Assignment 3 Report

Uwe Dauernheim[†]
uwe@dauernheim.net

Alex Averbuch[†]
alex.averbuch@gmail.com

[†] IV1200 SYSTEM MODELING AND SIMULATION – KTH

October 12, 2009

1 Introduction

The outcome of *The Game of Life* is determined by its initial state, it has no input beyond creating the initial configuration.

The Game of Life's universe is a 2D grid of square cells. Each cell is in one of two states, "alive" or "dead". All cells interact with their eight "neighbours", which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Live cells with less than two live neighbours die, due to underpopulation.
- Live cells with over three live neighbours die, due to overcrowding.
- Live cells with two or three live neighbours live on to the next generation.
- Dead cells with exactly three live neighbours become live cells again.

The initial configuration is the system's seed. The first generation is created by applying the above rules to all cells in the seed. The rules are then repeatedly applied to create each subsequent generation.

2 Implementations

The model is implemented in Java and the specific model is described in the following section.

Model The grid is represented as a two-dimensional `boolean[][]` array and – by the concept of information hiding – only directly accessible from within the `Grid` class.

Initialization The following grid attributes are initialized according to input parameters: *width*, *height*, *size*, *time*.

After the initialization, the main transformation part starts transforming all cells. Two different implementation algorithms are given in Section 2.1 and Section 2.2.

2.1 Sequential

In each time iteration the whole grid gets transformed into a temporary new grid in an "all-or-nothing" approach and then stored back to the original grid, as shown in Figure 1.

This is done by looping over all cells in the current generation's grid, applying the defined rules and storing the output of this operation in the future grid. By this we ensure that there is no data inconsistency during the transformation between two time steps.

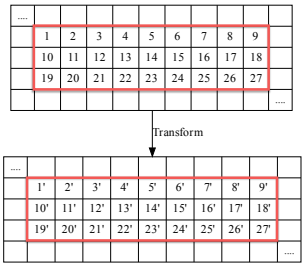

Fig. 1: Transformation cell area in the sequential algorithm

2.2 Parallel

For the implementation of the parallel algorithm we have to keep some assumptions from the sequential algorithm and make some more assumptions:

Threads There exists a single thread – in our solution called *worker* – for each sub-grid. Every worker is created at the start of each time iteration and destroyed at the end of each time iteration.

Ghost cells A ghost cell represents the border of a sub-grid. A worker is not transforming these cells but rather needs it for the transformation process of its own cells.

Sub-grid A sub-grid is a square part of the entire grid containing $(\text{size}+2)^2$ cells. Thus, each sub-grid contains the *ghost* border containing its immediate neighbour cells. This allows us to implement a conflict resolution by letting each worker transform its cells independently without a shared memory overlap.

The transformation process as well as area on which each worker is transforming cells is illustrated in Figure 2.

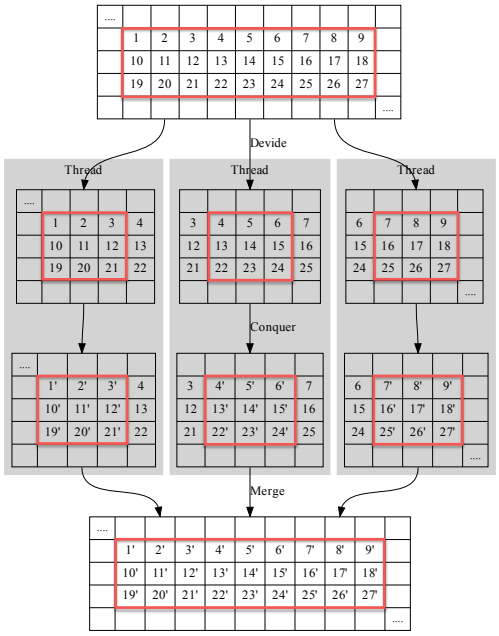

Fig. 2: Cell area to work on for each worker

Each worker gets a sub grid to work on, reading from a shared memory. Then, the new results are written to a temporary new grid, which becomes the normal grid in the next time step.

The synchronization between workers is implemented by using a Java `Thread.join()` operation allowing the coordination program to wait for each worker until their work is done and they have been destroyed.

■