Lab 7
Parallel Cellular Algorithms

```
Custom-function (x):
    return sum (x ** 2)

Initialize-population (grid-side, dim, lower, upper)
    population ← random (lower, upper, (grid-size, dim))

    return population

Evaluate-fitness (population, fitness-function)
    fitness ← zeros (population.shape[-1])
    for i in range (population.shape[0])
        for j in range (population.shape[1])

            fitness[i][j] ← custom-function (population[i,j])

    return fitness.

update-states (pop, fit, radius, lower, upper):
    grid-size, dim ← pop shape

    new-pop ← pop.copy()

    for i in 0 to grid-size:
        for j in 0 to grid-size:
            neighbours ← get-neighbors (i, j, grid-size, radius)
            best-neighbour ← None
            best-fitness ← float (inf)
```

```
if best_neighbour:
    ri, rj ← best_neighbor

    new_population[i,j] ← pop[ri, rj]
        + random(-0.5, 0.5, dim)

    new_population[i,j] ← clip(new_
    population[i,j], lower, upper)

return new_population

get_neighbors(i, j, grid_size, radius)
    neighbors ← []
    for di in range(-radius, radius+1):
        ni, nj ← (i+di) % grid_size,
                 (j+dj) % grid_size

        if (di != 0 or dj != 0)
            neighbors.append((ni, nj))

    return neighbors

Parallel_cellular_algorithm(grid_size, dim,
lower, upper, max_iters, radius, fitness_
                                    func)
    population ← initialize_population(grid_size,
    dim, lower, upper)

    fitness ← evaluate_fitness(population,
    fitness_function)

    min_fitness ← fitness.min()
    if min_fitness < best_fitness
        best_fitness ← min_fitness
```

```
best_sol ← None
best_fit ← ∞
for iter in max_iters
        pop ← update_states (population, fitness,
              radius, lower, upper)

        fitness ← evaluate_fitness (pop,
                  fitness_function)

        min_fitness ← fitness.min()
        if min_fitness < best_fitness :
                best_fitness ← min_fitness
                best_sol ← population (np.
                  unravel_index (fitness.min(),
                    fitness.shape)]

        print (Iteration, fitness)

return best_sol, best_fit
```

Inputs :

```
grid_size = 10
dim = 1
lower = -5
upper = 5
max_iters = 100
radius = 1
```