Parallel Renderer for Conway's Game of Life

Gustaf Ahdritz (gwa2107) and Lucie le Blanc (ll3163)

November 22, 2020

1 Background

Conway's Game of Life is a simple cellular automaton, one of a family of algorithms in which cells living on a uniform, finite grid are assigned states over time according to some universal ruleset. First proposed in 1970 by namesake John Conway, it distinguishes itself among such automata for its large public following, which has spent decades populating it with all manner of whimsical creatures: a self-replicating entity called "Gemini,"[1] functioning Turing machines[2], and tiny spaceships that drift across the plane.

In the Game of Life, cells are either "living" or "dead". The game begins in some initial configuraton, defining the state of each cell. At each timestep, whether any given cell lives or dies depends only on the states of its neighbors, including diagonal ones. If fewer than two neighbors are alive, it dies, probably of loneliness. More than three, and it dies of overpopulation. A dead cell with exactly three neighbors comes back to life. The "game" itself consists of watching the starting state evolve, sometimes endlessly. It can be translated into the following computational task: given a starting state and some frame number n, compute and produce an ASCII animation of the n-th or first n timesteps.

(a) t=0 (b) t=1

Figure 1: A single timestep in the Game of Life

[1] https://bit.ly/35Y9FjI
[2] https://bit.ly/2URUsKO

2 Goals

The game is simple, but our Haskell implementation doesn't have to be. There are a number of degrees of freedom involved, and we'll have do do some experimentation to settle on a reasonable implementation. Our choice of data structure might be a simple 2D array, but we could also try to implement the quadtrees involved in "Hashlife," a complex solver designed to be performant across billions of iterations. Since the rules only apply locally, the game is quite easily parallelizable, but the exact choice of parallelization strategy is also non-trivial; we will have to determine whether it makes more sense to parallelize across cells, columns/rows, individual patches of the board, multiple timesteps, or a combination of these. While traditional renderings of the Game of Life have focused on step-by-step iteration and visualization, being able to "fast-forward" through many iterations would make the parallelization effort more meaningful and would reduce time lost on I/O.

Given enough time, we could additionally consider extending the renderer to accept different rulesets, including more complicated criteria that involve neighborhoods larger than the immediate surroundings of a cell.