Grid-based Game of Life Parallel Algorithm

The presented algorithm builds on and improves upon the already posted solutions to the Conway's Game of Life.

The algorithm consists of two stages: setup and simulation.

Setup

We divide the 2D matrix with initial state into a grid of non-overlapping tiles. To improve load distribution, we may use tiles of unequal sizes, still it is crucial to preserve the grid layout. The goal of irregular grid is to distribute initial cells evenly or, using some heuristic algorithm.

We then assign the tiles to nodes. Ideally we would have as many nodes as we have tiles, and nodes would be connected through grid (or similar) interconnect.

Simulation

Each simulation consists of three steps (which are then repeated, possibly in an endless loop):

1. Solve local problem
2. Communicate conflicts
3. Solve conflicts

Solving local problem

In the first step the goal is to solve the tile on each one as if it were a completely isolated and independent matrix (which often is the case!) with **one significant difference**: each node must be aware of cells that are on the edge of a tile. The fate of those cells **cannot be decided locally**, hence the algorithm goes as follows:

we apply the usual rules of game of life, but before we change a state of a cell, we check if it is a cell living on the edge, if so then instead of just using usual states of *starved* (death due overpopulation), *suicided* (death due underpopulation), *survived* and *born* we also set a *possibly* bit to one.

Each cell with *possibly* bit set is henceforth called a conflict.

Communicating conflicts

Hence we nodes cannot decide the fate of some cells alone, their fate must be decided through communication. Each node, that has *right* (`R`), *bottom* (`B`) or *bottom right* (`C`) neighbour sends them the current state of the bordering cells:

```
 _____     _____
|    R| > |
|    R| > | R+C
|B_B_C| > |

 V V V  _\|
 _____     _____
      |   |
  B+C |   |  C
      |   |
```

For clarity we only illustrate one sending node and ignore the other sends.

Resolving conflicts

Each node that received the conflict lists proceeds to analyse them locally and resolve them using the available information (the messages from the neighbours and the local state). The precise rules of conflict resolution are not yet specified.

After resolving the conflicts the information flows in the opposite direction to the previous step:

```
 _____     _____
|    R| < |
|    R| < | R+C
|B_B_C| < |
         _
 ^ ^ ^  |\
 _____     _____
      |   |
  B+C |   |  C
      |   |
```

Each node that receives the conflict resolutions, then proceeds to change the state of the cells previously marked with the *possibly* bit.

Algorithm Assessment

The presented algorithm is especially well suited for message-passing based parallel systems with grid-like interconnect. It is is easy to see that most of the data is exchanged between nodes that are directly connected in the system.

Assuming that the living cells are evenly distributed among the tiles throughout the simulation, this approach optimally distributes the load.

The algorithm is self-synchronising and does not require a coordination. Each node computes independently and synchronises through conflict resolution with neighbors (this requires synchronous receive in the third step).

During each round `6*(N-1)` messages are passed (where `N` is the number of nodes and each node is assigned only one tile).

Disclaimer

This algorithm has not beed proven to work, nor it has been tested. It has not even been really well thought through (it is missing an important part!). Therefore: