Forest Fire Expansion Modelling Simulation Report

Introduction

The goal of this report is to describe how the simulation of a forest fire expansion model was implemented using sequential and parallel processing methods. For the purpose of the simulation, the whole forest was treated as a grid of cells with values that either represent the absence of trees, the presence of non-burning trees or trees that are on fire. This phenomena is used for cellular automation simulations (Shiflet & Shiflet, 2006), for which modelling a forest fire exemplifies. The possibility of fire at any position expanding increases due to external influences like lightning strikes on the forest site and closeness to neighbouring trees that are on fire, depending on the immunity status of the tree.

Initializing and Expanding the Fire

The simulation was implemented using the Python programming language, and some of its libraries namely, Numpy, Numba, Time, Random, and Matplotlib as shown in the Table below.

Python libraries and tools used	Purpose
Numpy	Used for initializing and extending the boundaries of the forest grids.
Numba	Used for compiling and parallelizing the model functions.
Time	Used for capturing time used for a single iteration of the forest fire expansion
Random	Used to generate random values between 0 and 1 to test the probabilities.
Matplotlib	Used for plotting and creating the animation of the forest grids.
Jupyter Notebook	Interactive web-based platform used to write the models and display the forest grid animations.

To initialize and expand the fire on the forest, a grid of zeros (or forest with no trees) were created with Numpy. After which the forest was looped through and the following conditions were applied at every stage.

```
# variables to initializing the forest

NO_TREE = 0 # represents an area with no trees
TREE = 1 # represents an area with trees
TREE_ON_FIRE = 2 # represents an area trees on fire

bounds = [0, 1, 2, 3]
cmap = colors.ListedColormap(['brown', 'green', 'red', 'orange'])
norm = colors.BoundaryNorm(bounds, cmap.N)

interval = 200 # represents the delay between each image of the anim
frame_size = 50 # represents the size of the frame in pixels, or num

probBurning = 0.01 # represents the probability that a tree is on fi
probImmune = 0.3 # represents the probability that a tree is immune
probLightning = 0.001 # represents the probability that an area suff
probTree = 0.8 # represents the probability that an area is occupied
```

The simulation of the forest fire being a Monte-Carlo simulation, (Gentle, 2003) as a result of the probabilities having an element of chance, random generator provided by the python was used to compare against the constant probabilities when filling the forest Grid.

Conditions for site initialization

```
3]: # initialize the forest site
def InitializeForest(forestSize):

    # creating a two-dimensional array of the forestgrid of size forestSize with no trees
    forestGrid = np.zeros((forestSize, forestSize))

    # now we fill the forest with trees, empty area and trees on fire
    for i in range(forestSize):
        for j in range(forestSize):

            # if there is a tree, we set the value of the area to TREE
            if random() < probTree:
                # if the tree is on fire, we set the value of the area to TREE_ON_FIRE
                if random() < probBurning:

                    forestGrid[i][j] = TREE_ON_FIRE
                else:
                    # the tree is not on fire, we set the value of the area to TREE
                    forestGrid[i][j] = TREE
            else:
                # the area is empty
                forestGrid[i][j] = NO_TREE

    # return the forest grid
    return forestGrid
```

Conditions for Fire Expansion

Once the grid is initialized, the forest boundaries are extended using periodic boundary conditions that works by creating invisible areas around the forest boundaries. It was done using the row_stack and column_stack functions provided by Numpy.

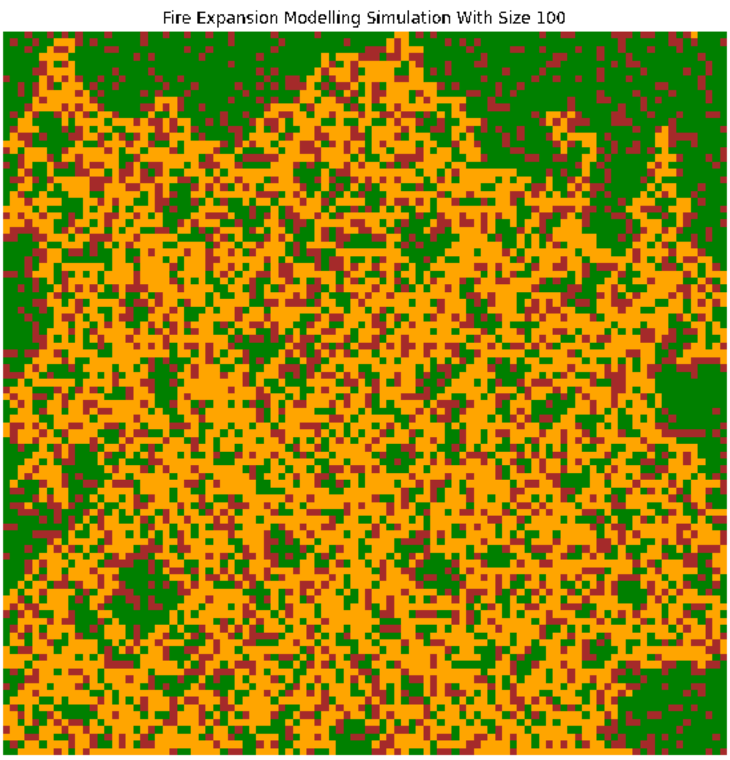
The essence of extending the boundaries is to mitigate boundary effect at the forest borders. And to help check for Moore neighbours of cells at the boundaries. The invisible areas are removed after each iteration.

```
]: # extend the grid using periodic boundary conditions
def ExtendTheForestGrids(forest, forestSize):

    # extending the forest grid boundaries with ghost cells of opposite boundaries
    row_stack = np.row_stack((forest[-1,:], forest, forest[0,:]))
    extendedGrid = np.column_stack((row_stack[:,-1], row_stack, row_stack[:,0]))

    # now we can spread the fire
    extendedGrid = ApplyTheSpreadWithMoore(extendedGrid, forestSize)

    # remove the ghost cells around the boundaries before plotting
    forest = extendedGrid[1:forestSize + 1, 1:forestSize + 1]

    return forest
```

```
]: initTime = time()
```

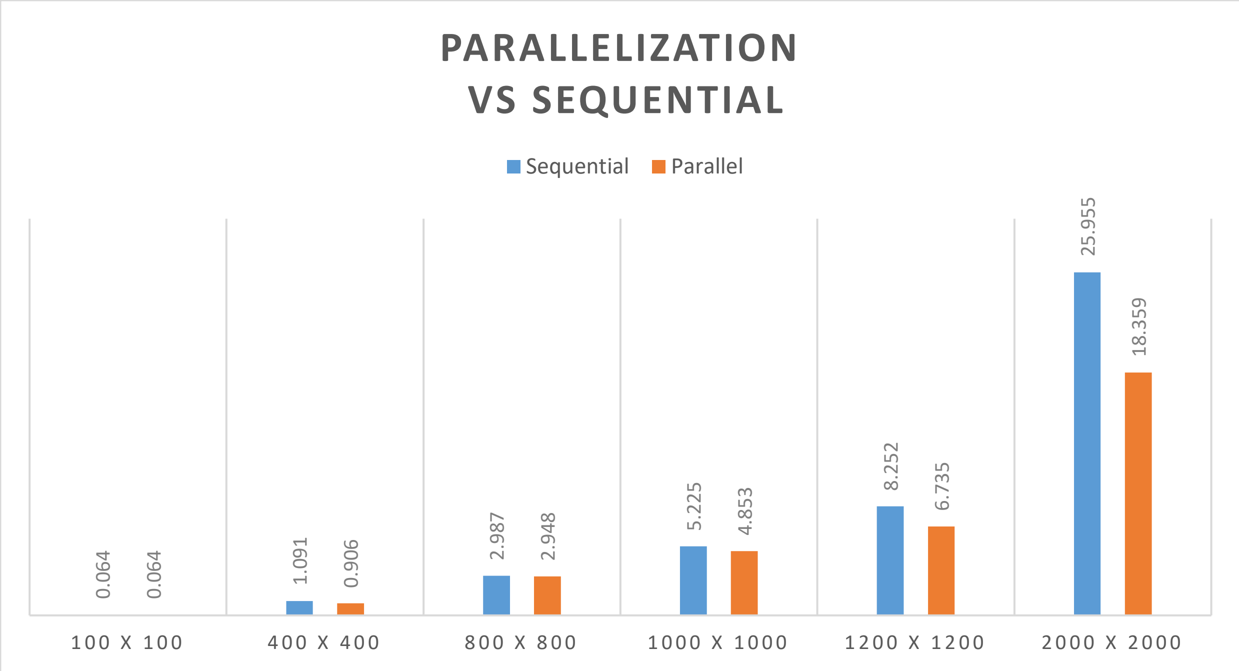
```python
def ApplyTheSpreadWithMoore(forest, forestSize):

    # looping through the forest grids (excluding the borders)
    for i in range(1, forestSize + 1):
        for j in range(1, forestSize + 1):

            # if the area has a tree
            if forest[i][j] == TREE:

                # if a tree on the 8 moore neighborhoods
                # (south, south-east, south-west, west, east, north, north-east, north-west)
                # is on fire, then the tree is likely to burn if it's not immune
                if (forest[i - 1][j] == TREE_ON_FIRE or forest[i + 1][j] == TREE_ON_FIRE or
                    forest[i][j - 1] == TREE_ON_FIRE or forest[i][j + 1] == TREE_ON_FIRE or
                    forest[i - 1][j - 1] == TREE_ON_FIRE or forest[i - 1][j + 1] == TREE_ON_FIRE or
                    forest[i + 1][j - 1] == TREE_ON_FIRE or forest[i + 1][j + 1] == TREE_ON_FIRE):

                    # if the tree is immune to fire it will not burn
                    if random() < probImmune:
                        forest[i][j] = TREE
                    else:
                        forest[i][j] = TREE_ON_FIRE

                # if the forest site suffered a lightning strike the tree is likely burns
                elif random() < probLightning:

                    # # if the tree is immune to fire it will not burn
                    if random() < probImmune:
                        forest[i][j] = TREE
                    else:
                        forest[i][j] = TREE_ON_FIRE

                # else the tree doesn't burn cause there are no external influences
                else:
                    forest[i][j] = TREE
```

Result and Evaluation



The parallelization of the functions was achieved using Numba, as it provides just in time compilation of the functions. However, to improve execution time, Numba recommends to execute the functions one time before trying it again, so that it compiles it in the first trial, and makes significant difference in the execution time.

Forest Grid Size	Sequential	Parallel
100 X 100	0.064	0.064
400 X 400	1.091	0.906
800 X 800	2.987	2.948
1000 X 1000	5.225	4.853
1200 X 1200	8.252	6.735
2000 X 2000	25.955	18.359



Conclusion

I believe I have been able to show how parallelization reduces execution time. However, in the future, I hope to try applying my knowledge from this simulation in other domain areas.

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.