Report on the Simulation of Forest Fire Spread

Overview

This report aims to describe the processes taken to simulate the spread of fire on an initial forest site filled with burning and non-burning trees. The spread of fire at every location of the forest could be influenced by a number of factors which might include, the closeness of one of the trees in the forest to a burning tree in its Moore neighbourhoods or the possibility of lightning strikes on the forest site. In addition to that, some types of trees are immune to fire such that even though there may be burning influences, they don't get burnt easily. As a result, in the following sections, I aim to describe how this simulation has been achieved according to the coursework specification.

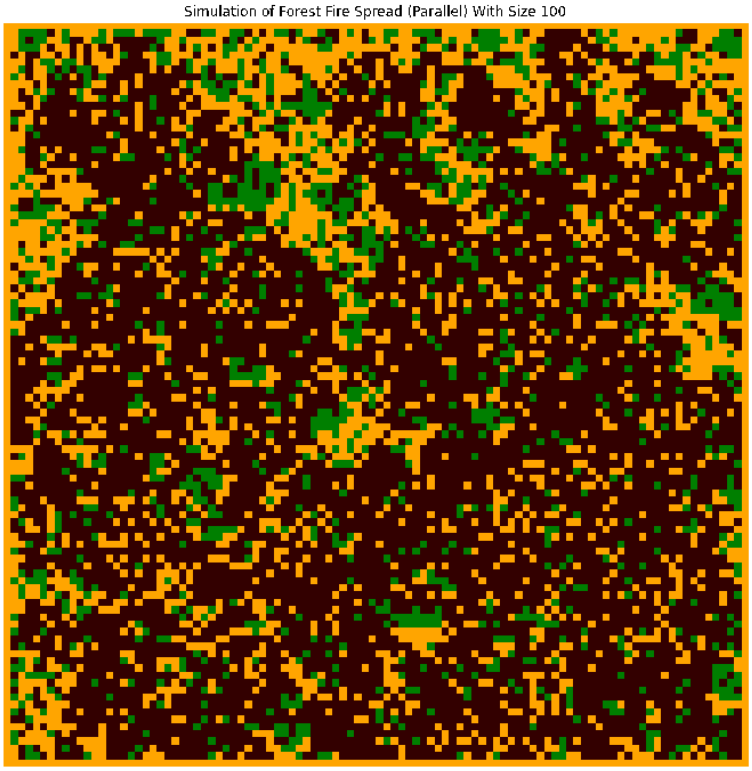


Figure 1- a forest site burning

Simulation Process

To commence the simulation, I needed to gather some details needed to initialize the forest site. This was important before spreading the fire on the site. As a result, this was done with the Numpy library in python. I also created variables for the various probabilities given in the coursework and set values to be inserted in the Numpy grid to represent burning and non-burning trees, empty land, and border. Also based on the assumption that trees are green, and fire red, I picked close colours to represent them in the simulation. Finally, I also assumed the fire flames burn in 300 milliseconds, represented by the animation delay. The image of the section is attached below.

```
GROUND = 0 # represents an empty ground
NON_BURNING_TREE = 1 # represents an area with a non burning tree
BURNING_TREE = 2 # represents an area with a burning tree
BORDERLINE = 3 # represents borderline around the forest

animation_delay = 300 # represents the delay between each frame of the a
bounds = [0, 1, 2, 3]
cmap = colors.ListedColormap([(0.2,0,0), (0,0.5,0), (1,0,0), 'orange'])
norm = colors.BoundaryNorm(bounds, cmap.N)
frame_size = 60
plot_size = 12

probTree = 0.8 # represents the probability that a tree initially ocupie
probBurning = 0.01 # represents the probability that a tree is burning
probImmune = 0.3 # represents the probability that a tree is immune to j
probLightning = 0.001 # represents the probability that an area suffered
```

Figure 2- Gather details for the simulation

To Initialize the forest I created an empty forest site grid depending on the size of the forest given, and filled it with empty lands, and burning and non-burning trees using two probabilities, probTree and probBurning as specified in the specification.

As a result, if at any given location in the forest grid, the probability that there is a tree there is true, the probability that that tree is burning is checked. If the tree is not burning, then, we just set the tree value, otherwise, we set the location as having a burning tree.

```
# here we create a function to initialize the forest site
def initForestState(sizeOfForest):

    # creating a multidimensional array of size sizeOfForest with one row and column
    # of borderline to all sides of the forest.
    # (assumming the site is firstly created with all borderlines),
    forestGrid = np.ones((sizeOfForest + 2, sizeOfForest + 2)) * BORDERLINE

    # now we fill the forest with trees, according to the given probabilities above
    # looping through the forest grids (excluding the borders) and assigning the values
    # for the ground, trees and burning trees
    for i in range(1, sizeOfForest + 1):
        for j in range(1, sizeOfForest + 1):

            # we check for a burning tree and tree at this position
            # but there has to be a tree, for the tree to be burning
            # so let's check for a tree first
            if random() < probTree:
                if random() < probBurning:
                    # the area contains a burning tree
                    forestGrid[i, j] = BURNING_TREE
                else:
                    # the area contains a non burning tree
                    forestGrid[i, j] = NON_BURNING_TREE
            else:
                # the area is empty
                forestGrid[i, j] = GROUND

    # returning the forest grid
    return forestGrid
```

Figure 3- initializing the forest site

Thereafter, we apply the spread rules to the site. But before then, to avoid boundary effects that might prevent us from successfully applying the spread at the borders, I extended the boundaries of the site using

the periodic boundary conditions. This is done by creating ghost areas of opposite sides to make it look like they are both boundaries are connected. Once the ghost areas have been created at the top, left, right and bottom, we then move to apply the spread on the site.

```
    # select only the forest site with no borderlines
    Onlyforest = forest[1:sizeOfForest + 1, 1:sizeOfForest + 1]

    # extend the grid using periodic boundary conditions

    # extending the boundary rows with ghost areas so that we can spread the fire at tl
    rowStack = np.row_stack((Onlyforest[-1,:], Onlyforest, Onlyforest[0,:]))

    # extending the boundary columns with ghost areas so that we can spread the fire at
    forestAfterExtension = np.column_stack((rowStack[:,-1], rowStack, rowStack[:,0]))

    # now we can spread the fire
    appliedSpreadForest = spreadTheFireWithMoore(sizeOfForest, forestAfterExtension)

    # remove the ghost areas and return the borderline before plotting
    Onlyforest = appliedSpreadForest[1:sizeOfForest + 1, 1:sizeOfForest + 1]
    forest[1:sizeOfForest + 1, 1:sizeOfForest + 1] = Onlyforest

    return forest
```

Figure 4 - extending the site boundaries with ghost cells

The spread of fire at a given area on the forest site is determined by burning influences around and the nature of trees in the area. Hence, according to specification, to spread the fire, during a loop across the site, if an area has trees, we check if any of its Moore neighbours (that is, trees at its north, north-east, east, south-east, south, south-west, west, and north-west) contains a burning tree. If one of them contains a burning tree and the tree is not immune, it burns, otherwise, it does not burn. In the same vein, if the probability of lightning strikes is true, and the tree is not immune, it burns, otherwise it does not burn.

And else if an area has a burning tree, we set the grid position as an empty land, because the tree burns to the ground, otherwise, the remaining area on the site remains empty if there's nothing there initially. Please see the figure below.

```
        # see if there's a burning tree next to the area
        # von Neumann neighborhood are 4-neighborhoods (north, south, east and west) but we won't use them
        # the 8 moore neighborhoods (north, north-east, north-west, south, south-east, south-west, west, e
        # can tell us if there's a burning tree
        if (forest[i - 1, j] == BURNING_TREE or forest[i + 1, j] == BURNING_TREE or
            forest[i, j - 1] == BURNING_TREE or forest[i, j + 1] == BURNING_TREE or
            forest[i - 1, j - 1] == BURNING_TREE or forest[i - 1, j + 1] == BURNING_TREE or
            forest[i + 1, j - 1] == BURNING_TREE or forest[i + 1, j + 1] == BURNING_TREE):

            # if the area is immune to fire the tree does not burn
            if random() < probImmune:
                forest[i,j] = NON_BURNING_TREE
            else:
                forest[i,j] = BURNING_TREE

        # if the area suffered a lightning strike the tree burns
        elif random() < probLightning:

            # if the area is immune to fire the tree does not burn
            if random() < probImmune:
                forest[i,j] = NON_BURNING_TREE
            else:
                forest[i,j] = BURNING_TREE

        # else the tree doesn't burn cause there are burning influences
        else:
            forest[i, j] = NON_BURNING_TREE

    # if the area is already a burning tree, then the tree burns to the ground
    elif forest[i, j] == BURNING_TREE:
        forest[i, j] = GROUND

    # otherwise the area remains empty if it's a ground
    else:
        forest[i, j] = GROUND
```

To visualize each iteration of the spread of fire, I used the animate.FunctionAnimation function in Matplotlib to animate the forest fires. It first displays the initialized site and then calls the VisualizeForestFireSpread function which expands forest boundaries and spreads the fire at every iteration. The spread was applied 60 times for each of the forest sizes. Please see the figure below for the code.

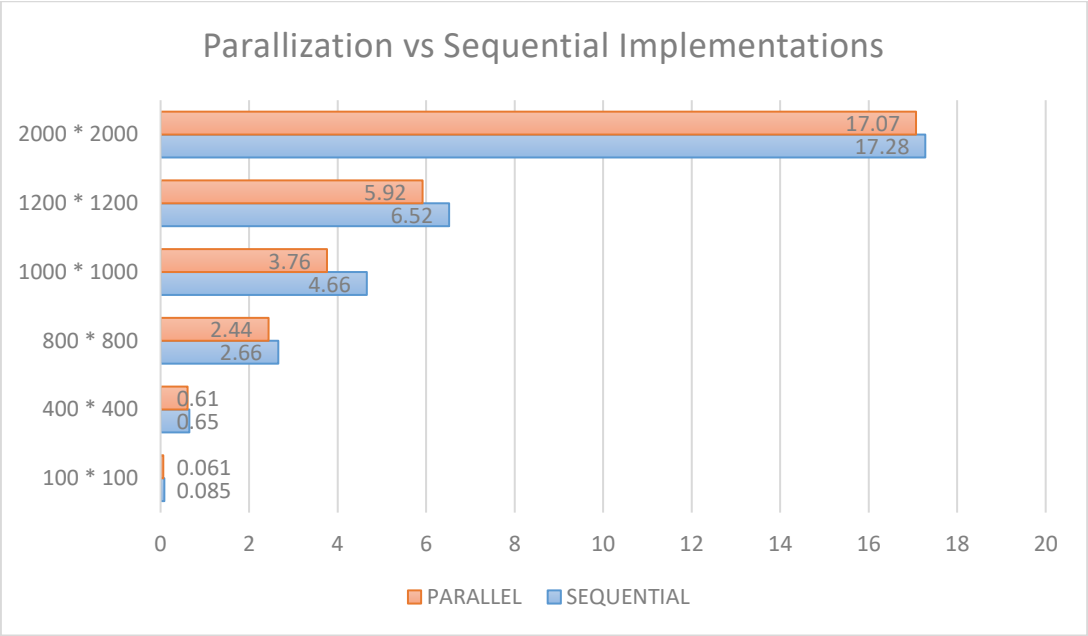
```
fig = plt.figure(num=fig_no, figsize=(plot_size, plot_size))
ax = fig.add_subplot(111)
ax.set_title('Simulation of Forest Fire Spread With Size {0}'.format(gridSize))
ax.set_axis_off()

# plot the forest
forestGridVisual = ax.imshow(forest, cmap=cmap, norm=norm)

def animate(i):
    forestGridVisual.set_array(animate.forest)
    animate.forest = VisualizeForestFireSpread(gridSize, animate.forest)

animate.forest = forest

anim = animation.FuncAnimation(fig, animate, interval=animation_delay, frames=frame_size)

# saving the animation as an mp4 video file
anim.save('fire_spread_animation{0}.gif'.format(gridSize))

plt.show()
```

Figure 5 – animation

Implementations, Results and Evaluations

The Forest fire simulation was implemented in two ways, sequentially and by means of parallelization. The essence of parallelization is the help utilize the processing resources available on the computer efficiently and make the simulations fast. By parallelizing the functions, significant improvements were recorded. See the table below for the time used for a single iteration of forest for each forest size. That is, the time in seconds it takes to initialize the forest, extend the boundaries of the site and spread the fire. This was captured for all forest sizes as displayed in the table below.

FOREST SIZES	SEQUENTIAL	PARALLEL
100 * 100	0.085	0.061
400 * 400	0.65	0.61
800 * 800	2.66	2.44
1000 * 1000	4.66	3.76
1200 * 1200	6.52	5.92
2000 * 2000	17.28	17.07



The parallelization was implemented using the Numba python package that works by compiling the functions before executing. And time was captured using the time python library before and after the execution. Below is an example of how a function was parallelized with Numba.

```
]: # Parallelizing the initForestState function

numba.jit(nopython=True, parallel=True)
def initForestState_Parallel(sizeOfForest):

    # creating a multidimensional array of size sizeOfForest with one row and col
    # of borderline to all sides of the forest.
    # (assumming the site is firstly created with all borderlines),
    forestGrid = np.ones((sizeOfForest + 2, sizeOfForest + 2)) * BORDERLINE

    # now we fill the forest with trees, according to the given probabilities abo
    # looping through the forest grids (excluding the borders) and assigning the
    # for the ground, trees and burning trees
    for i in numba.prange(1, sizeOfForest + 1):
        for j in numba.prange(1, sizeOfForest + 1):

            # we check for a burning tree and tree at this position
            # but there has to be a tree, for the tree to be burning
            # so let's check for a tree first
            if random() < probTree:
                if random() < probBurning:
                    # the area contains a burning tree
                    forestGrid[i, j] = BURNING_TREE
                else:
                    # the area contains a non burning tree
                    forestGrid[i, j] = NON_BURNING_TREE
            else:
```

Figure 6 – Parallelization

```
]: startTime = time.time()
   gridSize = 2000
   fig_no = 6

   forest = initForestState(gridSize)

   forest_test = VisualizeForestFireSpread(gridSize, forest)

   timeElapsed = time.time() - startTime
   print("Time elapsed for applying the spread on forest grid size {0}: {1} seconds".format(gridSize, timeElapsed))
```

Figure 7 – How time was captured

Conclusion

The simulation exercise shows that parallelization improves simulation time, which is a new exposure for me from this activity and seeing how much time difference it makes. However, for simulation of larger forest sites, I would recommend using faster systems with more processing cores to make the speed faster.