USN: 1BM22CS253

LAB-6: Parallel Cellular Algorithms and Programs

CODE:

```
#pcap import numpy
as np


# Define the problem: A simple op miza on func on (e.g., Sphere Func on) def
op miza on_func on(posi on):
    """Example: Sphere Func on for minimiza on."""
return sum(x**2 for x in posi on)


# Ini alize Parameters
GRID_SIZE = (10, 10)  # Grid size (rows, columns)
NEIGHBORHOOD_RADIUS = 1  # Moore neighborhood radius
DIMENSIONS = 2  # Number of dimensions in the solu on space
ITERATIONS = 30  # Number of itera ons


# Ini alize Popula on def ini alize_popula on(grid_size, dimensions):     """Ini alize a grid
with random posi ons."""      popula on = np.random.uniform(-10, 10, size=(grid_size[0],
grid_size[1], dimensions))     return popula on


# Evaluate Fitness def
evaluate_fitness(popula on):
    """Calculate the fitness of all cells."""      fitness =
np.zeros((popula on.shape[0], popula on.shape[1]))     for i in
range(popula on.shape[0]):         for j in range(popula
on.shape[1]):
```

```
        fitness[i, j] = op miza on_func on(popula on[i, j])
    return fitness


# Get Neighborhood def
get_neighborhood(grid, x, y, radius):

    """Get the neighbors of a cell within the specified
radius."""      neighbors = []      for i in range(-radius, radius +
1):         for j in range(-radius, radius + 1):             if i == 0
and j == 0:                 con nue  # Skip the current cell

            ni, nj = x + i, y + j             if 0 <= ni < grid.shape[0]
and 0 <= nj < grid.shape[1]:

                neighbors.append((ni, nj))
return neighbors


# Update States def
update_states(popula on, fitness):

    """Update the state of each cell based on its neighbors."""      new_popula on =
np.copy(popula on)     for i in range(popula on.shape[0]):        for j in range(popula
on.shape[1]):                         neighbors = get_neighborhood(popula on, i, j,
NEIGHBORHOOD_RADIUS)

            best_neighbor = popula on[i, j]
best_fitness = fitness[i, j]


            # Find the best posi on among neighbors
            for ni, nj in neighbors:
                if fitness[ni, nj] < best_fitness:
best_fitness = fitness[ni, nj]
best_neighbor = popula on[ni, nj]             # Update
```

```
the cell state (move towards the best neighbor)
new_popula on[i, j] = (popula on[i, j] +
best_neighbor) / 2  # Average posi on     return
new_popula on


# Main Algorithm def
parallel_cellular_algorithm():
    """Implementa on of the Parallel Cellular Algorithm."""
popula on = ini alize_popula on(GRID_SIZE, DIMENSIONS)
best_solu on = None
    best_fitness = float('inf')

    for itera on in range(ITERATIONS):
        # Evaluate fitness         fitness =
evaluate_fitness(popula on)


        # Track the best solu on         min_fitness = np.min(fitness)         if min_fitness <
best_fitness:             best_fitness = min_fitness             best_solu on = popula
on[np.unravel_index(np.argmin(fitness), fitness.shape)]


        # Update states based on neighbors         popula
on = update_states(popula on, fitness)


        # Print progress         print(f"Itera on {itera on + 1}: Best
Fitness = {best_fitness}")
```

print("\nBest Solu on Found:") print(f"Posi on:

{best_solu on}, Fitness: {best_fitness}")

# Run the algorithm
if __name__ == "__main__":

parallel_cellular_algorithm()

OUTPUT: