Conway’s Game of Life

Setup

- 2D matrix of size n,n containing binary values
- cell := a coordinate in the matrix and the assigned value
- living cells := any cell with value 1
- dead cells := any cell with value 0
- neighbours of a cell := all cells adjacent in at least one dimension
- state of the game := coordinates of all living cells
- initial state := {(x_1, y_1), (x_2, y_2), …, (x_n, y_n)}

Matrix implementation

Many possible ways of implementations

- 1 to 1 implementation in memory
- linked-list
- 2d hash table

Calculating next state

Next state of the game follows from the current state and a set of rules:

- cell becomes dead for any living cell, if 2 > sum(neighbours) > 3
- cell becomes alive for any dead cell, if sum(neighbours) > 2

Hence to actually calculate the next state of the game, we need to calculate for all cells of the matrix the sums of their neighbours.

Assuming however we deal with a sparse matrix, and seeing that activity propagates from alive and not dead cells, we can simplify this method:

DEAD = 0
ALIVE = 1

```
count_matrix = matrix(n+2, n+2) // count matrix includes border around matrix
next_matrix = matrix(n, m)

for any living node (x, y) in old_matrix:
    inc_neighbors(x,y)

for any living node (x,y) in border_case:
    inc_neighbors(x,y)

function inc_neighbors(x,y):
    inc_count_matrix[x][y+1]
    inc_count_matrix[x][y-1]
    inc_count_matrix[x-1][y+1]
    inc_count_matrix[x-1][y-1]
    inc_count_matrix[x+1][y+1]
    inc_count_matrix[x+1][y-1]
    inc_count_matrix[x+1][y]
    inc_count_matrix[x-1][y]

function inc_count_matrix(x, y):
    x += 1 // account for greater size of
    y += 1 //  count matrix
    count_matrix(x, y) += 1
    if count_matrix(x, y) > 3:
        next_matrix(x,y) = DEAD
    if count_matrix(x, y) > 2:
        next_matrix(x,y) = ALIVE
```

Be aware the matrix implementation can vary.

Parallel processing

Partition

The matrix is assumed to have the shape of a square and is divided into a set of squares. Every square is governed by a node deciding this part of the game.

An alternative would be to divide the matrix into adjacent columns. Thereby the number of recipients of messages would be reduced to max two, meanwhile the process overhead of deciding destination is smaller, so does memory reading overhead (for variant A only). However the circumference of the areas would rise and make it necessary to send greater amount of data. (The bordering areas are becoming bigger.)

Algorithm

Every node governing a region sends an message to all adjacent regions, containing all coordinates of living cells directly bordering the region:

Coordinates

a	b	c	d
e	f	g	h
k	l	m	n
o	p	q	r

State

0	1	0	1
0	0	0	0
0	1	0	1
0	0	1	1

Region green will send region red it has an active node at b
Region blue will send region green, red and blue its active node at l
Region yellow will send region red and blue its active nodes at n and q respectively

Synchronization

Additionally every region will send to every adjacent region which it has not informed previously an empty message (similar to a heartbeat) to make it aware it has proceeded to a further time step.

Thereby any node will receive a message by any adjacent node before it starts calculating another step.