```
import numpy as np
import multiprocessing
from multiprocessing import Pool

# Initialize the grid with random values
def initialize_grid(width, height):
    return np.random.choice([0, 1], size=(width, height))

# Function to compute the next state of a single cell
def compute_cell_state(i, j, grid):
    # Get the 3x3 neighborhood around (i, j)
    neighborhood = grid[i-1:i+2, j-1:j+2]
    # Count how many live cells (1s) are in the neighborhood
    live_neighbors = np.sum(neighborhood) - grid[i, j]  # Subtract the center cell itself
    # Apply the Game of Life rules:
    if grid[i, j] == 1 and (live_neighbors < 2 or live_neighbors > 3):
        return 0  # Cell dies due to underpopulation or overpopulation
    elif grid[i, j] == 0 and live_neighbors == 3:
        return 1  # Cell becomes alive due to reproduction
    return grid[i, j]  # Cell stays in the same state

# Function to update the grid in parallel
def update_grid_parallel(grid):
    height, width = grid.shape
    # Create an empty grid for the next state
    new_grid = np.zeros_like(grid)

    # Use multiprocessing to apply compute_cell_state in parallel
    with Pool(processes=multiprocessing.cpu_count()) as pool:
        # Create a list of all the grid cell indices (i, j)
        indices = [(i, j) for i in range(1, height-1) for j in range(1, width-1)]
```

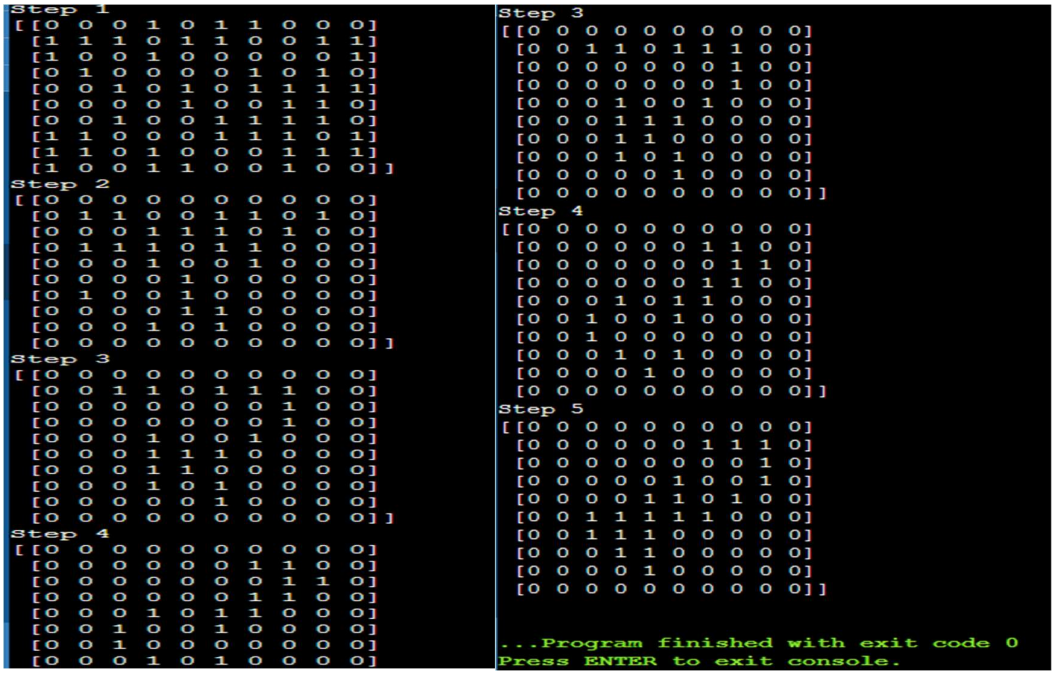
```
        # For each (i, j), compute the next state
        results = pool.starmap(compute_cell_state, [(i, j, grid) for (i, j) in indices])

        # Place the results in the new grid
        idx = 0
        for i in range(1, height-1):
            for j in range(1, width-1):
                new_grid[i, j] = results[idx]
                idx += 1

    return new_grid

# Main loop to run the simulation
def run_game_of_life(width, height, steps):
    grid = initialize_grid(width, height)
    for step in range(steps):
        print(f"Step {step+1}")
        print(grid)
        grid = update_grid_parallel(grid)

# Run the simulation
if __name__ == '__main__':
    run_game_of_life(10, 10, 5)  # 10x10 grid, 5 steps
```


Step 1
[[0 0 0 1 0 1 1 0 0 0]
[1 1 1 0 1 1 0 0 1 1]
[1 0 0 1 0 0 0 0 0 1]
[0 1 0 0 0 0 1 0 1 0]
[0 0 1 0 1 0 1 1 1 1]
[0 0 0 0 1 0 0 1 1 0]
[0 0 1 0 0 1 1 1 1 0]
[1 1 0 0 0 1 1 1 0 1]
[1 1 0 1 0 0 0 1 1 1]
[1 0 0 1 1 0 0 1 0 0]]
Step 2
[[0 0 0 0 0 0 0 0 0 0]
[0 1 1 0 0 1 1 0 1 0]
[0 0 0 1 1 1 0 1 0 0]
[0 1 1 1 0 1 1 0 0 0]
[0 0 0 1 0 0 1 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 1 0 0 1 0 0 0 0 0]
[0 0 0 0 1 1 0 0 0 0]
[0 0 0 1 0 1 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]]
Step 3
[[0 0 0 0 0 0 0 0 0 0]
[0 0 1 1 0 1 1 1 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 1 0 0 1 0 0 0]
[0 0 0 1 1 1 0 0 0 0]
[0 0 0 1 1 0 0 0 0 0]
[0 0 0 1 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]]
Step 4
[[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 1 1 0 0]
[0 0 0 0 0 0 0 1 1 0]
[0 0 0 0 0 0 1 1 0 0]
[0 0 0 1 0 0 1 0 0 0]
[0 0 1 0 0 1 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 1 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]]
Step 3
[[0 0 0 0 0 0 0 0 0 0]
[0 0 1 1 0 1 1 1 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 1 0 0 1 0 0 0]
[0 0 0 1 1 1 0 0 0 0]
[0 0 0 1 1 0 0 0 0 0]
[0 0 0 1 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]]
Step 4
[[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 1 1 0 0]
[0 0 0 0 0 0 0 1 1 0]
[0 0 0 0 0 0 1 1 0 0]
[0 0 0 1 0 1 1 0 0 0]
[0 0 1 0 0 1 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]]
Step 5
[[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 1 1 1 0]
[0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 1 0 0 1 0]
[0 0 0 0 1 1 0 1 0 0]
[0 0 1 1 1 1 1 0 0 0]
[0 0 1 1 1 0 0 0 0 0]
[0 0 0 1 1 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]]
...Program finished with exit code 0
Press ENTER to exit console.