```python
import numpy as np
import random

# Step 1: Define the Problem (Optimization Function)
def fitness_function(position):
    """Example fitness function: Sphere function"""
    return sum(x**2 for x in position)

# Step 2: Initialize Parameters
grid_size = (10, 10)  # Grid size (10x10 cells)
dim = 2  # Dimensionality of each cell's position
minx, maxx = -10.0, 10.0  # Search space bounds
max_iterations = 50  # Number of iterations

# Step 3: Initialize Population (Random positions)
def initialize_population(grid_size, dim, minx, maxx):
    population = np.zeros((grid_size[0], grid_size[1], dim))
    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            population[i, j] = [random.uniform(minx, maxx) for _ in
range(dim)]
    return population

# Step 4: Evaluate Fitness (Calculate fitness for each cell)
def evaluate_fitness(population):
    fitness_grid = np.zeros((grid_size[0], grid_size[1]))
    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            fitness_grid[i, j] = fitness_function(population[i, j])
    return fitness_grid

# Step 5: Update States (Update each cell based on its neighbors)
def get_neighbors(i, j):
    """Returns the coordinates of neighboring cells."""
    neighbors = []
    for di in [-1, 0, 1]:
        for dj in [-1, 0, 1]:
            if not (di == 0 and dj == 0):  # Exclude the cell itself
                ni, nj = (i + di) % grid_size[0], (j + dj) % grid_size[1]
                neighbors.append((ni, nj))
```

```
    return neighbors

def update_cell(population, fitness_grid, i, j, minx, maxx):
    """Update the state of a cell based on the average state of its
neighbors."""
    neighbors = get_neighbors(i, j)
    best_neighbor = min(neighbors, key=lambda x: fitness_grid[x[0], x[1]])

    # Update cell position to move towards the best neighbor's position
    new_position = population[best_neighbor[0], best_neighbor[1]] + \
                   np.random.uniform(-0.1, 0.1, dim)  # Small random
perturbation

    # Ensure the new position stays within bounds
    new_position = np.clip(new_position, minx, maxx)
    return new_position

# Step 6: Iterate (Repeat for a fixed number of iterations)
population = initialize_population(grid_size, dim, minx, maxx)
for iteration in range(max_iterations):
    fitness_grid = evaluate_fitness(population)

    # Update each cell in parallel (simultaneously)
    new_population = np.zeros_like(population)
    for i in range(grid_size[0]):
        for j in range(grid_size[1]):
            new_population[i, j] = update_cell(population, fitness_grid,
i, j, minx, maxx)

    population = new_population

    # Print best fitness at each iteration
    best_fitness = np.min(fitness_grid)
    print(f"Iteration {iteration + 1}, Best Fitness: {best_fitness}")

# Step 7: Output the Best Solution
best_index = np.unravel_index(np.argmin(fitness_grid), fitness_grid.shape)
best_position = population[best_index[0], best_index[1]]
best_fitness = np.min(fitness_grid)
print("Best Position Found:", best_position)
```

```
print("Best Fitness Found:", best_fitness)
```

```
Iteration 24, Best Fitness: 0.0002113674736078839
Iteration 25, Best Fitness: 0.00031670680514341276
Iteration 26, Best Fitness: 8.749816523333429e-05
Iteration 27, Best Fitness: 6.674629634635891e-05
Iteration 28, Best Fitness: 8.576319639727489e-05
Iteration 29, Best Fitness: 0.00039857908161797706
Iteration 30, Best Fitness: 0.0002973724117323112
Iteration 31, Best Fitness: 6.896879335991218e-05
Iteration 32, Best Fitness: 0.00020011848699482218
Iteration 33, Best Fitness: 0.0001359440055165302
Iteration 34, Best Fitness: 0.0004178896328875618
Iteration 35, Best Fitness: 1.0176660032292261e-05
Iteration 36, Best Fitness: 6.915483191347572e-05
Iteration 37, Best Fitness: 0.00010260147831570198
Iteration 38, Best Fitness: 1.0576922123165615e-05
Iteration 39, Best Fitness: 0.0002547450683222206
Iteration 40, Best Fitness: 1.5666459142473822e-06
Iteration 41, Best Fitness: 3.4347455503344177e-06
Iteration 42, Best Fitness: 0.00041413837551337244
Iteration 43, Best Fitness: 4.13764187766922e-05
Iteration 44, Best Fitness: 0.0001569781296599252
Iteration 45, Best Fitness: 6.106147331445545e-05
Iteration 46, Best Fitness: 0.0002838860337482483
Iteration 47, Best Fitness: 2.6713429266698007e-05
Iteration 48, Best Fitness: 8.144509163252784e-06
Iteration 49, Best Fitness: 7.672313960845285e-05
Iteration 50, Best Fitness: 6.750446484473827e-05
Best Position Found: [-0.00763284 -0.01939989]
Best Fitness Found: 6.750446484473827e-05
```