Lab 7
Parallel Cellular Algorithm

```
[4]
✓ 0s
import numpy as np
def objective_function(x):
    return x**2 - 4*x + 4
def parallel_cellular_algorithm(obj_func, grid_size=(10, 10), lb=-10, ub=10, max_iter=100):
    rows, cols = grid_size
    num_cells = rows * cols
    cells = np.random.uniform(lb, ub, (rows, cols))
    fitness = obj_func(cells)
    def get_neighbors(r, c):
        neighbors = []
        for i in range(r-1, r+2):
            for j in range(c-1, c+2):
                if (0 <= i < rows) and (0 <= j < cols) and not (i == r and j == c):
                    neighbors.append((i, j))
        return neighbors
    best_value = np.min(fitness)
    best_position = np.unravel_index(np.argmin(fitness), fitness.shape)
    for t in range(max_iter):
        new_cells = np.copy(cells)
        for r in range(rows):
            for c in range(cols):
                neighbors = get_neighbors(r, c)
                neighbor_values = np.array([cells[i, j] for i, j in neighbors])
```

```
[4]
✓ 0s
            for r in range(rows):
                for c in range(cols):
                    neighbors = get_neighbors(r, c)
                    neighbor_values = np.array([cells[i, j] for i, j in neighbors])
                    neighbor_fitness = np.array([obj_func(cells[i, j]) for i, j in neighbors])
                    best_neighbor_value = neighbor_values[np.argmin(neighbor_fitness)]
                    new_cells[r, c] = (cells[r, c] + best_neighbor_value) / 2
            new_cells = np.clip(new_cells, lb, ub)
            cells = new_cells
            fitness = obj_func(cells)
            current_best = np.min(fitness)
            if current_best < best_value:
                best_value = current_best
                best_position = np.unravel_index(np.argmin(fitness), fitness.shape)
            if (t + 1) % 10 == 0 or t == 0:
                print(f"Iteration {t+1}/{max_iter} → Best fitness: {best_value:.6f}")
        print("\nBest solution found:")
        print(f"x = {cells[best_position]:.6f}, fitness = {best_value:.6f}")
        return cells[best_position], best_value
    best_x, best_fit = parallel_cellular_algorithm(
        objective_function,
        grid_size=(10, 10),
        lb=-10,
        ub=10,
```

[4]
✓ 0s

```
            new_cells = np.clip(new_cells, lb, ub)
            cells = new_cells
            fitness = obj_func(cells)
            current_best = np.min(fitness)
            if current_best < best_value:
                best_value = current_best
                best_position = np.unravel_index(np.argmin(fitness), fitness.shape)
            if (t + 1) % 10 == 0 or t == 0:
                print(f"Iteration {t+1}/{max_iter} → Best fitness: {best_value:.6f}")
        print("\nBest solution found:")
        print(f"x = {cells[best_position]:.6f}, fitness = {best_value:.6f}")
        return cells[best_position], best_value
    best_x, best_fit = parallel_cellular_algorithm(
        objective_function,
        grid_size=(10, 10),
        lb=-10,
        ub=10,
        max_iter=60
    )
```

```
    )

•••  Iteration 1/60 → Best fitness: 0.001454
     Iteration 10/60 → Best fitness: 0.000000
     Iteration 20/60 → Best fitness: 0.000000
     Iteration 30/60 → Best fitness: 0.000000
     Iteration 40/60 → Best fitness: 0.000000
     Iteration 50/60 → Best fitness: 0.000000
     Iteration 60/60 → Best fitness: 0.000000

     Best solution found:
     x = 2.000000, fitness = 0.000000
```