PARALLEL CELLULAR ALGORITHM

Code:

```
import numpy as np
import random

# Objective function (Sphere function)
def objective_function(x):
    return np.sum(x ** 2)

# Initialize the grid (population)
def initialize_grid(grid_size, dim, bounds):
    return np.random.uniform(bounds[0], bounds[1], (grid_size, grid_size, dim))

# Evaluate fitness of the grid
def evaluate_grid(grid, objective_function):
    fitness = np.zeros((grid.shape[0], grid.shape[1]))
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
            fitness[i, j] = objective_function(grid[i, j])
    return fitness

# Selection using the best individual in the neighborhood
def select_best_neighbor(grid, fitness, x, y):
    neighbors = [
        ((x - 1) % grid.shape[0], y),   # Up
        ((x + 1) % grid.shape[0], y),   # Down
        (x, (y - 1) % grid.shape[1]),   # Left
        (x, (y + 1) % grid.shape[1]),   # Right
    ]
    best_pos = min(neighbors, key=lambda pos: fitness[pos[0], pos[1]])
```

```
    return grid[best_pos[0], best_pos[1]]

# Crossover operation
def crossover(parent1, parent2):
    alpha = np.random.rand()
    return alpha * parent1 + (1 - alpha) * parent2

# Mutation operation
def mutate(individual, bounds, mutation_rate=0.1):
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            individual[i] += np.random.uniform(-1, 1)
            individual[i] = np.clip(individual[i], bounds[0], bounds[1])
    return individual

# Main Parallel Cellular Genetic Algorithm
def parallel_cellular_ga(objective_function, grid_size=5, dim=2, bounds=(-5, 5), max_iter=100,
mutation_rate=0.1):
    # Initialize the grid and fitness
    grid = initialize_grid(grid_size, dim, bounds)
    fitness = evaluate_grid(grid, objective_function)

    for iteration in range(max_iter):
        new_grid = np.copy(grid)

        for i in range(grid_size):
            for j in range(grid_size):
                # Select parents from the neighborhood
                parent1 = grid[i, j]
                parent2 = select_best_neighbor(grid, fitness, i, j)
```

```
                # Apply crossover and mutation
                offspring = crossover(parent1, parent2)
                offspring = mutate(offspring, bounds, mutation_rate)

                # Replace if offspring is better
                offspring_fitness = objective_function(offspring)
                if offspring_fitness < fitness[i, j]:
                    new_grid[i, j] = offspring
                    fitness[i, j] = offspring_fitness

        grid = new_grid

        # Output the best solution in the grid
        best_position = np.unravel_index(np.argmin(fitness), fitness.shape)
        best_fitness = fitness[best_position]
        print(f"Iteration {iteration + 1}: Best Fitness = {best_fitness}")

    # Return the best solution
    best_position = np.unravel_index(np.argmin(fitness), fitness.shape)
    return grid[best_position[0], best_position[1]], fitness[best_position]

# Parameters
grid_size = 5        # Size of the grid
dim = 2              # Dimensionality of the problem
bounds = (-5, 5)     # Search space boundaries
max_iter = 50        # Number of iterations
mutation_rate = 0.1  # Mutation rate

# Run PCGA
best_solution, best_fitness = parallel_cellular_ga(objective_function, grid_size, dim, bounds, max_iter, mutation_rate)
```

```
# Output the best solution
print(f"\nBest solution: {best_solution}")
print(f"Best fitness: {best_fitness}")
```

Output:

```
Iteration 35: Best Fitness = 1.4700677504738945e-07
Iteration 36: Best Fitness = 1.4700677504738945e-07
Iteration 37: Best Fitness = 1.4700677504738945e-07
Iteration 38: Best Fitness = 1.4700677504738945e-07
Iteration 39: Best Fitness = 1.4700677504738945e-07
Iteration 40: Best Fitness = 1.4700677504738945e-07
Iteration 41: Best Fitness = 1.4700677504738945e-07
Iteration 42: Best Fitness = 1.4700677504738945e-07
Iteration 43: Best Fitness = 1.4700677504738945e-07
Iteration 44: Best Fitness = 1.4700677504738945e-07
Iteration 45: Best Fitness = 1.4700677504738945e-07
Iteration 46: Best Fitness = 1.4700677504738945e-07
Iteration 47: Best Fitness = 1.4700677504738945e-07
Iteration 48: Best Fitness = 1.4700677504738945e-07
Iteration 49: Best Fitness = 1.4700677504738945e-07
Iteration 50: Best Fitness = 1.4700677504738945e-07

Best solution: [ 0.00036358 -0.00012172]
Best fitness: 1.4700677504738945e-07
```