Assignment Details

Project Details

The ACW aims to design, implement and evaluate both parallel and non-parallel cellular automation simulation for the spread of fire in a forest based on a given partial sequential implementation of the simulation.

The problem

The problem is to simulate the spread of fire from an initial landscape of empty ground, non-burning trees, and trees that are on fire. Moreover, the area can suffer from lightning strikes, which may or may not start additional fires.

For the cellular automaton simulation of the spread of fire, a cell of an n × n grid can contain a value of 0, 1, or 2 indicating an empty cell, a cell with a non-burning tree, or a cell with a burning tree, respectively.

The discrete stochastic system is initialized with two probabilities (**probTree** and **probBurning**). **probTree** is the probability that a tree initially occupies a site (the initial tree density measured as a percentage), and **probBurning** is the probability that the tree is initially burning (the fraction of the trees that are burning, when the simulation begins).

The fire spread is determined at every simulation iteration, which determines whether a cell is empty, has a tree or is burning in the next time step. The value in a cell depends on the neighboring cell values using the von Neumann neighborhood. The update rules below determine the cell's value:

- If a site is empty (cell value EMPTY), it will remain empty at the next time step.
- If a tree grows at a site (cell value TREE)
    - The site doesn't catch fire if it is immune (random < probImmune).
    - Otherwise, the site catches fire.
        - If a neighbour has fire or lightning strikes the site (random < probLightning).
        - ProbImmune is the probability of immunity of a site from catching fire. pro lightning is the probability of lightning hitting a site
    - A burning tree (cell value BURNING) always burns down, leaving an empty site (value EMPTY) for the next time step.

For this simulation, apply the function **spread** (a function that determines spread of fire in the next iteration) to every grid point, using **periodic boundary** (ghost cells applied to the initial grid) conditions. To apply a spread, extend the boundaries by one cell. After, apply the function **spread** to each internal cell and then remove the boundary cells.

Implementation requirements

Python will be the language for the implementation on a multicore computer. You will need the partial sequential implementation of the simulation, which you implemented in Lab sessions.

The following functionalities should be converted from sequential implementation to parallel implementation:

- Initialization of forest grid
- Extending the boundaries of the grid using periodic boundary condition, and applying the spread.

Additionally, you are also required to change the **von Neumann neighbourhood** in the initial sequential implementation to the **Moore neighbourhood** and implement a **visualization** for the spread of fire for both the sequential and parallel implementations.

Running time of the whole program and each of the above functionalities should be evaluated and compared between the sequential and parallel implementations. For each algorithm, evaluate the running time on different grid sizes (100x100, 400x400, 800x800, 1000x1000, 1200x1200 and 2000x2000), using the different neighbourhood algorithms, and tabulate your results (see example in Table 1). The running time can be measured using Python "time" or "timeit" function.

Implementation. Grid size	Sequential	Parallel
100x100		
400x400		
800x800		

Table 1 *Running times of initialization of forest in sequential implementation using Moore's neighborhood algorithm*

Use the following parameter values in comparing the running times of the various algorithms:

- probTree = 0.8
- probBurning = 0.01
- probImmune = 0.3
- probLightning = 0.001