CS 140 Assignment 5:
Life in the Fast Lane

Assigned Wednesday, February 11, 2015

Due by 11:55 pm Monday, February 23, 2015

The object of this problem is to implement a cellular automaton called the Game of Life in Cilk Plus, and to tune it to get maximum performance. The program itself is pretty simple, but there are subtle issues in performance tuning.

There's a prize for the fastest program; details are below.

1 Background

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 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. Many computational simulations of physical phenomena have the same structure as Life (but usually 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.

2 Implementation

One immediate question is how to simulate an infinite grid of cells with a finite computer. For this project you will use a finite n-by-n array of cells, with n as large as possible, and you will wrap the grid around at the top and bottom and sides, forming a torus. That is, you 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, you are imposing "periodic boundary conditions."

Life is pretty simple to write as a sequential program. The course web site has a Matlab code for Life that includes visualization, generators for three sample starting positions, and a Matlab harness that validates that the output is correct for some of the starts. You should play with the Matlab codes until you understand them thoroughly.

Your assignment is to implement Life in Cilk Plus on Triton. The goal of the assignment is to get the best possible running time for 1000 iterations of Life on the biggest grid you can fit into the memory on one node of Triton. Anything you can do to speed this up is legal. There is a Cilk Plus harness linked to the GauchoSpace page.

You will want to experiment with different ways of decomposing the data for the divide-and-conquer step: should you split up the array by rows, by columns, by blocks, or how? You may want to experiment with using "ghost cells," in which each piece of the array includes some redundant data from the neighboring pieces. You might want to browse the specs for the sizes of Triton's caches as part of your optimization.

You can debug your code on any machine, including your laptop or CSIL, and using any version of Cilk. However, the runs you turn in must be on Triton using Cilk Plus. We will compile your code on Triton with `icc` under Cilk Plus to grade it for both correctness and performance.

3 What experiments to do

As usual, begin by getting your code thoroughly debugged, using the test inputs provided on the website along with the validation harness. To get any credit for performance, the output of your code must be correct. We will test your code on our own starting configurations to verify correctness.

Most of your grade will be based on the performance of your code. For performance testing, your goal is to run the largest possible boards on the maximum possible number of cores, for 1000 iterations. You should measure performance in "CUPS", which stands for Cells Updated Per Second. If you can run an n-by-n board for 1000 iterations in s seconds, you are doing $1000 \times n^2/s$ CUPS. Your CUPS measurements should be done using various numbers of cores on one node of Triton.

You should turn in plots of CUPS versus n and versus p, and also plots of parallel efficiency t_1/pt_p versus n and versus p.

We will test your code for large n and large p on a starting configuration of our own choosing. The authors of the code with the best CUPS performance on Triton will be awarded a prize consisting of a large quantity of chocolate.

Please make sure to follow the harness's specification for the output of your program. You'll use lots of debugging output while you're writing and tuning your program, but the version you turn in should have exactly the specified output—leftover debugging output makes it harder for the grading scripts, and we might not be able to give you credit for a correct run.

You should do this assignment in a team of two people. Be sure to put the names of both members both on your report and also in your code.