Proceedings — NCUR VI (1992)

A PARALLEL APPROACH TO 3D CELLULAR AUTOMATA

Charles R. Calkins
James Robertson
James D. Teresco

Department of Mathematics
Union College
Schenectady, NY 12308

Faculty Advisor: Michael Frame

INTRODUCTION

There are many interesting concepts in Mathematics, but few are as promising as cellular automata. Cellular automata (CA) can both be studied in their own right and used to model other phenomena. They can form self-organizing systems and be studied for their mathematical value, but serve other purposes as well. CA can model fluid flow, magnetic spin systems, diffusion, disease spreading through a community, computation, and can even be used for entertainment, as in Conway's Game of Life.

Cellular automata can be described by a space, and a *rule* that operates on that space. The space consists of cells, each of which can be in one of possibly many states. A rule describes the next state a cell will have. A rule is often dependent on the states of other cells adjacent the cell to be computed (the cell's *nearest neighbors*), but can also depend on other factors, such as a cell's past history of states or on a random number. If a rule depends only on the count of cells in its neighborhood and not their position, it is said to be *outer totalistic*. One application of a rule on a CA space is considered a *generation*. Often, many generations must pass for interesting behavior to be observed.

OUR PROBLEM

CA can be studied in as many dimensions as one has an interest. The computational complexity of a cellular automaton depends on d, the number of dimensions, and is $O(n^d)$. One-dimensional automata are computationally easy. Two-dimensional automata are more useful, but take longer to compute. Software for serial machines is common, and special hardware, such as the

CAM-PC from Automatrix, Inc., have been designed to compute 2D automata in real time. 3D automata are more useful still as they can model the real world more closely, but take even more time to compute. For example, suppose you wished to compute automata for a problem size of 100. For one-dimensional automata, only 100 cells need to be computed. For three-dimensional automata the number increases to 1,000,000 cells. Thus, methods to speed up the computation of higher dimensional cellular automata are important.

OUR SOLUTION

Our project focuses on the computation of three-dimensional binary (two-state) outer totalistic cellular automata. Our rules are generalizations of typical two-dimensional rules. In two dimensions, the Von Neumann neighborhood considers cells that differ in position by one coordinate (4 neighbors), while the Moore neighborhood considers cells that differ in at most two coordinates (8 neighbors). In three dimensions, there are 6 neighbors that differ in at most one coordinate, 18 that differ in at most two, and 26 that differ in at most three.

Our software is written in the language SR, (Synchronizing Resources) a language for distributed processing developed at the University of Arizona in the 1980s. SR has a procedural syntax similar to C or Pascal, with added constructs to aid the programmer with interprocess communication. Our version of SR is an early one, and only works well in a homogeneous UNIX environment. Because of this, we restricted our computations to a network of eight Sun Microsystems sun3s connected by a 10Mhz Ethernet.

SR is designed around the concepts of virtual machines and resources. Virtual machines are processes which can reside on one machine or on many, and are the foundation of an SR program. Resources act as collections of procedures which run on a virtual machine. In our project, we created a single virtual machine on each sun3, and an instance of the resource `Cell` on each virtual machine. The actual computation of the CA can then begin.

We divide our CA space, cubic for simplicity, into collections of planes, or slices in the space. A nested loop then begins. The outer loop determines the number of times we calculate the remote machine loads. A polynomial is computed on each

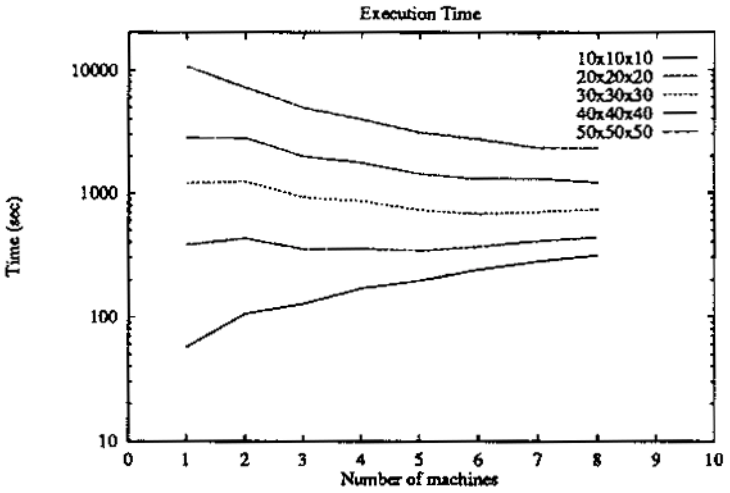
remote machine and the real time it took is recorded. The size of a slice sent out to a remote machine depends on how busy the machine is (related to how long it took to evaluate the polynomial). A thicker slice is sent to less busy machines, a thinner slice to busier ones. Once the slice is determined, it is sent to the remote `Cell` resources.

Now the inner loop begins. The two neighboring planes of each slice are now sent to the remote machine. In order to compute a generation of a slice, the planes directly above and directly below are needed. For example, say a problem of size 100 is to be computed. If three machines are used, a machine load calculation may determine that the three slices should be of size 20, 30 and 50 planes respectively. Thus, 22, 32 and 52 planes are actually sent.

When the computation finishes, only the edges are sent back. For the example above, we know that 20, 30 and 50 planes were computed properly because enough information was provided. The edges of these slices are then sent back to the remote machines for another generation to be computed. For example, the first plane in the set of 30 is really the last plane in the set of 20, so this plane must be sent to both machines for correct computation of the slices. This procedure is then repeated until the inner loop terminates.

A message is then sent to the remote machines to send back their slice. This allows the CA to be recombined and saved. The entire space can be written to disk, to avoid loss of data in case of the failure of any machine in the SR network, which unfortunately terminates the simulation. If the outer loop has not terminated, another machine load calcluation is performed and the process continues. Thus, the number of generations calculated is the product of the number of machine load calculations and the number of iterations in the interim.

RESULTS



In the graph above, the number of machines vs. the problem size is shown for one hundred generations with 26 neighbor rules. One should note that extending the problem from a single machine to multiple machines adds communication overhead. For small problems, the machines spend more time talking to one another than they do computing values of cells, and performance decreases as the number of machines is increased. For large problems, however, the computation outweighs the added communication, and parallelism is beneficial. In our test, parallelism was not profitable until we considered a problem that was greater than 20x20x20. For the 50x50x50, the problem on eight machines could be computed, on the average, about 4.6 times faster than on one machine, 179 minutes for 100 generations as compared to 39 minutes. Not shown on the graph is the 100x100x100 problem. In a typical run, a single sun3 took almost twelve hours, while eight sun3s took just over four.

Ideally, eight machines should give a factor of eight speedup, but dividing the problem as we have done still incurs a significant communications overhead. It should also be mentioned that the machines used for this computation were not dedicated to this project. Processing for our project was done in the background, and the machines were used for normal lab activity. Slightly greater speeds should be able to be reached if the machines were assigned to this problem alone.

We have also written an X windows automaton viewer in C. It will display axis-parallel cross-sections of the CA, as well as a magnified view in three dimensions of a local neighborhood specified by the user. This program provides a convenient way to examine the CA space.

CONCLUSION

Our project demonstrates that parallel speedup for this problem can be achieved by conventional means. The machines we used are common on many college campuses, and even faster machines that would support a larger problem abound. SR is free and available by anonymous FTP, and is easy to install and learn. The machines need no special hardware configuration and do not have to be taken out of active service for the duration of the computation. Thus, almost anyone who wishes to achieve a significant speedup for this problem can do so without difficulty.

Gardner, M. "Mathematical Games" Scientific American October 1970.

Toffoli, T. and Margolus N. Cellular Automata Machines MIT Press, Cambridge, 1987

Carriero, N. and Gelernter, D. How to Write Parallel Programs MIT Press, Cambridge, 1990

Andrews, G. and Olsson, R. "Report on the SR Programming Language" University of Arizona, May 1989