LAB 6:

PARALLEL CELLULAR ALGORITHM:

```
#LAB 6:
#PARALLEL CELLULAR ALGORITHM

import numpy as np
import random

def objective_function(x):
    return np.sum(x ** 2)

def initialize_population(grid_size, solution_space_dim):
    population = np.random.uniform(-5.12, 5.12, (grid_size, grid_size, solution_space_dim))
    return population

def evaluate_fitness(population):
    fitness_grid = np.zeros(population.shape[:-1])
    for i in range(population.shape[0]):
        for j in range(population.shape[1]):
            fitness_grid[i, j] = objective_function(population[i, j])
    return fitness_grid

def update_cell(population, fitness_grid, i, j, neighborhood_size=1):
    neighbors = []
    for di in range(-neighborhood_size, neighborhood_size + 1):
        for dj in range(-neighborhood_size, neighborhood_size + 1):
            ni, nj = i + di, j + dj
            if 0 <= ni < population.shape[0] and 0 <= nj < population.shape[1]:
                neighbors.append((ni, nj))
    avg_position = np.zeros(population.shape[2])
    for ni, nj in neighbors:
        avg_position += population[ni, nj]
    avg_position /= len(neighbors)
    population[i, j] = avg_position + 0.01 * np.random.randn(*avg_position.shape)

def parallel_cellular_algorithm(grid_size, solution_space_dim, max_iterations=1000):
    population = initialize_population(grid_size, solution_space_dim)
    fitness_grid = evaluate_fitness(population)

    best_solution = None
    best_fitness = float('inf')

    for iteration in range(max_iterations):
        new_population = population.copy()

        for i in range(population.shape[0]):
            for j in range(population.shape[1]):
                update_cell(new_population, fitness_grid, i, j)

        fitness_grid = evaluate_fitness(new_population)

        min_fitness_idx = np.unravel_index(np.argmin(fitness_grid), fitness_grid.shape)
        min_fitness_value = fitness_grid[min_fitness_idx]

        if min_fitness_value < best_fitness:
            best_fitness = min_fitness_value
            best_solution = new_population[min_fitness_idx]
```

```
        population = new_population

        print(f"Iteration {iteration + 1}: Best Fitness = {best_fitness}")

    return best_solution, best_fitness
```

```
grid_size = 10
solution_space_dim = 2
max_iterations = 50
```

```
best_solution, best_fitness = parallel_cellular_algorithm(grid_size, solution_space_dim,
max_iterations)
```

```
print("Best Solution Found:", best_solution)
print("Best Fitness Value:", best_fitness)
```

OUTPUT:

```
Best Solution Found: [-8.57032959e-06 -6.30756966e-05]
Best Fitness Value: 4.051994051566561e-09
```