29/11/24

6. Parallel Cellular Algorithms

Algorithm

1. Initialization

1) Define the grid dimensions.
2) Initialize each cell state based on the problem
3) Define the neighborhood for each cell
4) Define the transition rule.

2. Set initial conditions

Populate the grid with initial states for all cells

3. Parallel Update

For each time step t:

For each C(i, j, ---) in the grid (in parallel):

1. Retrieve the current state of C(i, j, ---)
2. Retrieve the states of the neighboring cells
3. Apply the transition rule to compute the next state of C(i, j, ---)

Update all cells simultaneously to their new states

4. Check stopping condition

Terminate if a predefined condition is met.

- A fixed number of iterations completed
- System reaches a steady state
- A specific pattern or result is observed.

5. Output Results

Retrieve the final grid configuration as the solution.

Code

```
import numpy as np
from scipy.ndimage import convolve
import matplotlib.pyplot as plt

rows, cols = 5, 5
grid = np.random.randint(0, 255, size=(rows, cols), dtype=np.uint8)

sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])

sobel_y = np.array([[-1, -2, -1],
                    [0, 0, 0],
                    [1, 2, 1]])

def apply_filter(grid, kernel):
    return convolve(grid, kernel, mode='constant', cval=0)

def update_grid(grid):
    edges_x = apply_filter(grid, sobel_x)
    edges_y = apply_filter(grid, sobel_y)

    new_grid = np.hypot(edges_x, edges_y)
    new_grid = (new_grid / new_grid.max()) * 255

    return new_grid.astype(np.uint8)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Original Image (Random)")
plt.imshow(grid, cmap='gray')

new_grid = update_grid(grid)
```

```
plt.subplot(1,2,2)
plt.title("Edge detection (Sobel) Filter")
plt.imshow(new_grid, cmap='gray')
plt.show()
```