

CS442/ECE432: Project 2

April 1, 2008

General

- This project is **due April 15, 2008**.
- Submit via e-mail to riesen@cs.unm.edu (Mail it before class on the 15th.) Your subject line must say: “Project 2 Submission”
- Your program must be in a single file called `ex_01.c` (or `ex_01.cc` for C++ programs)
- Your program must compile without warnings (we use the `-Wall` flag for gcc)
- Name your description file `ex_01.pdf`
- Put your files into a tar or zip file called `Proj2.tgz` (or `Proj2.zip`)

1 The Game of Life

1.1 Introduction

The following is a description of the rules for Conway’s game of Life from http://en.wikipedia.org/wiki/Conway's_Game_of_Life

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead. Every cell interacts with its eight neighbors, which are the cells that are directly horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if by loneliness.
2. live cell with more than three live neighbors dies, as if by overcrowding.
3. live cell with two or three live neighbors lives, unchanged, to the next generation.
4. dead cell with exactly three live neighbors comes to life.

The initial pattern constitutes the first generation of the system. The second generation is created by applying the above rules simultaneously to every cell in the first generation – births and deaths happen simultaneously, and the discrete moment at which this happens is sometimes called a tick. (In other words, each generation is based entirely on the one before.) The rules continue to be applied repeatedly to create further generations.

```
16 # X dimension
12 # Y dimension
100 # Number of cycles to simulate
0, 0 # Cell at coordinate 0, 0 is alive
15, 11 # This is a neighbor of the cell above!
3, 2
3, 3
3, 4
```

Figure 1: Example input file for the game of life

1.2 Project

Your task is to implement the above rules in a parallel program. Your program must run on an arbitrary number of processors; i.e., it must be size independent. The grid of cells in your simulation wraps around in both dimensions.

Initially, your program must read a configuration file that specifies the size of the playing grid, and which cells are initially alive. The input file also contains the number of cycles (ticks) your simulation must execute. Figure 1 shows an example input file.

White space (spaces and tab stops) in the input file should be ignored. The `#` character begins a comment. It and any other characters until the end of the line should be ignored. The first number in the input file specifies the horizontal size of the grid your program must simulate. The second number determines the vertical size. These dimensions can be arbitrarily large. You can assume the grid will fit into the aggregate memory of all the nodes allocated to the job, but your program will not know until run time how much that is. That means you will have to allocate memory dynamically and do error checking to make sure you have enough room to run the problem.

The third integer is the number of cycles (i.e. updates) you must simulate. After that follows an arbitrary long list of cells which are initially alive. All others are assumed to be dead. The cells need not be sorted, and duplication is allowed.

Only one node should read the input file. If more than one node needs that information, it is your task to distribute it to those other nodes. The `MPI_Bcast()` operation might be of use here, but something more sophisticated could also be done to send initialization information only to those nodes that actually need it.

Decide how you will decompose the problem. One obvious way is to assign rectangular portions of the grid to each node. Your decomposition must be such that all nodes will perform about the same amount of work as any other. Furthermore, each node can only store a portion of the grid. No node must have the whole grid (or all live cells) stored.

Even though one bit is enough to represent each cell, feel free to use a whole byte or word to store the state of a given cell.

After the simulation print the list of alive cells one per line using the x, y scheme from the input file. This list is not required to be sorted. Then print some statistics:

- The number of cycles simulated and the size of the grid
- The number of life cells at the end
- The total number of cell deaths and births

Figure 2 shows an example.

```
1 11, 9
2 12, 3
3 0, 0
4 Simulated 100 cycles on a 16 x 12 grid
5 3 cells alive at end
6 Total cell deaths: 93
7 Total cell births: 13
```

Figure 2: Example output from the program

5 points For a brief description of how you solved the problem

10 points For correctly reading and parsing the input file on a single node

5 points For distributing the information from the input file to all the nodes that need it

5 points For correctly simulating the game of life

5 points For an even distribution of the work load

5 points For correct and complete output at the end

10 points For dealing correctly with wraparound of the grid

10 points For a scalable solution; i.e. one that works with arbitrary size grids and number of processors