Cellular Automaton

This is a cellular automaton program using OpenMP. The program consists of a two-dimensional matrix, divided into **cells**. Each cell is either **dead** or **alive** at a given **generation**. The game consists of a set of rules that describe how the cells evolve from generation to generation. These rules calculate the state of a cell in the next generation as a function of the states of its neighboring cells in the current generation. A cell's neighbors are those 8 cells vertically, horizontally, or diagonally adjacent to that cell. Of course, corner and border cells will have less number of neighbors.

The rules are summarized as follows:

1. An alive cell with fewer than two alive neighbors dies in the next generation.

2. An alive cell with more than three alive neighbors also dies in the next generation.

3. An alive cell with exactly two or three alive neighbors stays alive in the next generation.

4. A dead cell with exactly three alive neighbors becomes alive in the next generation.

Even though the 2D world is supposed to be infinite, we will restrict it for this project to be NxN matrix where N is entered by the user.

Each entry contains a cell that can be ALIVE or DEAD.
The main loop of the program will then be:

```
For each generation
    For each cell
        look at the neighbors and decide the state of the cell (DEAD or ALIVE)
        update the cell's state
```

Important: The neighbors must NOT see the new state of the cell until next generation. They must do their decisions based on the current state of the cell. For example, if a cell is alive in generation x and, based on the rules, must be dead in generation x+1, that new status must not be used by the neighboring cells in generation x. That is, all the other cells must not see the new states of the neighbors till the following generation.

Here is an example:

Generation 0:

	*	*		
		*		
		*		

Generation 1:

	*	*		
		*	*	

The program that you will work on needs four arguments entered through the command line in the following order:

1. The number of generations
2. The dimension N of the matrix. The matrix is assumed to be always square.
3. The number of threads
4. A filename of the file that contains the initial matrix. The format of the file is a NxN matrix of integers. Cells at each row are separated by a space. An alive cell is presented by a 1, while a dead one is presented by a 0.

For example, the first generation above is presented in the file as (in the project it will be 10x10):

```
0 0 0 0 0
0 1 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 0 0 0
```

Notes:

- We compile your code with: **gcc -Wall -std=c99 -o automata -fopenmp automata.c**
- Run: **./automata X Y Z filename** which means start with
  the position in filename that contains YxY matrix and run it for X generations using Z threads.
- The output of your program is a text file called filename.out that contains the YxY matrix of the last generation.
- Assume that we will NOT test your code with wrong inputs (e.g. 0 or negative x, …).

To help you:

- You will find on the course website two executable files: **automata** and **genmap**
- They run only on Linux machines. That is, they will not run on your MAC or Windows machines.
- **standard**: is the sequential implementation of the problem. It has the same command line arguments as your OpenMP program but without the number of threads. You do not need to compare the speed of your program with this version. This version is provided only to check the correctness of your output code. You executed as:
  ./automata X Y filename
  where X is the number of generations, Y is the dimension, and filename is the input file that contains the initial YxY matrix of cells. The program will generate the final generate in a file called: filename.out
- **genmap**: generates an input file that contains a matrix of NxN cells. You execute it with:
  ./genmap N filename
  where N is the dimension and the filename is the file that will contain the NxN matrix generated with random cells dead and alive. This is the file that you can use as input file to automata
- Important, before executing automata and genmap, execute the following two commands:
  chmod 777 standard
  chmod 777 genmap

Experiments

To see how efficient your implementation is, you need to compare the performance with different number threads.
Therefore, do the following:

- Prepare three tables: one for the time in seconds, one for the speedup (relative to single thread, which is Z = 1), and the third for efficiency (speedup/# threads).
- Each table is really one row and a column for each number of threads.
- The dimension of the matrix is 1000x1000
- The number of generations is 1000
- The experiments will try with the following number of threads: 1, 2, 5, 10, and 20.
- Time only the parallel part using omp_get_wtime(). Do not time file opening, reading, and the output file writing. Time only the parallel part.