```python
import numpy as np
import concurrent.futures

# Define the size of the grid
GRID_SIZE = 5

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

# Count live neighbors of a given cell (i, j)
def count_neighbors(grid, i, j):
    # Get indices of neighbors (8 directions)
    neighbors = [
        (-1, -1), (-1, 0), (-1, 1),
        (0, -1),          (0, 1),
        (1, -1), (1, 0), (1, 1)
    ]
    count = 0
    for dx, dy in neighbors:
        ni, nj = i + dx, j + dy
        if 0 <= ni < grid.shape[0] and 0 <= nj < grid.shape[1]:
            count += grid[ni, nj]
    return count

# Define the Game of Life rules and parallel update function
def game_of_life_step(grid):
    new_grid = np.copy(grid)

    def update_cell(i, j):
        live_neighbors = count_neighbors(grid, i, j)
        if grid[i, j] == 1:
            if live_neighbors < 2 or live_neighbors > 3:
                new_grid[i, j] = 0  # Cell dies
        elif grid[i, j] == 0 and live_neighbors == 3:
            new_grid[i, j] = 1  # Cell becomes alive

    # Use ThreadPoolExecutor to update cells in parallel
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = []
        for i in range(grid.shape[0]):
            for j in range(grid.shape[1]):
                futures.append(executor.submit(update_cell, i, j))

        # Wait for all threads to finish
        concurrent.futures.wait(futures)

    return new_grid

# Print the grid for visualization
def print_grid(grid):
    for row in grid:
        print(" ".join(str(cell) for cell in row))

# Main simulation loop
def simulate_game_of_life(steps=10):
    grid = initialize_grid(GRID_SIZE)
    print("Initial Grid:")
    print_grid(grid)

    for step in range(steps):
        print(f"\nStep {step + 1}:")
        grid = game_of_life_step(grid)
        print_grid(grid)

# Run the simulation
simulate_game_of_life(steps=5)
```

```
Initial Grid:
0 0 0 1 0
0 1 0 1 0
0 1 0 0 1
0 1 1 0 1
1 1 0 0 1
```

```
Step 1:
0 0 1 0 0
0 0 0 1 1
1 1 0 0 1
0 0 1 0 1
1 1 1 1 0

Step 2:
0 0 0 1 0
0 1 1 1 1
0 1 1 0 1
0 0 0 0 1
0 1 1 1 0

Step 3:
0 0 0 1 1
0 1 0 0 1
0 1 0 0 1
0 0 0 0 1
0 0 1 1 0

Step 4:
0 0 0 1 1
0 0 1 0 1
0 0 0 1 1
0 0 1 0 1
0 0 0 1 0

Step 5:
0 0 0 1 1
0 0 1 0 0
0 0 1 0 1
0 0 1 0 1
0 0 0 1 0
```