Parallel Cellular Algorithms and Programs

Application:

Image Segmentation

Code:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import pairwise_distances_argmin_min



# Step 1: Get user inputs for image parameters

grid_size_x = int(input("Enter the number of rows in the image grid: "))

grid_size_y = int(input("Enter the number of columns in the image grid: "))

T = int(input("Enter the number of iterations for segmentation: "))

neighborhood_size = int(input("Enter the neighborhood size (odd integer): "))



# Step 2: Generate a random grayscale image

image = np.random.rand(grid_size_x, grid_size_y)  # Random image as an example
(could be grayscale)



# Parameters

G = (grid_size_x, grid_size_y)  # Grid size (user input)

labels = np.random.choice([0, 1], size=(G[0], G[1]))  # Initial random labels
(0 or 1)



# Step 3: Evaluate fitness (pixel intensity difference with neighbors)

def evaluate_fitness(image, labels):
```

```
    fitness_values = np.zeros_like(labels)

    for i in range(G[0]):

        for j in range(G[1]):

            # Fitness based on the difference in pixel intensity with
neighboring cells

            fitness_values[i, j] = np.sum(np.abs(image[i, j] - image[max(i-1,
0):min(i+2, G[0]), max(j-1, 0):min(j+2, G[1])]))

    return fitness_values



# Step 4: Update the state of each cell (pixel label based on the
neighborhood)

def update_cell(i, j, labels, fitness_values):

    # Look at the neighborhood around the cell

    neighborhood = labels[max(i-1, 0):min(i+2, G[0]), max(j-1, 0):min(j+2,
G[1])]

    neighbor_fitness = fitness_values[max(i-1, 0):min(i+2, G[0]), max(j-1,
0):min(j+2, G[1])]



    # Update the cell's label if the neighboring cells have a better fitness

    if np.min(neighbor_fitness) < fitness_values[i, j]:

        # Assign the most frequent label in the neighborhood

        return np.argmin(np.bincount(neighborhood.flatten()))

    else:

        return labels[i, j]  # Return current label if no update is needed



# Step 5: Perform the segmentation in parallel by updating cells iteratively

def parallel_segmentation(image, labels, T):
```

```
    best_labels = labels.copy()

    best_fitness = np.inf



    for t in range(T):

        fitness_values = evaluate_fitness(image, labels)  # Calculate fitness
for each cell



        # In parallel, update each cell based on its neighbors

        new_labels = labels.copy()

        for i in range(G[0]):

            for j in range(G[1]):

                new_labels[i, j] = update_cell(i, j, labels, fitness_values)
# Only update the label here



        # Check if the new labels improve the fitness

        new_fitness = np.sum(fitness_values)

        if new_fitness < best_fitness:

            best_fitness = new_fitness

            best_labels = new_labels.copy()



        labels = new_labels



    return best_labels



# Step 6: Run the segmentation algorithm

final_labels = parallel_segmentation(image, labels, T)
```

```
# Step 7: Visualize the results

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')


plt.subplot(1, 2, 2)

plt.title("Segmented Image")

plt.imshow(final_labels, cmap='tab20')

plt.show()
```

Output :

Enter the number of rows in the image grid: 4

Enter the number of columns in the image grid: 2

Enter the number of iterations for segmentation: 2

Enter the neighborhood size (odd integer): 3

Enter the neighborhood size (odd integer): 3