Research Report on:
[Effective Simulation Execution of Cellular Automata on GPU]

Sanket Dandge(12140540), Josyula Venkata Aditya(12140840), Shubham Daule(12141550)

1 Problem Statement

Cellular Automata (CA), such as Conway's Game of Life, require repeated and simultaneous updates of cell states based on neighborhood rules, making them computationally intensive, especially on large grids. In this phase, we aim to implement a basic version of Conway's Game of Life on GPU using CUDA to lay the groundwork for further optimization and performance enhancements in subsequent phases.

2 Approach and Implementation

2.1 Overview

In this phase, the simulation is parallelized by assigning each cell of the 2D grid to a separate CUDA thread. Each thread is responsible for computing the next state of its respective cell based on the standard rules of the Game of Life, using the current state of neighboring cells.

2.2 Technical Details

The core computation is handled by the `computeNextGenKernel`, a CUDA kernel that determines the state of each cell for the next generation. Each thread computes the number of living neighbors using the `countNeighbors` function. The rules applied are:

- A cell with exactly three living neighbors becomes alive.
- A cell with two living neighbors remains in its current state.
- All other cells die or remain dead.

To handle boundary conditions, we avoid processing edge cells in this version. Additional kernels are defined for updating ghost rows, columns, and corners.

2.3 Implementation

The simulation begins by allocating memory on the GPU for both the current and next generation grids. The grid is copied from the host (CPU) to the device (GPU), and the kernel is launched with a grid of thread blocks (32x32). After computation, the resulting grid is copied back from device to host.

CUDA error-checking macros are used to ensure proper memory handling and kernel execution. The implementation currently supports square grids of size $N \times N$ and assumes a 2D block and grid configuration calculated from N.

3 Preliminary Results

3.1 Comparison

The table below compares the execution time of benchmark results with our proposed GPU-based implementation across various patterns.

Patterns	Reference Implementation (µs)	Our Implementation (µs)
104p177.rle	1612020	1845649
2c5-spaceship-gun-p416.rle	675873	819086
ak94.rle	210593	247627
blinker.rle	183423	247572
blockandglider.rle	203580	251610
empty.rle	195532	254224
floodgate_predecessor.rle	224748	257349
glider.rle	204306	256802
gosper_gun.rle	212238	256728
heisenburp-30.rle	260071	333301
infinity-hotel1.rle	314679	437710
p46racetrack.rle	729460	819951
reflectors.rle	603096	820034
stargate.rle	253131	297103
toad.rle	213007	257543

Table 1: Execution time comparison for various Game of Life patterns

References:

GitHub – bryanoliveira/cellular-automata

GitHub - Sanket-Dandge/Cellular-Automata-GPU