

CS 240A: Applied Parallel Computing // Homework 2

Assigned January 30, 2006

Due by class time Wednesday, February 8

The object of this problem is to write a parallel program to simulate a cellular automaton called the Game of Life. As originally defined, Life takes place on an infinite two-dimensional grid of squares, each of which is either empty or occupied by an organism. Each grid square has eight neighbors: two horizontal, two vertical, and four diagonal. Time moves in discrete steps called generations. At each generation, the organisms are born, live, and die, according to the following rules.

- If an empty square has exactly three occupied neighbors, a new organism is born in that square and it becomes occupied.
- If an occupied square has exactly two or exactly three occupied neighbors, it remains occupied.
- If an occupied square has more than three occupied neighbors, its organism dies of overcrowding and the square becomes empty.
- If an occupied square has fewer than two occupied neighbors, its organism dies of loneliness and the square becomes empty.

The Game of Life was defined by the mathematician John Horton Conway; it became well-known when Martin Gardner wrote about it in *Scientific American* in 1970. You can find a lot more about it on the web, for example at

http://en.wikipedia.org/wiki/Conway's_Game_of_Life.

Many computational simulations of physical phenomena have the same structure as Life (but with more complicated rules): space is modelled as a two- or three-dimensional grid of cells; time is modelled by individual, discrete ticks of a clock (or generations); and at each clock tick the state of each cell is updated depending on the previous state of the cell and its neighbors.

One immediate question is how to simulate an infinite grid of cells with a finite computer. For this homework we will use a finite n -by- n array of cells, with n as large as possible, and we will wrap the grid around at the top and bottom and sides, forming a torus. That is, we will consider the rightmost grid squares to have the leftmost grid squares as their right-hand neighbors, and similarly the top squares will be neighbors of the bottom squares. In the lingo of partial differential equations, we are imposing “periodic boundary conditions”.

Life is pretty simple to write as a sequential program; there’s a Matlab code on the course web site. (The Matlab code includes a harness with a data generator and validator, as in the first homework; you should use it to check the correctness of your code. You will write your code to work with the C and Fortran harnesses provided in the class discussion group.) Making Life run efficiently in parallel is, not surprisingly, a matter of two things: data distribution and communication. For data distribution, you will probably first distribute the array of cells across the processors by rows

(though at least in theory a two-dimensional block distribution might do less communication for large sizes). For communication, you may want to consider so-called “ghost cells”, in which the part of the array assigned to each processor includes a copy of the first layer of cells assigned to each adjacent processors.

A more sophisticated version of Life might exploit the sparsity of the array, avoiding computation and storage for at least some of the large areas of empty cells. We won't ask you to worry about that issue for this assignment.

As decided by the coin flip in class, those of you whose Google group member email begins with letters A through H will do this problem in MPI (either C or Fortran), and J through Z will do it in UPC (or CAF). Email me if you're not sure which group you're in. The groups will swap for the third homework.

Grading on this problem will be 50% on correctness and 50% on performance. (Don't panic! You can get a good grade on performance by careful choice of data layout and communication.) For performance, we will measure your code running on the sample data in the test harness, on the largest possible array sizes. The measure of performance will be a combination of raw speed (on a fixed number of processors and a fixed array size) and speedup (ratio of time on one processor to time on p processors).

You should do this assignment on DataStar at SDSC. (If you want to do it on a different machine, talk to the instructor or the t.a. first.)

The file `hw2/faq.text` has some details of how to get things working. We'll be adding to that file during the course of the assignment as questions come up.