Lab – 6 : Parallel Cellular Algorithm

```python
import random
import numpy as np

# Initialize Grid (N x M) with random states (0 or 1)
def initialize_grid(N, M):
    return np.random.choice([0, 1], size=(N, M))

# Count live neighbors for a cell (i, j)
def count_live_neighbors(grid, i, j, N, M):
    return sum(
        grid[ni, nj] for ni in range(i-1, i+2) for nj in range(j-1, j+2)
        if 0 <= ni < N and 0 <= nj < M and (ni != i or nj != j)
    )

# Update cell's state based on neighbors' count
def update_cell(grid, new_grid, i, j, N, M):
    live_neighbors = count_live_neighbors(grid, i, j, N, M)
    if grid[i, j] == 1:
        new_grid[i, j] = 1 if live_neighbors in [2, 3] else 0
    else:
        new_grid[i, j] = 1 if live_neighbors == 3 else 0

# Print the current state of the grid (0s and 1s)
def print_grid(grid):
    for row in grid:
        print(' '.join(map(str, row)))
    print()  # For spacing between generations

# Main Game of Life simulation with printing grid
def parallel_game_of_life(N, M, steps):
    grid = initialize_grid(N, M)

    for _ in range(steps):
        print_grid(grid)  # Print current grid state
        new_grid = np.zeros((N, M), dtype=int)  # New grid for next state
        for i in range(N):
            for j in range(M):
                update_cell(grid, new_grid, i, j, N, M)
        grid = new_grid  # Update grid for next iteration

    print_grid(grid)  # Print final grid after all steps

# Example Usage
N, M = 5, 5  # Smaller grid size for readability
steps = 5  # Number of iterations
final_grid = parallel_game_of_life(N, M, steps)
```

Output:

```
0 0 0 0 0
0 1 0 0 0
0 0 0 1 1
0 1 0 1 1
1 1 1 0 1

0 0 0 0 0
0 0 0 0 0
0 0 0 1 1
1 1 0 0 0
1 1 1 0 1

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
1 0 0 0 1
1 0 1 0 0

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 1 0 0 0
0 1 0 0 0

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```