import numpy as np

# Define the optimization problem: Example is minimizing a quadratic function (e.g., f(x, y) = x^2 + y^2)

def optimization\_function(position):

return position[0]\*2 + position[1]\*2

# Initialize parameters

def initialize\_parameters():

grid\_size = (10, 10) # Grid size (10x10 grid)

num\_iterations = 100 # Number of iterations

neighborhood\_size = 1 # Size of the neighborhood (e.g., Moore neighborhood with radius 1)

return grid\_size, num\_iterations, neighborhood\_size

# Generate an initial population of cells with random positions

def initialize\_population(grid\_size):

population = np.random.uniform(-10, 10, (grid\_size[0], grid\_size[1], 2)) # Random positions in 2D space

return population

# Evaluate the fitness of each cell

def evaluate\_fitness(population):

fitness = np.zeros((population.shape[0], population.shape[1]))

for i in range(population.shape[0]):

for j in range(population.shape[1]):

fitness[i, j] = optimization\_function(population[i, j])

return fitness

# Update the state of each cell based on neighboring cells

def update\_states(population, fitness, neighborhood\_size):

updated\_population = np.copy(population)

for i in range(population.shape[0]):

for j in range(population.shape[1]):

# Determine the neighborhood boundaries

x\_min = max(i - neighborhood\_size, 0)

x\_max = min(i + neighborhood\_size + 1, population.shape[0])

y\_min = max(j - neighborhood\_size, 0)

y\_max = min(j + neighborhood\_size + 1, population.shape[1])

# Find the best neighbor (minimum fitness)

best\_neighbor = population[i, j]

best\_fitness = fitness[i, j]

for x in range(x\_min, x\_max):

for y in range(y\_min, y\_max):

if fitness[x, y] < best\_fitness:

best\_neighbor = population[x, y]

best\_fitness = fitness[x, y]

# Update the cell's position toward the best neighbor

updated\_population[i, j] = (population[i, j] + best\_neighbor) / 2

return updated\_population

# Main function to implement the Parallel Cellular Algorithm

def parallel\_cellular\_algorithm():

# Step 1: Initialize parameters

grid\_size, num\_iterations, neighborhood\_size = initialize\_parameters()

# Step 2: Initialize population

population = initialize\_population(grid\_size)

# Step 3: Iterate over the algorithm

best\_solution = None

best\_fitness = float('inf')

for iteration in range(num\_iterations):

# Step 4: Evaluate fitness

fitness = evaluate\_fitness(population)

# Track the best solution

min\_fitness = np.min(fitness)

if min\_fitness < best\_fitness:

best\_fitness = min\_fitness

best\_solution = population[np.unravel\_index(np.argmin(fitness), fitness.shape)]

# Step 5: Update states

population = update\_states(population, fitness, neighborhood\_size)

# Print progress

print(f"Iteration {iteration + 1}: Best Fitness = {best\_fitness}")

# Step 6: Output the best solution

print(f"Best Solution: {best\_solution}, Best Fitness: {best\_fitness}")

return best\_solution, best\_fitness

# Run the algorithm

if \_name\_ == "\_main\_":

parallel\_cellular\_algorithm()