3-rd laboratory assignment

1. Description of the problem

Create a simulation on a grid with the following rules:

1. Any live cell with fewer than two live neighbors dies, as if by under population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Cells are squares. We can parallelize each cell's neighbor count calculation on different threads.

2. How you have parallelized the algorithm

1. How you divide the work into separate parts?

   Each cell's neighbors can be calculated in parallel. Each cell can execute if clauses (if they should be alive) in parallel.

2. How you allocate the sub works (parts) between threads?

   I created a range of all possible coordinates in the game and for each of them gave a different thread if possible. Each coordinate calculated its neighbors and determined if it should be alive.

3. How you synchronized the threads?

   By using .Net's Parallel.ForEach library. Deeper down it utilizes the TaskScheduler class that queues tasks and assigns threads from the threadpool. It also assigns a synchronization context.

   For GUI synchronization I notify the GUI thread that updates happened and it fetches the new values itself from shared memory.

3. Describe an experiment

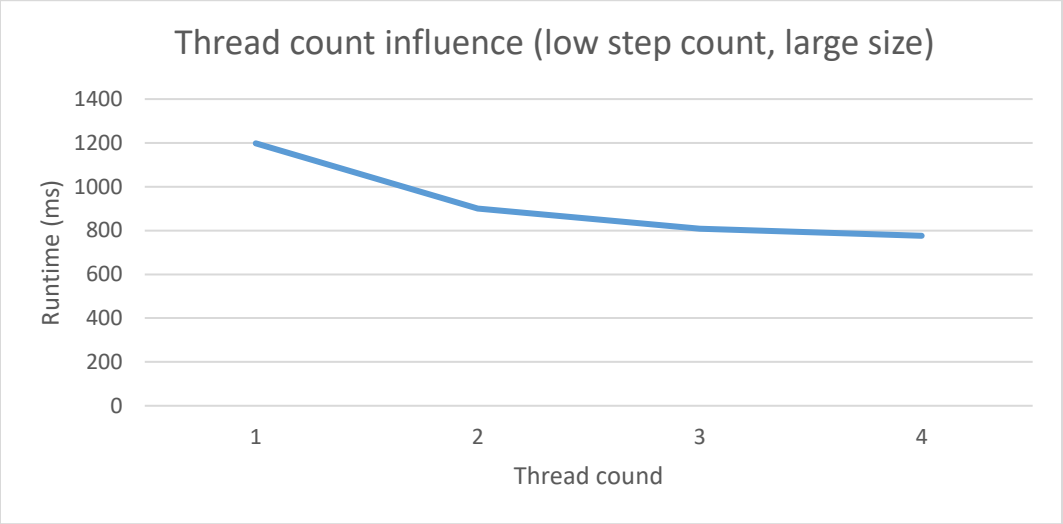
1. System where it was performed

   My local machine.
   Processor: Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz (4 CPUs), ~3.0GHz

2. Results achieved

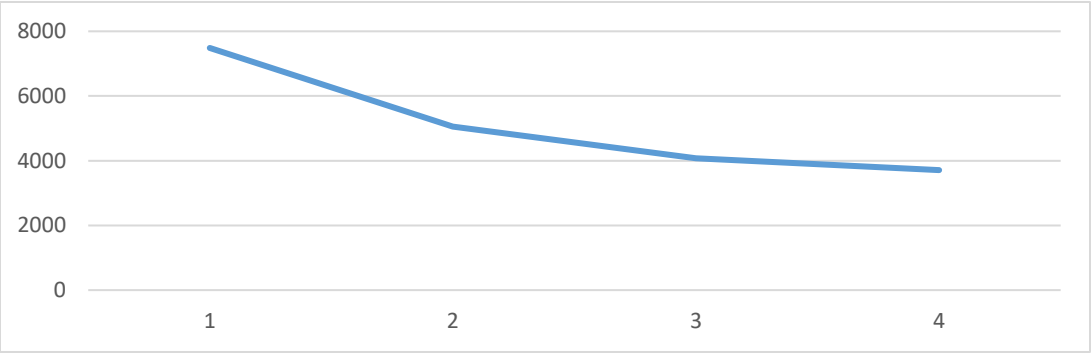
Thread count influence (low step count, large size)

Thread count	Runtime (ms)	Step count	Size
1	1198	20	800
2	901	20	800
3	808	20	800
4	776	20	800



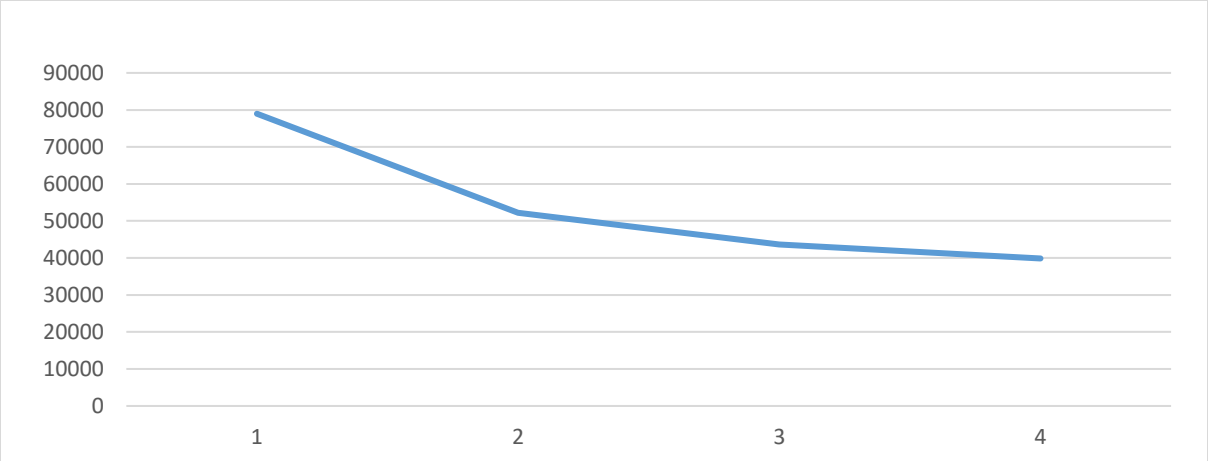
Thread count influence (medium step count, large size)

Thread count	Runtime (ms)	Step count	Size
1	7485	100	800
2	5050	100	800
3	4080	100	800
4	3709	100	800



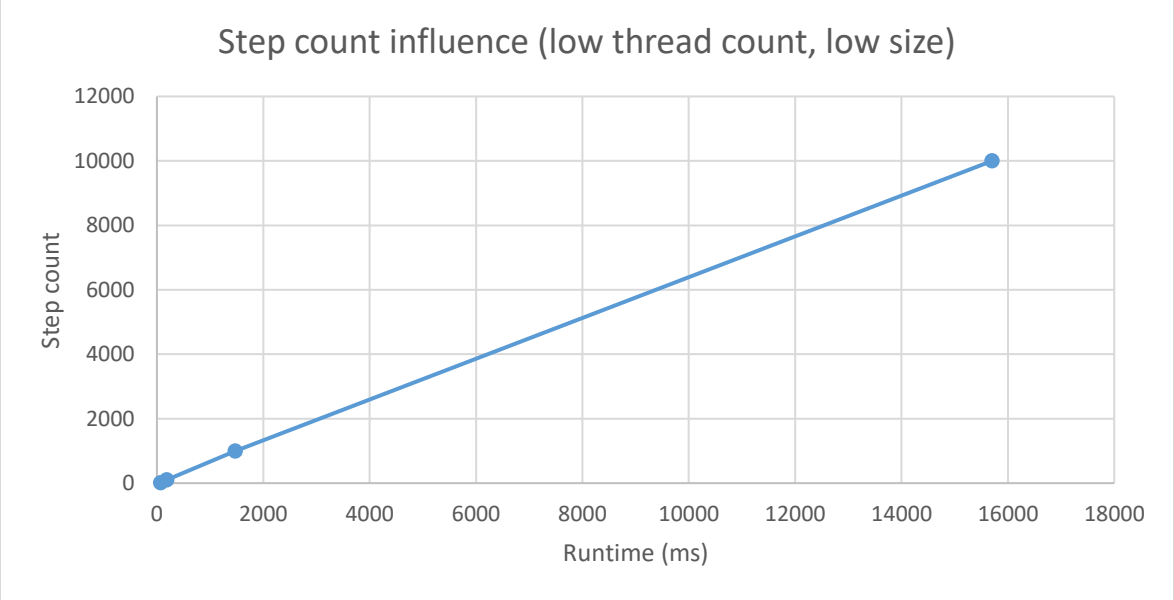
Thread count influence (large step count, large size)

Thread count	Runtime (ms)	Step count	Size
1	78971	1000	800
2	52175	1000	800
3	43642	1000	800
4	39855	1000	800



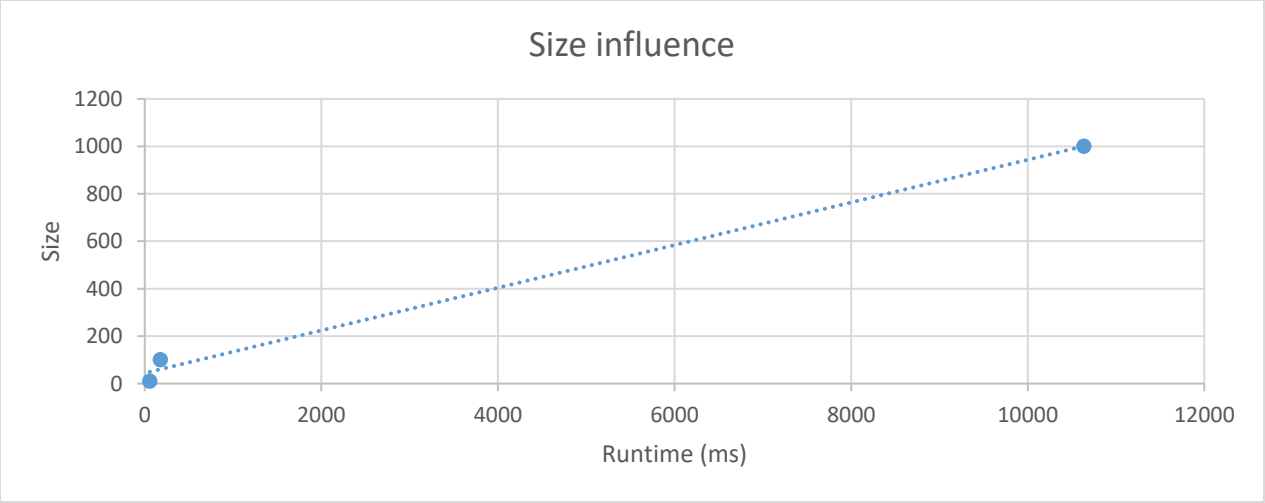
Step count influence (low thread count, low size)

Thread count	Runtime (ms)	Step count	Size
1	69	10	100
1	186	100	100
1	1471	1000	100
1	15703	10000	100



Size influence

Thread count	Runtime (ms)	Step count	Size
1	56	100	10
1	176	100	100
1	10635	100	1000



4. What are the conclusions

1. What maximum speedup has been achieved?

   Increasing the core count three times cuts the execution time to ~50%.

2. Is the speedup linear /perfect/, very good, not bad, little, nothing?

   The speed up is decent. Adding a new core reduces the runtime by around 25 % except for the last core.

3. If the speed up is not "perfect" what (you think) is a reason for this

   Synchronization and added overhead of using a safe library.

4. Does the experimental data shows if the algorithm is scalable?

   The algorithm is scalable up until we have the same number of cores as there are coordinates.