

Parallelizing C++ Game of Life with OpenMP

Parallel Computing Laboratory

Giacomo Orsucci

February, 2026



UNIVERSITÀ
DEGLI STUDI
FIRENZE
Da un secolo, oltre.



Project objective

- Implement a first sequential version of cellular automata simulation Game of Life.
- Experiment various parallel versions using OpenMP.
- Evaluate the impact given by parallelization, cache tiling, SIMD, etc etc.

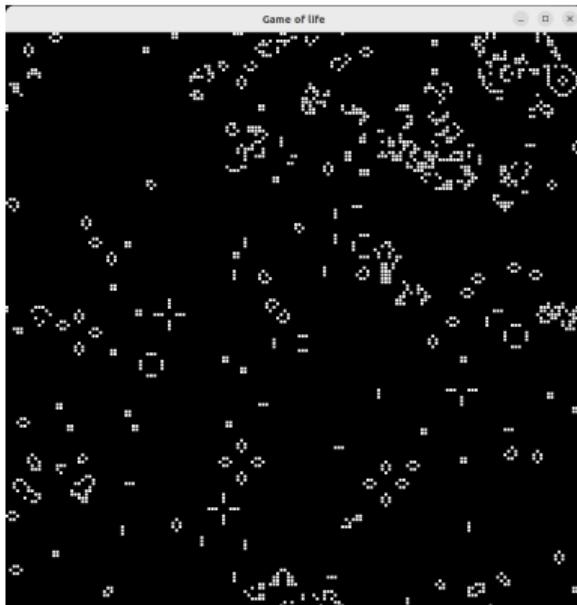


Figure: Graphical Game of Life simulation.

Game of life rules

Four main rules based on counting neighbors of a 3x3 grid centered on cell:

- **Underpopulation:** A live cell with fewer than two live neighbors dies.
- **Stays Alive:** A live cell with two or three live neighbors stays alive.
- **Overpopulation:** A live cell with more than three live neighbors dies.
- **Reproduction:** A dead cell with exactly three live neighbors becomes alive.

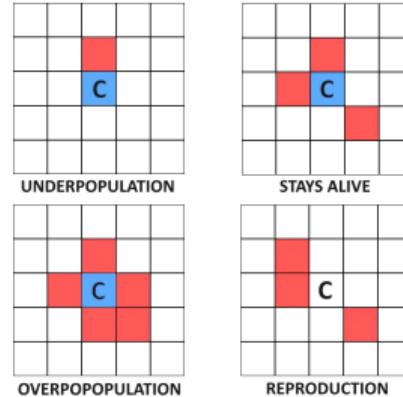


Figure: Four main rules of Game of Life.



Different implementations and experiments

Sequential

```
1
2     for each row:
3         ...
4             for each column:
5
6                 count neighbors around cell (i,j);
7                 apply Game of Life rules to cell (i,j);
8                 ...
9
```



Different implementations and experiments

Parallel

```
1
2 #pragma omp parallel for collapse(2)
3   for each row far from borders:
4     for each column far from borders:
5
6       count neighbors_noBorder around cell (i,j);
7       apply Game of Life rules to cell (i,j);
8       uses branchless logic;
9
10  for each row and column near border:
11
12    count neighbors_Border around cell (i,j);
13    apply Game of Life rules to cell (i,j);
14    uses branchless logic;
```

Different implementations and experiments

Tiled

```
1 #pragma omp parallel for collapse(2)
2 for each ii in TILE_SIZE:
3     for jj in TILE_SIZE:
4
5         count neighbors_Border around cell (i,j) and
6         apply Game of Life rules to cell (i,j) differently if
7         is near or far from borders.
8
9     Uses simd and branchless logic.
10
11
12
```



Different implementations and experiments

SIMD optimization

```
1 #pragma omp parallel for
2     for each row far from borders:
3
4         count neighbors (using precalculated indexes)
5         and apply Game of Life branchless logic really
6         optimized for classic Game of Life SCAN_SIZE=3.
7
8     for each row near top and bottom borders:
9
10        count neighbors_border and apply
11        Game of Life branchless logic.
12
13     for each column near left and right borders:
14
15        count neighbors_border and apply
16        Game of Life branchless logic.
```



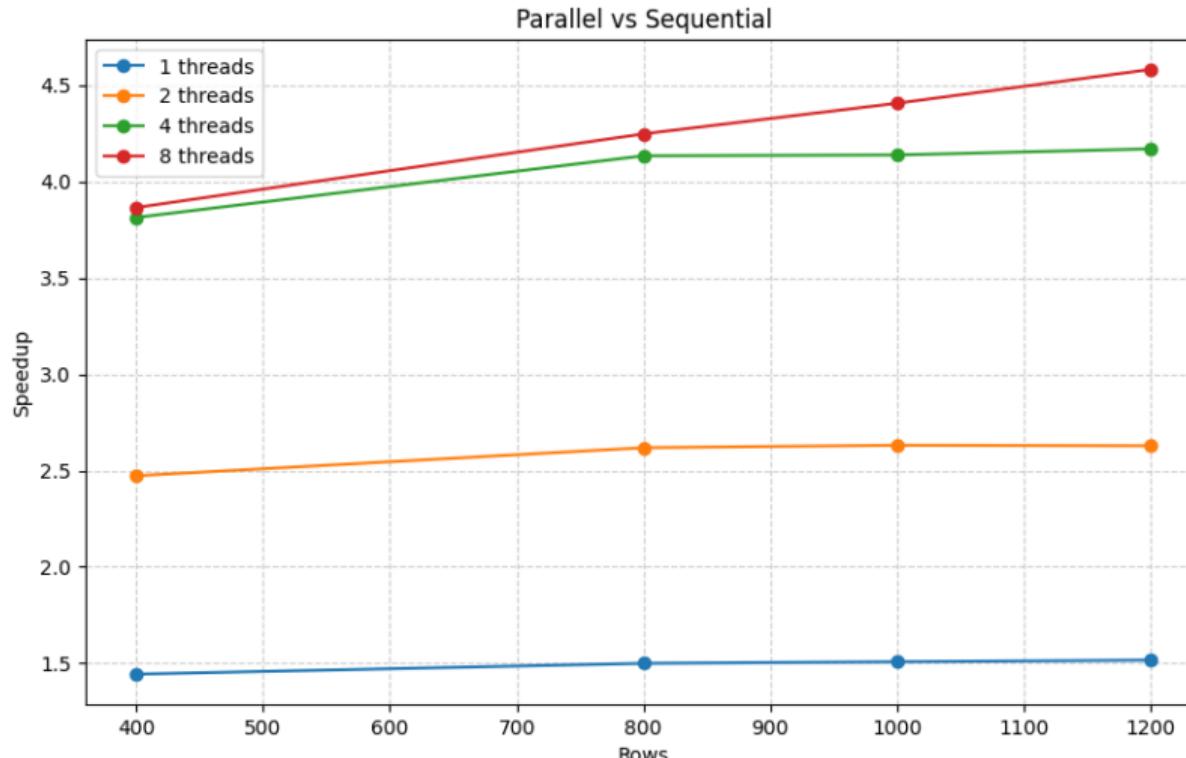
Machine Specifications

- **CPU:** AMD Ryzen 7 3700U, 4 cores, 8 threads, base clock 2.3 GHz, max boost clock 4.0 GHz.
- **Cache:**
 - L1 Cache: 96 KB per core.
 - L2 Cache: 512 KB per core.
 - L3 Cache: 4 MB shared.
- **RAM:** 20 GB DDR4 (1 x 4 GB SODIMM DDR4 + 1 x 16 GB Row of Chips DDR4) at 2400 MHz.
- **OS:** Ubuntu 22.04 LTS.
- **Compiler:** GCC 11.4.0.
- **OpenMP:** version 5.2.



Experimental results

Parallel vs Sequential

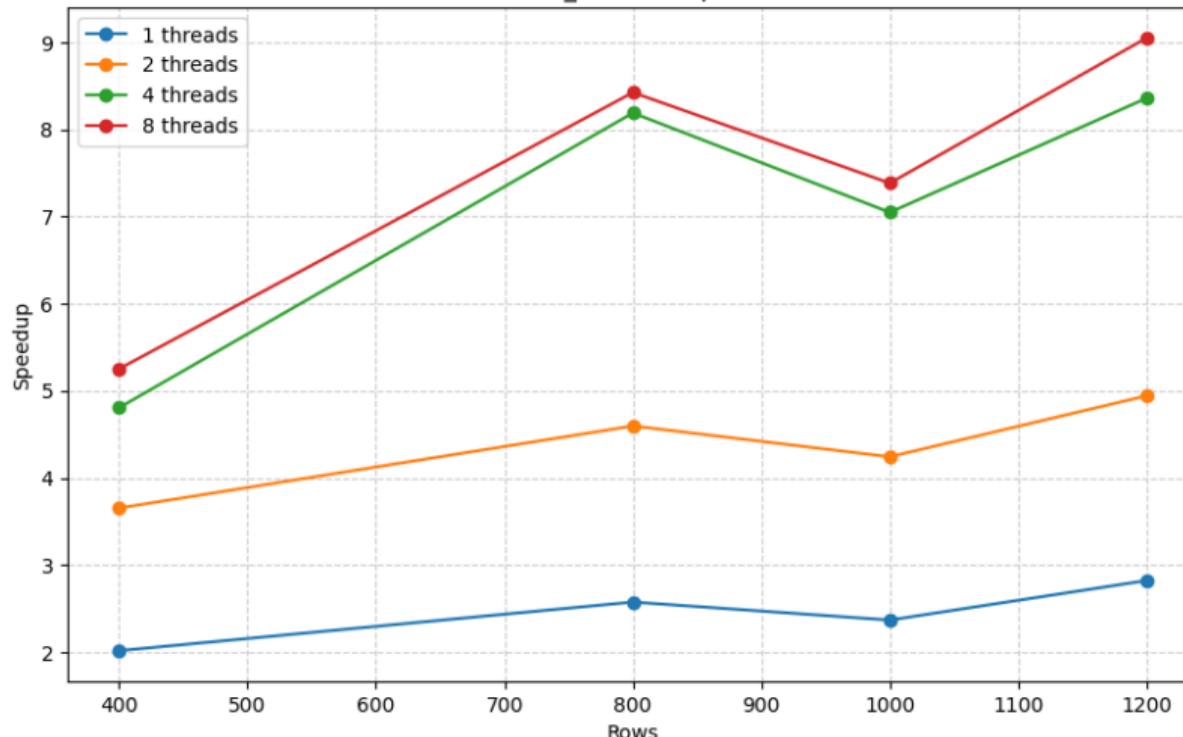




Experimental results

Tiled vs Sequential

Parallel_Tile vs Sequential

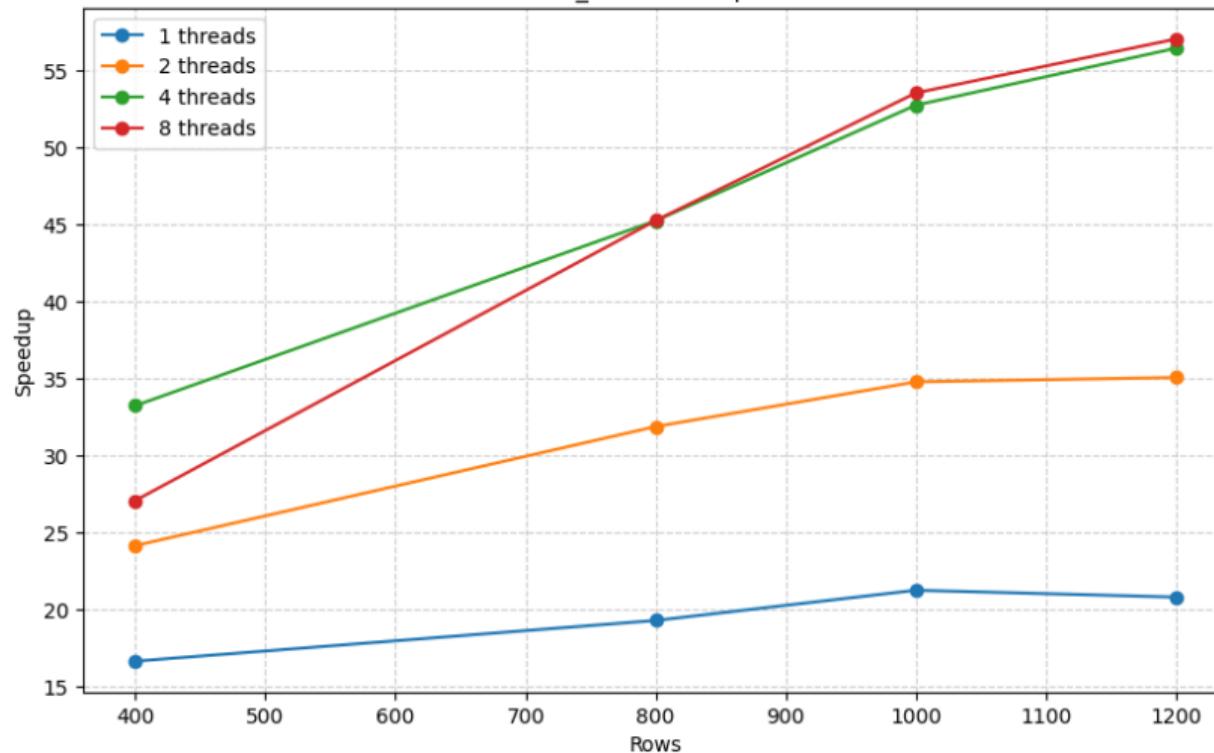




Experimental results

SIMD optimized vs Sequential

Parallel SIMD vs Sequential





Conclusion

- OpenMP parallelization is effective, but its full potential is locked without proper code optimization.
- The experimented cache tiling is a good optimization.
- Hyper-Threading proved to be not so beneficial, significantly increasing contention in the Memory-Bound scenario.
- The maximum SIMD optimization has achieved impressive results.