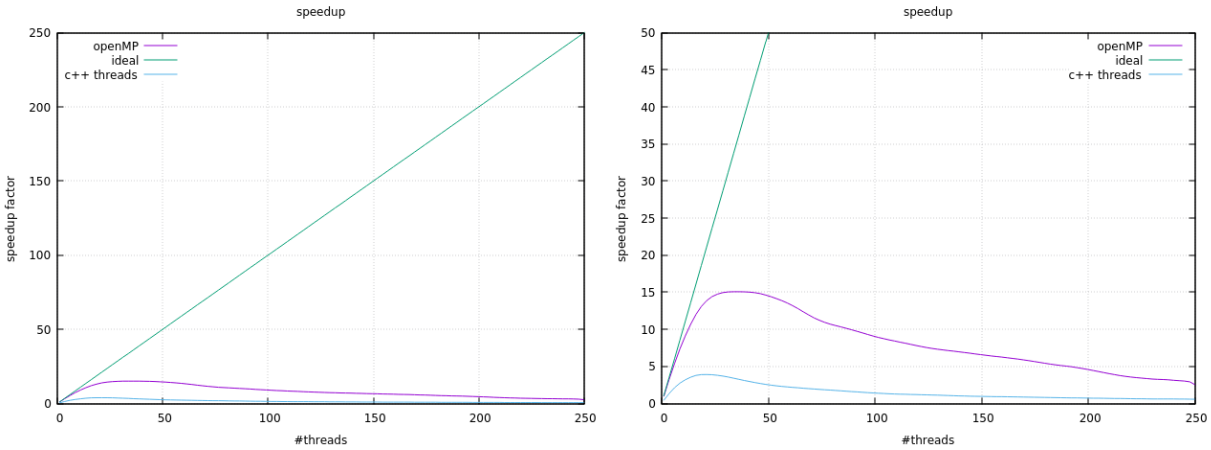

Figure 2: Speedup curves for 1000x1000 boards.

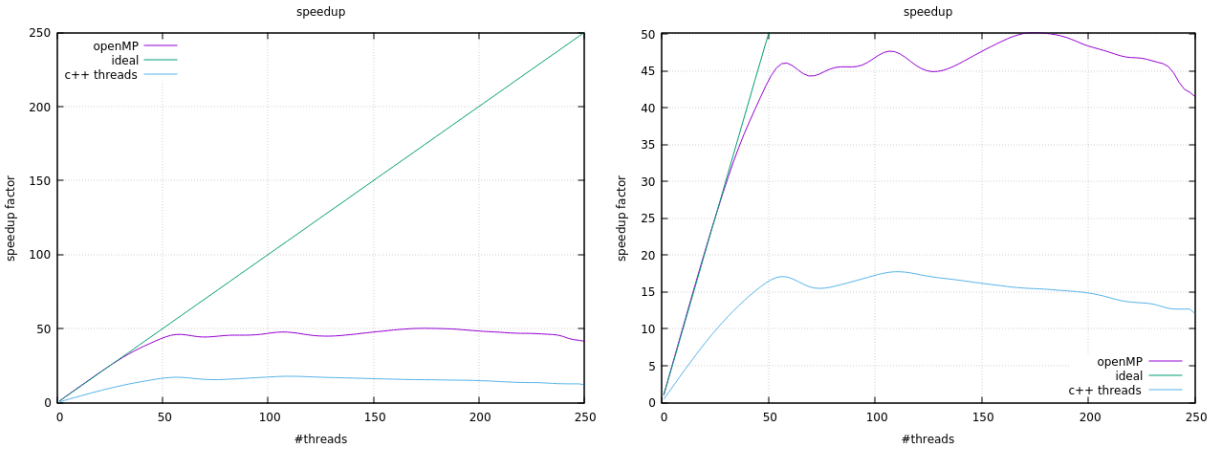

Figure 3: Speedup curves for 5000x5000 boards.

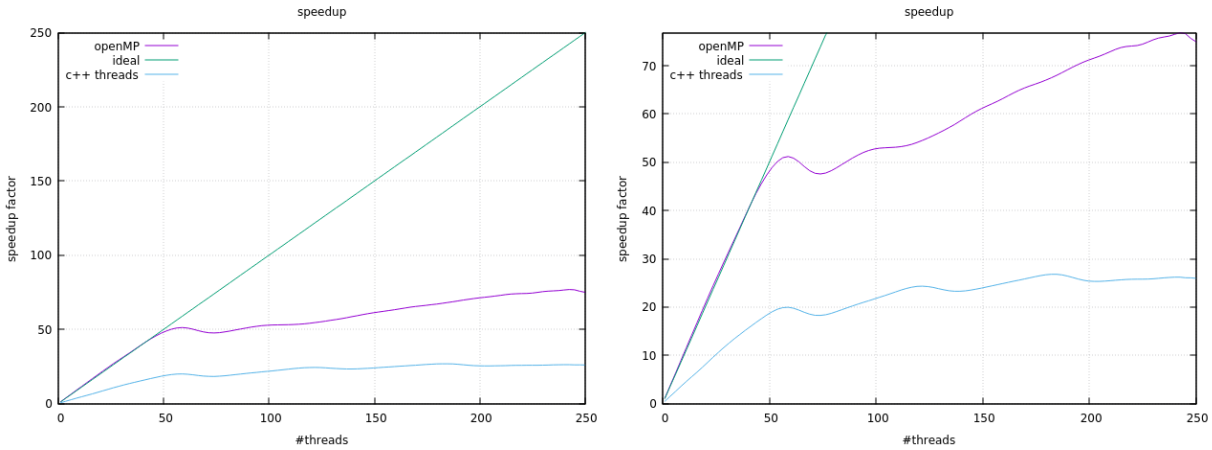

Figure 4: Speedup curves for 10000x10000 boards.

3.2 Scalability graph

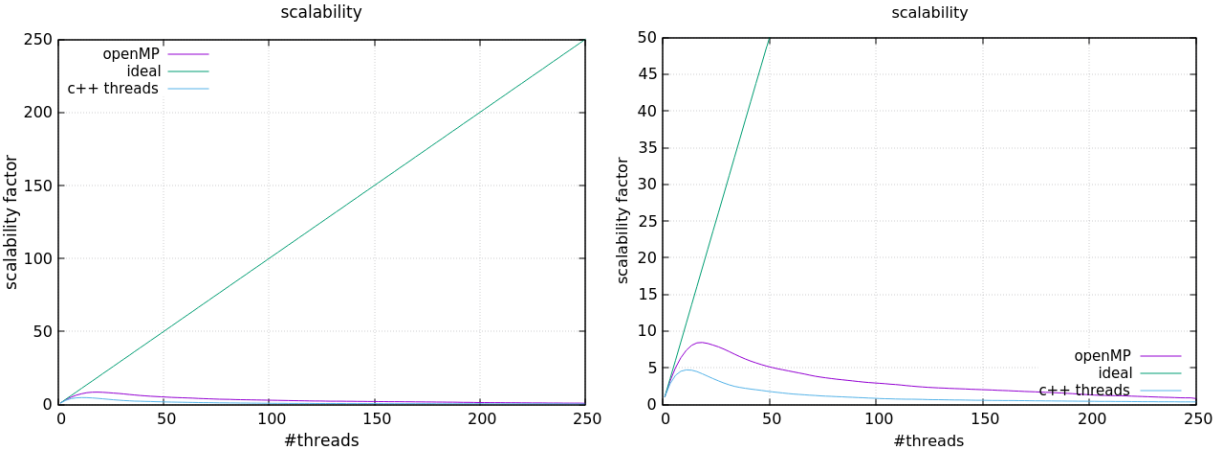

Figure 5: Scalability curves for 500x500 boards.

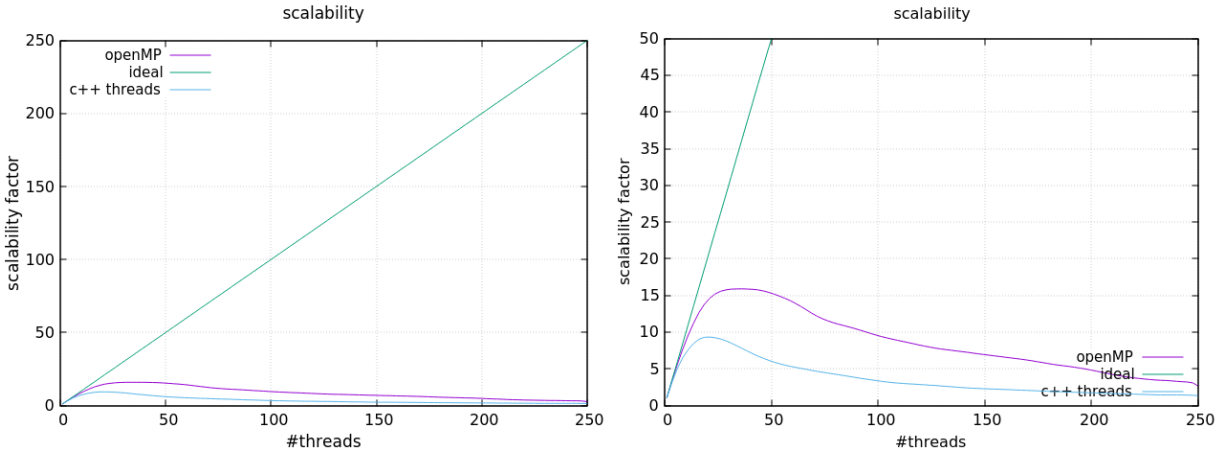

Figure 6: Scalability curves for 1000x1000 boards.

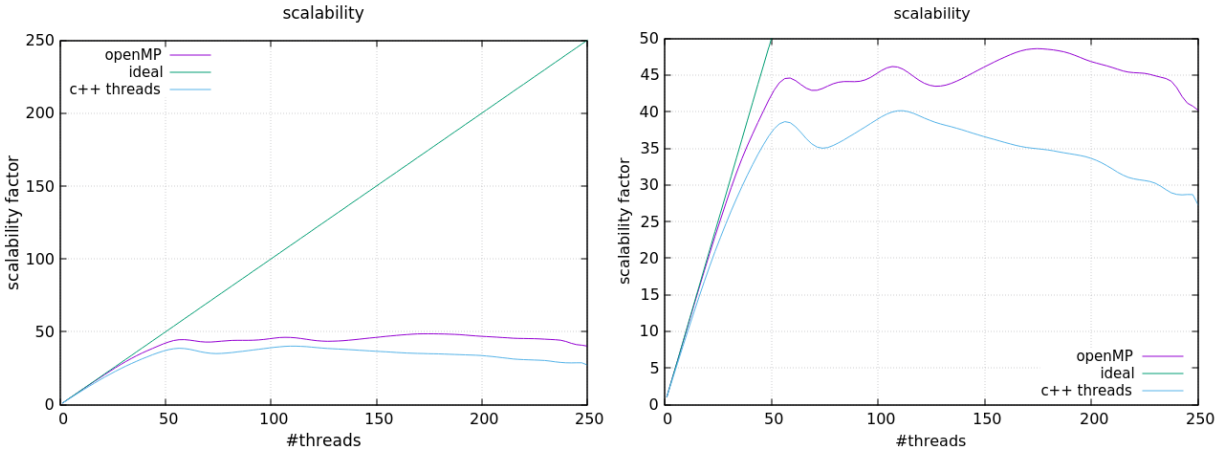

Figure 7: Scalability curves for 5000x5000 boards.

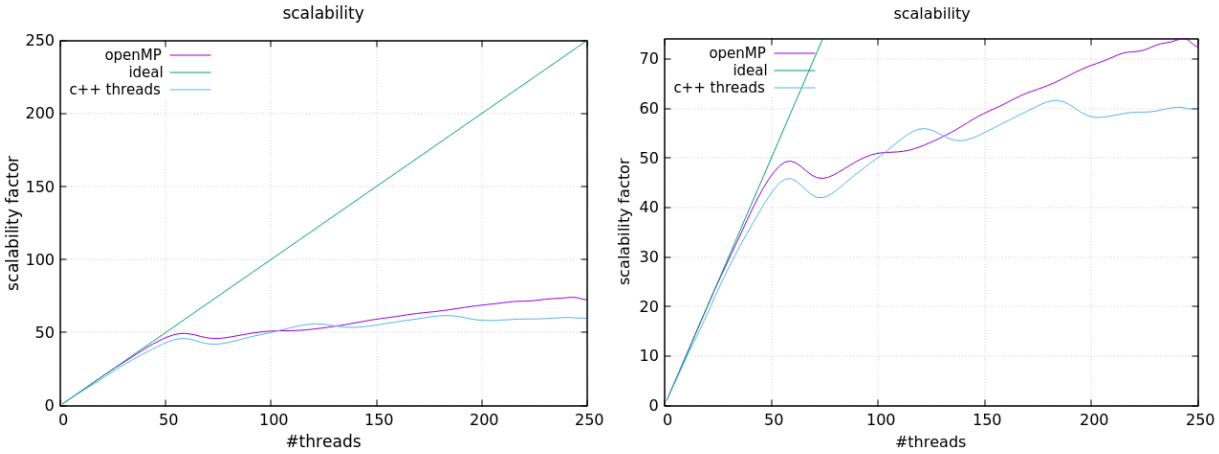

Figure 8: Scalability curves for 10000x10000 boards.

4 Conclusions

We can see in the graphs that as expected the speedup and the scalability do not increase linearly, but at a certain point reach an inflection point and start decrease or stabilize itself. This can be attribute to the increasing of the overhead of the splitting. In conclusion, it is important to find the right number of parallelism degree and not underestimate the overhead.