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

# Initialize the grid with random state (0 or 1)
def initialize_grid(width, height):
    grid = np.random.randint(2, size=(width, height))  # Random 0s and 1s
    return grid

# Count the number of alive neighbors for a given cell
def count_alive_neighbors(grid, x, y, width, height):
    count = 0
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                continue  # Skip the cell itself
            neighbor_x = (x + dx + width) % width  # Handle wrapping around edges
            neighbor_y = (y + dy + height) % height
            count += grid[neighbor_x][neighbor_y]
    return count

# Determine the next state of a cell based on its neighbors
def next_state(grid, x, y, width, height):
    alive_neighbors = count_alive_neighbors(grid, x, y, width, height)
    if grid[x][y] == 1:
        return 1 if alive_neighbors == 2 or alive_neighbors == 3 else 0
    else:
        return 1 if alive_neighbors == 3 else 0

# Function to update the grid in parallel
def parallel_update(grid, width, height):
    next_grid = np.zeros((width, height), dtype=int)

    # Create a thread pool to update the grid in parallel
    def update_cell(x, y):
        next_grid[x][y] = next_state(grid, x, y, width, height)

    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = []
        for x in range(width):
            for y in range(height):
                # Submit the task of updating each cell to the thread pool
                futures.append(executor.submit(update_cell, x, y))

        # Wait for all futures to complete
        concurrent.futures.wait(futures)

    return next_grid

# Function to run the simulation for a set number of steps
def run_simulation(width, height, num_steps):
    grid = initialize_grid(width, height)
    for step in range(num_steps):
        print(f"Step {step + 1}")
        grid = parallel_update(grid, width, height)
        display_grid(grid)  # Display or process the current state of the grid
        time.sleep(0.1)  # Sleep for a brief moment to visualize changes (optional)

# Function to display the grid (print it as 0s and 1s)
def display_grid(grid):
    for row in grid:
        print(" ".join(str(cell) for cell in row))
    print("\n" + "-" * (len(grid[0]) * 2))  # Just a separator for visualization

# Function to take user input for grid size and number of simulation steps
def get_user_input():
    width = int(input("Enter the width of the grid: "))
    height = int(input("Enter the height of the grid: "))
    num_steps = int(input("Enter the number of steps to run the simulation: "))
    return width, height, num_steps

# Example usage
if __name__ == "__main__":
    # Get user input for grid size and number of steps
    width, height, num_steps = get_user_input()
```

```
# Run the simulation with the user's input
run_simulation(width, height, num_steps)
```

```
Enter the width of the grid: 7
Enter the height of the grid: 7
Enter the number of steps to run the simulation: 10
Step 1
0 0 0 1 0 0 0
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
1 1 0 0 0 0 1
0 1 0 0 0 0 0
0 1 0 0 0 0 0

--------------
Step 2
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 0 0 0 0 0 0
1 1 0 0 0 0 0
0 1 1 0 0 0 0
0 0 1 0 0 0 0

--------------
Step 3
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 1 0 0 0 0 0
1 0 1 0 0 0 0
1 0 1 0 0 0 0
0 1 1 0 0 0 0

--------------
Step 4
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 1 0 0 0 0 0
1 0 1 0 0 0 1
1 0 1 1 0 0 0
0 1 1 0 0 0 0

--------------
Step 5
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 1 0 0 0 0 1
0 0 1 1 0 0 1
1 0 0 1 0 0 1
0 1 1 1 0 0 0

--------------
Step 6
0 0 1 0 0 0 0
0 0 0 0 0 0 0
1 0 0 0 0 0 0
1 1 1 0 0 0 1
```

Start coding or generate with AI.