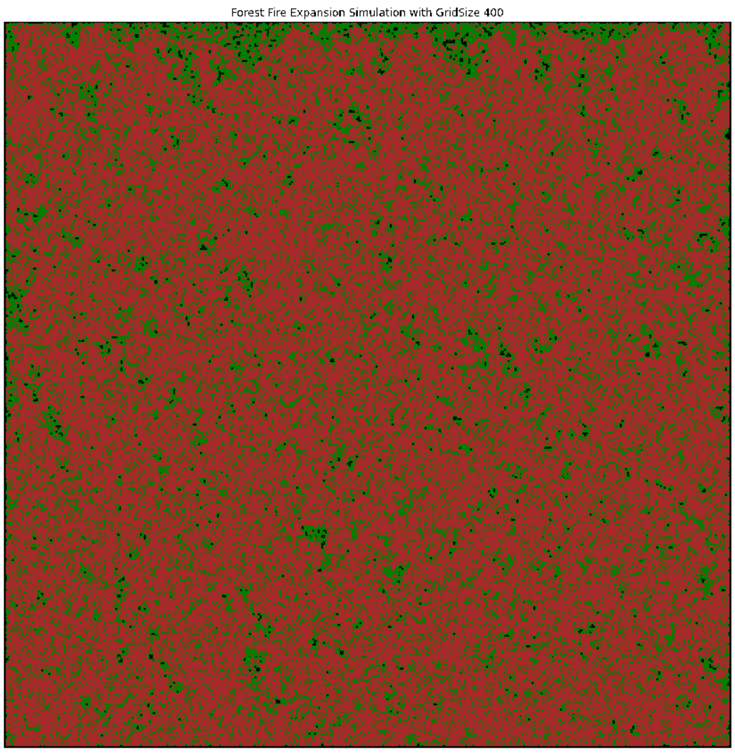
Forest Fire Expansion Simulation Report

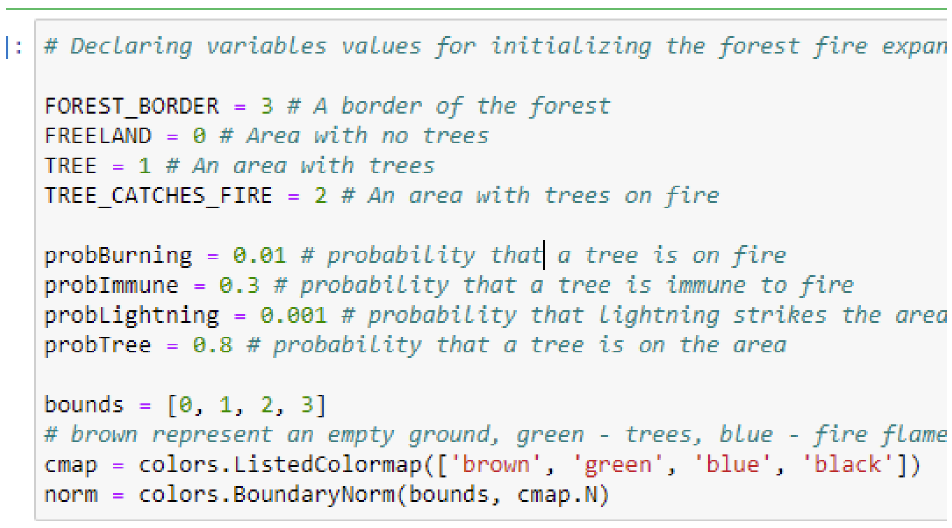
Overview

This study aims to detail the techniques used to apply and assess the results of the cellular automation models designed for expanding fires in a forest. The forest comprises of free lands, tree that are on fire and trees that are not on fire. To do so, I made a few assumptions and collected the data required to create and fill the initial forest grid. After which I applied the fire expansion model across the forest grid for every given position on the grid. It also compares the execution times of sequential and parallel implementations of the models to demonstrate the performance gains of executing them sequentially or concurrently among computer processing units.

Simulations



The forest grids was initialized using the Numpy python library and displayed using the Matplotlib library. Being a cellular automation simulation (Shiflet & Shiflet, 2006), each cell in the forest grids interact with each other, and could cause a change in cell state, — that is, a fire. However, before initializing the forest site, I declared the variables for the different cell states and probabilities as seen in the image below. The cell states are the forest border, free land, tree and trees on fire depicted using the numeric values 3, 0, 1, and 2 respectively in the forest grid.

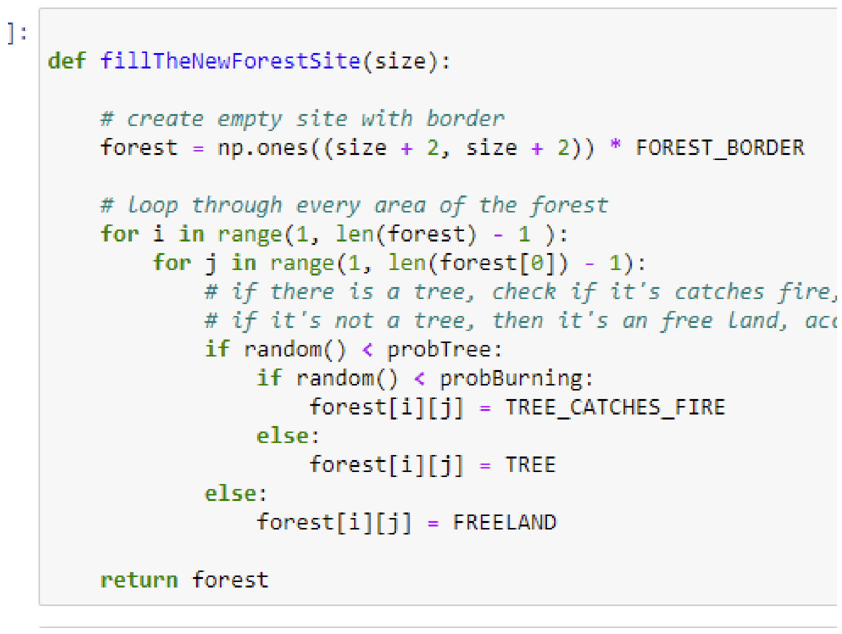


```
|: # Declaring variables values for initializing the forest fire expan

FOREST_BORDER = 3 # A border of the forest
FREELAND = 0 # Area with no trees
TREE = 1 # An area with trees
TREE_CATCHES_FIRE = 2 # An area with trees on fire

probBurning = 0.01 # probability that a tree is on fire
probImmune = 0.3 # probability that a tree is immune to fire
probLightning = 0.001 # probability that lightning strikes the area
probTree = 0.8 # probability that a tree is on the area

bounds = [0, 1, 2, 3]
# brown represent an empty ground, green - trees, blue - fire flame
cmap = colors.ListedColormap(['brown', 'green', 'blue', 'black'])
norm = colors.BoundaryNorm(bounds, cmap.N)
```

Upon declaring the variables for those cell states and probabilities, the forest grid was initialized under the following conditions

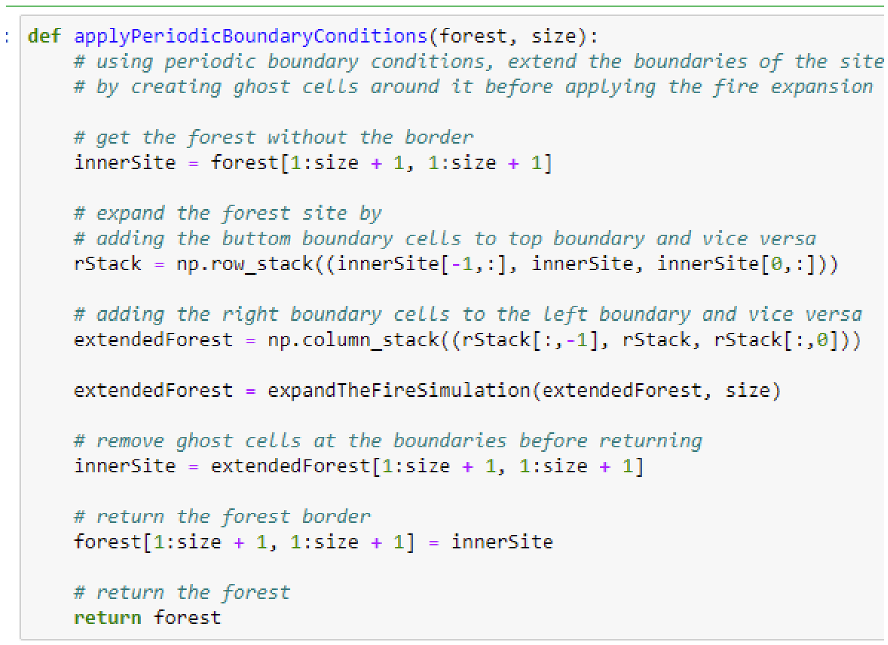
- That the position of the cell states is a product of chance, hence, the Monte Carlo randomized approach (Gentle, 2003) was used to mathematically determine the probability that a tree or burning tree on fire was existing initially at a location.
- That a tree existing at a location is a condition for the existence of burning trees at that location.
- That for forest grid locations where there are no trees, a value representing a free land is assigned to it, showing it's an empty area.

See the image below for the **fillTheNewForestSite** function responsible for initializing the forest grid using the conditions listed above.



```
]:
def fillTheNewForestSite(size):

    # create empty site with border
    forest = np.ones((size + 2, size + 2)) * FOREST_BORDER

    # loop through every area of the forest
    for i in range(1, len(forest) - 1 ):
        for j in range(1, len(forest[0]) - 1):
            # if there is a tree, check if it's catches fire,
            # if it's not a tree, then it's an free land, acc
            if random() < probTree:
                if random() < probBurning:
                    forest[i][j] = TREE_CATCHES_FIRE
                else:
                    forest[i][j] = TREE
            else:
                forest[i][j] = FREELAND

    return forest
```

Having initialized the forest grid, I created another function to help extend the boundaries of the forest by creating ghost cells around the forest boundaries. These essence of creating these ghost cells is to provide neighbours to cells at the borders of the forest grid. This step was important because one of the requirements for the expansion of the fire, is the existence of a tree at any of its Moore neighbours. That is, cells at the north, south, east, west, south east, south west, northwest and north east. The function responsible creating the ghost cells at the boundaries is called the **applyPeriodicBoundaryConditions()** and is displayed below.

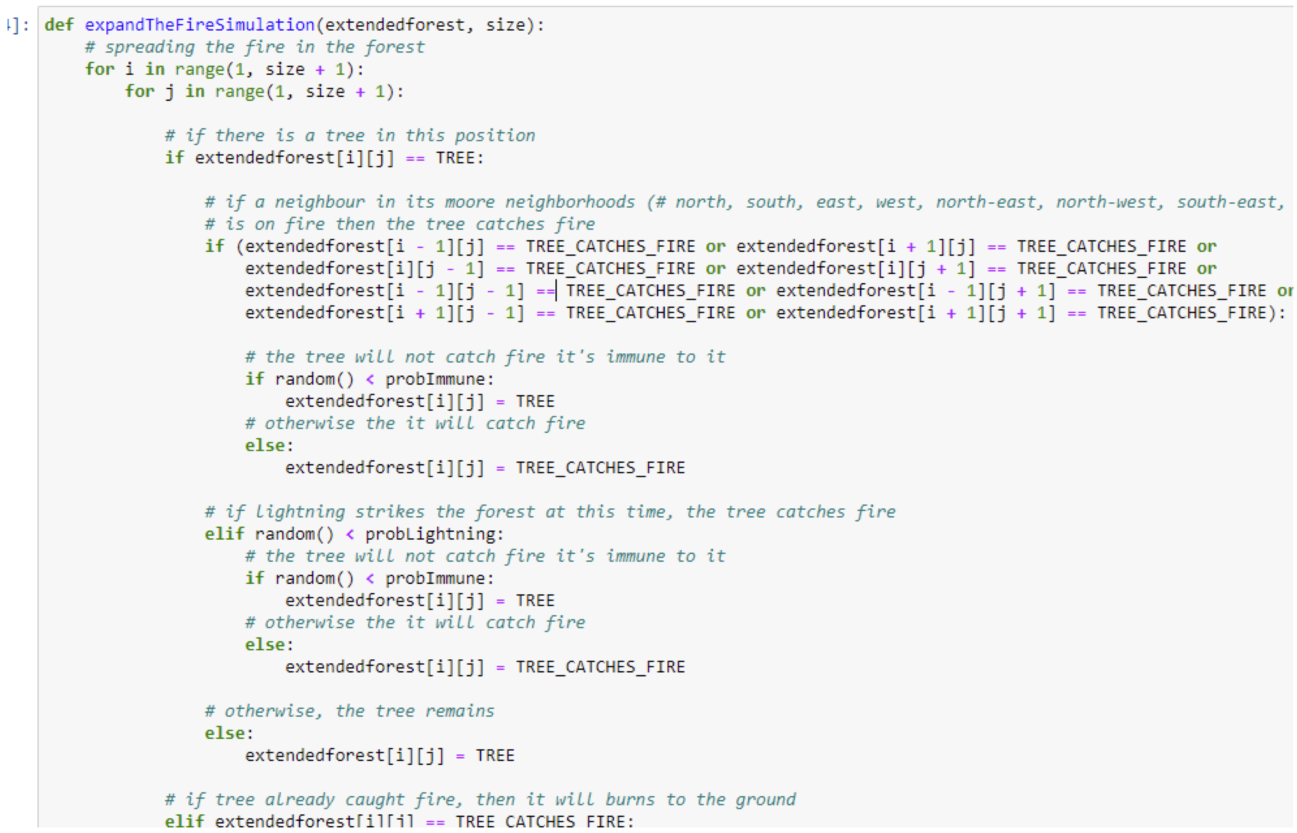


```
def applyPeriodicBoundaryConditions(forest, size):
    # using periodic boundary conditions, extend the boundaries of the site
    # by creating ghost cells around it before applying the fire expansion

    # get the forest without the border
    innerSite = forest[1:size + 1, 1:size + 1]

    # expand the forest site by
    # adding the buttom boundary cells to top boundary and vice versa
    rStack = np.row_stack((innerSite[-1,:], innerSite, innerSite[0,:]))

    # adding the right boundary cells to the left boundary and vice versa
    extendedForest = np.column_stack((rStack[:,-1], rStack, rStack[:,0]))

    extendedForest = expandTheFireSimulation(extendedForest, size)

    # remove ghost cells at the boundaries before returning
    innerSite = extendedForest[1:size + 1, 1:size + 1]

    # return the forest border
    forest[1:size + 1, 1:size + 1] = innerSite

    # return the forest
    return forest
```

Upon successful completion of extending the boundaries, the expansion rules are applied as stated below.

- A tree will catch fire if it's not immune and one of the cells in its Moore neighbourhoods is on fire. Otherwise, it's won't catch fire.
- A tree will catch fire if it is not immune and lightning strikes the site. Otherwise it doesn't burn
- A burning tree burns to the ground and the cell in the grid is assigned the value of a free land.
- Free lands remain the same.

The implementation of the fire expansion is displayed in the image below.

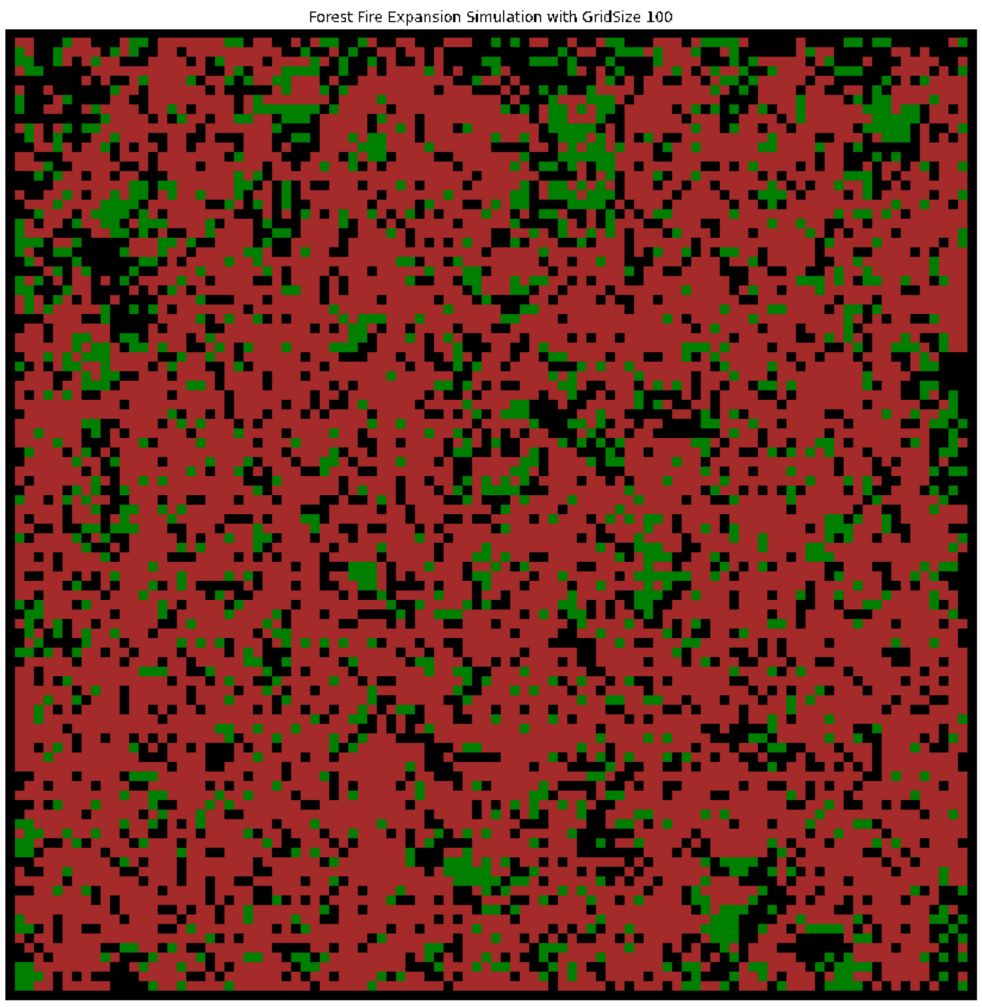


```
def expandTheFireSimulation(extendedforest, size):
    # spreading the fire in the forest
    for i in range(1, size + 1):
        for j in range(1, size + 1):

            # if there is a tree in this position
            if extendedforest[i][j] == TREE:

                # if a neighbour in its moore neighborhoods (# north, south, east, west, north-east, north-west, south-east,
                # is on fire then the tree catches fire
                if (extendedforest[i - 1][j] == TREE_CATCHES_FIRE or extendedforest[i + 1][j] == TREE_CATCHES_FIRE or
                    extendedforest[i][j - 1] == TREE_CATCHES_FIRE or extendedforest[i][j + 1] == TREE_CATCHES_FIRE or
                    extendedforest[i - 1][j - 1] == TREE_CATCHES_FIRE or extendedforest[i - 1][j + 1] == TREE_CATCHES_FIRE or
                    extendedforest[i + 1][j - 1] == TREE_CATCHES_FIRE or extendedforest[i + 1][j + 1] == TREE_CATCHES_FIRE):

                    # the tree will not catch fire it's immune to it
                    if random() < probImmune:
                        extendedforest[i][j] = TREE
                    # otherwise the it will catch fire
                    else:
                        extendedforest[i][j] = TREE_CATCHES_FIRE

                # if lightning strikes the forest at this time, the tree catches fire
                elif random() < probLightning:
                    # the tree will not catch fire it's immune to it
                    if random() < probImmune:
                        extendedforest[i][j] = TREE
                    # otherwise the it will catch fire
                    else:
                        extendedforest[i][j] = TREE_CATCHES_FIRE

                # otherwise, the tree remains
                else:
                    extendedforest[i][j] = TREE

            # if tree already caught fire, then it will burns to the ground
            elif extendedforest[i][j] == TREE_CATCHES_FIRE:
```

Visualization

Visualization of the forest grid was done with Matplotlib that displays the forest grid and executes the applyPeriodicBoundaryConditions and expandTheFireSimulation functions for every iteration in the animation.

Here is an image showing the result of the 100 by 100 forest grid size. More gif images have been submitted with the assignment.

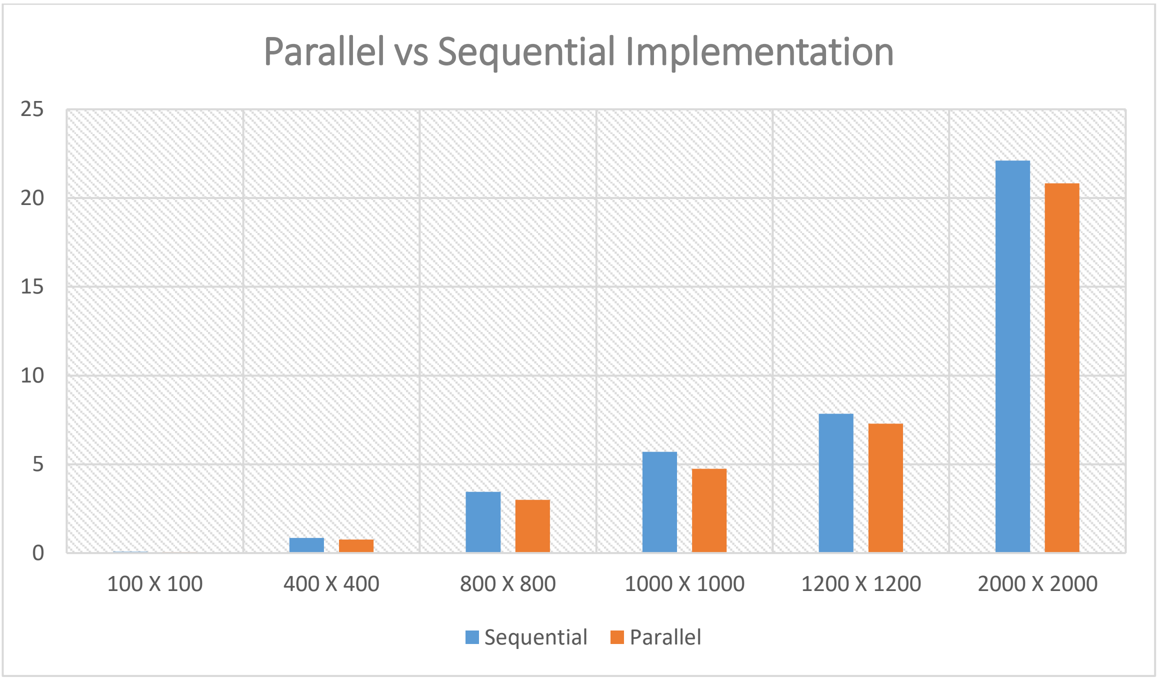


Methods and Result Evaluation

To allow comparison between the parallel and sequential implementations, the time diff

Forest Grid Size	Sequential	Parallel
100 X 100	0.073	0.064
400 X 400	0.86	0.77
800 X 800	3.45	3.00
1000 X 1000	5.71	4.74

1200 X 1200	7.85	7.29
2000 X 2000	22.1	20.83



The simulation was executed at the university computer lab.

The above simulation compares the execution time of sequential and parallel cellular automation models to demonstrate the performance gains of executing simulations sequential and concurrently among computer processing units. As shown in the chart above. The parallelization improved the execution of the simulation.

Future Recommendations

I recommend trying out larger forest grid sizes with parallelization and if possible it should be done on a larger screen, so that the animations can display properly.

Bibliography

Gentle, J. E. (2003) *Random number generation and monte carlo methods*, Second edition. London; New York: Springer.

Shiflet, A. B. & Shiflet, G. W. (2006) *Introduction to computational science: Modeling and simulation for the sciences*. Princeton; Oxford: Princeton University Press.