Computer Science 338 - Project 2

Three-dimensional Cellular Automata

code due: 11:59 PM, Friday, October 13, 2000

writeup due: 11:59 PM, Friday, October 20, 2000

For this project, you will implement a distributed three-dimensional cellular automata simulator using MPI.

Background

Cellular automata have many applications in mathematics and physics. Many simulators have been written for one and two dimensional automata. The most popular example of a two-dimensional automaton is Conway's Game of Life. It is not difficult to write simulators for such automata, and many examples can be found by a simple web search. The simple rules of a cellular automaton, such as the Game of Life, can produce remarkably complex behaviors.

The three dimensional automaton problem is more complex and computationally costly, making it a good candidate for distributed computation. Many of the issues that arise here are similar to those that arise in parallel scientific computation, such as finite element or finite difference methods. We can parallelize the simulation by assigning parts of the simulation to different processes (*domain decomposition*). Each process contains local data representing its part of the domain. The computation can work primarily with local data, and will use message passing to exchange information at boundaries.

An important decision is how to break up the domain to distribute to the processes. Our automaton is essentially a three-dimensional array, whether or not you choose to represent the data with such an array. This allows us to partition the domain by assigning slices of the domain to each process. For example, if the simulation contains n rows, and there are p processes, each process gets assigned n/p rows.

What your simulator should do

A cellular automaton can be very general, so we will restrict our implementation to something manageable. Your simulator should implement a binary cellular automaton (that is, each cell can be either alive, 1, or dead, 0) on an $n \times n \times n$ grid, with wraparound on boundaries. You should provide options for both a 6-neighbor (all neighboring cells whose locations differ by one in one dimension) and 26-neighbor (all neighboring cells whose locations differ by one in one, two, or three dimensions) rule, with the actual rules specified by two integer values, one which applies to a currently dead cell and one which applies to a currently live cell. The binary digits of the integer indicate what should happen to a cell in the next iteration if it has different numbers of neighbors currently alive. For this type of rule, it is not the location of the neighboring cells which are alive, but just the total number of living neighbors which matters. Initial configuration of the grid should be random, with the probability of a given cell being alive initially given as a parameter. The number of iterations to perform should also be specified. You should include a debug mode which prints out statistics about the simulation (such as the total number of live cells, number of cells which died this iteration, and the number of cells born this iteration, after each iteration), but you must be able to turn this off for your timing runs.

In summary, the run-time options you should provide, which are to be specified by command-line arguments:

- The grid size
- A neighborhood type (6 or 26 neighbors)
- Rule to apply to a dead cell at each iteration
- Rule to apply to a live cell at each iteration
- Probability of a cell being initialized as live
- The number of iterations in the simulation
- A flag to enable or disable debugging output

Ideally, we would also provide some real-time or *a posteriori* visualization of the simulation, but that is left for another time.

Discussion questions

1. Describe your communication patterns in detail. Why did you choose the methods you used? Do you believe it was the best choice? Does your strategy allow for overlap of computation and communication?
2. Would this program be easier to write using pthreads rather than MPI? What would you do differently? When would it be feasible to use a threaded version? Do you think it would be more efficient than the MPI version in those cases?
3. Conduct a scalability study of your software by running the same size problem on 1, 2, 4, 6, 8, 10, and 12 of the FreeBSD systems in the lab. Plot the run times as a function of the number of processes. Do this for three problem sizes, one small (say, 10x10x10), one medium (50x50x50), and one large (make it as large as you can). Next, run problems so that each process has the same number of cells, and vary the number of processors as before. Plot these run times as a function of the number of processes. What do these say about the scalability of your simulator in a distributed memory environment?
4. Now run similar tests on the Origin 2000 at NCSA. Details will follow on how many processes to run and how to run them. What does this say about the scalability of your simulator in a shared-memory environment?
5. How could you optimize the performance of the simulator?

Ground rules

You may work alone or with one partner. Group submissions must be clearly labelled as such. You may discuss the ideas with your classmates, but do not share any code.

Your grade for this project will be based on both the correctness of your simulation code and the quality of the project writeup. Your simulation code will be worth 50 points and the writeup 25 points.

What to submit

At the first due date, your working simulator code along with a Makefile and a file describing how to run the simulation should be submitted electronically. I prefer a *tar* archive of your files, sent as a MIME attachment via e-mail. You may develop your software anywhere, but I will compile and run it for testing on the Williams CS Lab FreeBSD systems.

At the second due date, a formal project writeup, including a more detailed manual for your software, and answers to the discussion questions, should be submitted electronically. I prefer a postscript or PDF format file sent as a MIME attachment via e-mail. Good technical writing style is expected.