An Interactive Cellular Array Simulator

Doug Seeley
Department of Computer Science
University of British Columbia

Introduction

Much interest was generated by the popular description of John Conway's game of "life".[1] This game incorporates an infinite array or grid of cells, within which simple objects are "born", "survive", and "die". The development of populations of these objects is controlled by simple rules at discrete time steps, rules that depend upon the number of neighbouring cells that are "alive". The great interest centered around the complex and unpredictable behaviour that certain innocuous initial configuration of "alive" objects displayed. In fact, interest was such that a second account appeared[2], one that popularized the related abstract field of cellular automata theory. In so doing, it made concrete the potential of this theory as a framework for many fundamental areas of knowledge.

Cellular automata models were first proposed by von Neumann[3] in order to demonstrate the possibility of constructing self-reproducing machines. Ulam[4] persued very simple systems much like "life" in order to investigate patterns of growth in two and three dimensional arrays. Distributed computation has been investigated one-dimensionally with the "firing-squad synchronization" problem, and multi-dimensionally in models of neural behaviour such as the frog visual system[5] and mammalian hearts[6]. There has also been much interest in retina-like cellular arrays that would perform pattern recognition functions for robot devices[7]. Also self-repairing properties of systems of like computers is an intense area of investigation[8]. The biological-like properties of "life" itself are being studied[2] in order to obtain models of cancerous growth, antibody action, and the evolution of large molecules, and life is only a simple model.

The author believes, in fact, that cellular models offer an exciting alternate world-view than those provided by essentially atomistic models. The latter are at the root of most of our science and mathematics and have warped our perception of the world along atomistic lines.

The cellular approach to modelling can be used in broad areas of information processing. What has held back its use is the following: (a) the behaviour of these models can rarely be expressed in any closed analytic form, (b) there are no formal descriptions of this behaviour (that can be used in general situations), (c) moderately large arrays must be used in order to investigate complex behaviour, (d) computing global systems states at each time step is an arduous task by hand, and very cumbersome if hard copy computer printout is used. Because our intuition is so weak in conceptualizing the parallel behaviour of these models, it is clear that one must substitute analysis by much exploratory empirical investigation. An important example of this point is the way in which Codd[9] found his eight-state five-neighbour self-reproducing automata. He made incremental changes to the behaviour of his cells in a trial and error approach until the self-replication behaviour was observed. It is clear then, that an interactive system that would allow easy model modification and display of model behaviour is required. In this manner, simulation of specific models

are the only means whereby the concepts and principles that will deal with the complex behaviour exhibited by cellular array models, may be discovered.

A graphics facility and general purpose language for the interactive simulation of cellular arrays has been designed and implemented by Brender[10] on an IBM 1800/PDP-7 graphic facility; it has been extensively used at the University of Michigan. However, it was not portable to U.B.C.'s System 360/67 - Adage AGT-10 system. Therefore, an exploratory system called CELL was designed and implemented along somewhat different lines, with a view toward the construction of a more general system. This system also was implemented such that it executed the model simulation within the small core of the Adage computer. This pilot version of CELL[14] has now been transformed into a suitable research tool with much more flexibility in cellular model definition and exploration. What follows is a description of the new CELL command language, sample protocols, and comments upon its use.

The CELL System

Physical Configuration

CELL models are displayed upon the advanced graphics terminal of an Adage AGT-10 computer (8K 30 bit words, vector-oriented) in conjunction with an adjacent IBM 3270 CRT terminal for communication with the operating system (see figure 1). The simulation of a cellular model is controlled by the AGT's function keys (an available light pen has not been found particularly useful for this type of modelling). Cellular model definition is carried out on the 3270 or read in from user files. Also current models and spatial configuration may be saved using this same device.

However, a user need not always use the Adage graphics, he may initially construct and debug a cellular model by carrying out the simulation entirely upon regular system terminals at a considerable saving in computer costs, by using a TEST mode of the program.

Model Definition

In order to construct a cellular model the array must be described, the connectivity or cell neighbourhoods indicated, the initial states of each of the cells specified, and the locally-applied but universal transition rules must be detailed. In addition, various alternative display configurations may be chosen, certain simulation modes selected, and a library of cellular displays saved. Control and interactive commands carry out these specifications.

1. Cellular array description.
2. The neighbourhood of a cell.
3. Initial configuration of the array.
4. Local state transition function.

A description of the array includes the type of regular structure it uses; that is, whether the cells are arranged in rectangular, hexagonal, triagonal, etc., mosaics. Also, its size must be specified. CELL uses only a rectangular grid of cells, although other structures may be embedded in it. The limiting size in CELL is a 600 x 600 grid.

The neighbourhood of a cell is that group of cells that normally hold a fixed geometric relationship to it within the grid. The current states of its neighbourhood determine what the next state of a cell will be after the next time step. Probably, the most common neighbourhood is that of the 4 adjacent cells along the major axes of the grid. CELL allows any combination of 8 neighbouring cells (includes diagonals) for tabular functions, while the subprogram transition function can use arbitrary and dynamic neighbourhoods.

The initial configuration of the array is simply a display of the initial states of each cell in the array. For graphical display convenience, CELL provides an optional mapping function from those symbols used to describe cell states to those symbols (including special characters) that are actually displayed on the CRT screet. There are also a set of commands that allow the user to describe a configuration without having to specify the state of every cell in the array.

The transition function describes the states that each cell enters based upon its own current state and the previous state of its neighbourhood. In CELL this function can be described by a table of neighbourhoods (for convenience we shall call these latter entries, transition functions without ambiguity), or by a user-supplied Fortran subroutine (object module). This function is applied at each time step to each cell of the array in turn, building a new configuration of states on an alternate array. Once this pass has been accomplished, this new configuration can be displayed giving the effect of a parallel global transformation of the array (note that still-to-be-used information about the previous array must not be destroyed in the process). For a nine-neighbourhood several-state model, the number of distinct neighbourhood configurations possible grows very large; hence CELL incorporates special definition features in order to reduce the number of configurations that must be specified by table functions.

CELL Commands

Model Definition

Often a simulation will be desired of a model that has been previously defined. The CELL system is initialized with such a model by using the command:

```
SIMULATE   'model'
  where model is the name of a file where a previously
  defined model was saved
```

The gross characterization of the cellular space that is to be simulated requires specification of grid size (≤600), the boundary character, and whether wraparound is required (for arrays on surfaces). The following commands carry out this function:

```
SIZE        nnn
BOUNDARY    'c'
WRAP        ON/OFF  }
XWRAP       ON/OFF  } default to OFF
YWRAP       ON/OFF  }
```

An initial configuration is required for the running of the model. This is done by filling the space with a background or quiescent state, and then constructing a figure either by describing a string of states along a row or column, or by specifying a contiguous block of states. Note that the numbering of the array cells has the left uppermost cell at location x=1, y=1.

```
FILL        'c'
ROWDATA     x y  'string'
COLDATA     x y  'string'
UPDATE      x y
string           the top left cell of this
string           block is at location x y.
 . . .
$END
```

The transition function that is to be applied locally to every cell in the array can be one of several kinds specified by a "TYPE name" command where name is one of the following.

RANDOM - the order, and orientation of the states in the neighbourhood are unimportant.

DIRECTED - order and orientation are significant.

ORDERED - order of states is significant but not orientation.

FORTRAN - no table of functions will be used, instead the object code of a special FORTRAN subprogram is provided (described in a later example).

The above options for table-driven functions can make possible quite compact descriptions when a default character and a universal character is specified. Below is an example that describes in table form the rules for "life", and illustrates the format of these commands:

```
TYPE RANDOM
BOUNDARY '0'
DEFAULT '0'
NEIGHBOUR AROUND
FUNCTION
'1' 0 0 0 0 0 0 1 1 '1'
'U' 0 0 0 0 0 1 1 1 '1'
$END
```

Inside the first pair of quotes is the current state of the cell under consideration, and inside the last is the new state. In between are the neighbourhood states. If 1 = "alive" and 0 = "quiet", then the 1st transition indicates that an alive state will remain alive if it has exactly 2 'live' neighbours. The 2nd transition indicates that a cell will always be alive if it formerly had exactly three neighbours. All other neighbourhood state configurations lead to the quiet state, since it is the default state. These are exactly the transition rules of Conway's game.

If you number the 8 neighbouring cells of a cell in a rectangular grid, starting from the upper left diagonal one, then the following neighbourhood specifications may be used:

```
NEIGHBOUR  1 2 3
  .  .  .
NEIGHBOUR  1 3 5 7
 etc.
NEIGHBOUR  SIDES     (= 2 4 6 8)
NEIGHBOUR  CORNERS   (= 1 3 5 7)
NEIGHBOUR  AROUND    (= 1 2 3 4 5 6 7 8)
```

An example of a Fortran subprogram that describes the transition function of the cellular space is detailed in figure 2. Note that it will implicitly define the neighbourhood. The calling convention for this subprogram is as follows:

Subroutine Sub (IROW, ICOL, IST, *)

IROW - integer *4 value of the row of the current cell under consideration.

ICOL - integer *4 value of the column of the current cell.

IST - integer *4, whose last byte is the "alphanumeric" state of the current cell upon entry.

- this should also have the new state of the current cell as computed by the subprogram upon return.

- in order to convert to integer form 240 should be subtracted from this value.

- a RETURN 1 will make the new state of the current cell default.

These transition subprograms can interact with the cellular array by acquiring the states of other cells, and the size of the space with function calls, and by altering wraparound conditions:

ICELL (JROW, JCOL, ISW)

This function delivers the state of the cell that is JROW positions and JCOL positions away from the current cell. Function ICELLA performs similarily but with absolute co-ordiantes as arguments. ISW may be omitted since it is only used to alter the wrapping conditions. A subprogram call NSIZE(N) will deliver the current size of the array in N.

Display Commands

There are several commands that will alter the display on the Adage screen and control the simulation from the adajacent system terminal. These are also very useful when the TEST mode of the program is used for displays only on a system terminal. They include:

TITLE	'name of model'	
MAP	'c_1 c_2 ... c_n' d_1 d_2 ... d_n'	
DISPLAY	(ON)/OFF	(OFF is default)
INTER-DISPLAY	ON/OFF	(ON is default)
GØ		
IN		
KEYS		

The MAP command just replaces the "c" symbols that have been used to characterize the cell transitions to "d" symbols for display purposes, the quiescent state being usually mapped into a blank. DISPLAY causes the current space to appear on the screen, while INTER-DISPLAY OFF prevents the space from being displayed during a simulation command of several iterations. In causes n interations of the current space to be executed, while KEYS transfers CELL control to the Adage function keys. Both DISPLAY and GØ (which alters the current space) can take the following arguments:

INITIAL - the space when the user last defined a new space.

PREVIOUS - the space from the iteration before the current one.

BUFFER n - the nth space saved in the buffer library.

Since the arrays that can be simulated can be potentially 600 x 600 it is essential to have a windowing facility for display purposes. Hence a WINDOW command allows the user to indicate a window of some size (square) at a specific location of the upper left corner. e.g.

```
WINDOW SIZE 20 AT 9 12
```

or

```
WINDOW AT 9 12 SIZE 20
```

Display Interaction

Interaction with an ongoing simulation may be performed using function keys. The only situation that would seem to favour the use of a light-pen is when neighbourhoods are encountered that have not been specified in the transition function definition. Such scenarios occur when one is still developing the required transition function and is not depending upon the

default character (e.g. '?') for any of the transitions. When a iteration encounters a strange neighbourhood the '?' appears; at this point it would be useful to interrogate the array at such locations in order to determine the neighbourhood from the previous iteration that produced it. However, this can still be done using the current CELL system by using the Display Previous function key.

There is a 4 x 4 function key grid available, the first 2 columns of which are for simulation iteration counts of 1, 5, 10, 15, 20, 25, 30, and ∞. The upper rightmost dial can be used as an interrupt to any of the lengthy simulations (by turning it CW from the extreme CCW position). Other keys act as switches for altering the inter-display mode (1, 3), and wraparound in the horizontal and vertical directions, (2, 3; 2, 4). One key (1, 4) will cause the current state of the array to be stored in a buffer library, while others cause a new display of the array from the previous iteration (3, 3), the array last sent to the buffer (3, 4), and the initial array (4, 3). Finally, the remaining key returns control of the simulation to the system terminal.

Model Manipulation

Facilities in the command language are provided for the temporary or permanent saving of array states, transition functions, and entire models, and their subsequent recovery. The command BUFFER, optionally followed by a title (<15) in single quotes, causes the current array to be stored in the buffer library along with its title and iteration count.

A LIST command is available that allows various output to be copied to user-owned system files; it may take the following operands:

BUFLIB - a list of the titles of the arrays currently in the buffer.

BUFFER n - the nth array from the buffer library.

CURRENT - current array state.

FUNCTIONS - current transition functions.

INITIAL - initial array state.

PREVIOUS - previous array state.

The user file is specified by appending ON 'filename-or-device' to the command; the default is the system terminal. Arrays may be brought back into the simulation by a DATA FROM 'filename-or-device' command.

Entire models including all of the display and array parameters may be stored permanently by using the SAVE command. Available operands are CURRENT, BUFFER n, INITIAL (or blank), and PREVIOUS followed by "IN 'file-or-device'." Once a model has been saved in this manner, a simulation of it may be re-started by a SIMULATE 'file-name' command.

SYSTEM COMMUNICATION

Provision has been made for incorporating the operating system into a simulation run. An EDIT command, for instance, will make the MTS file editor available for altering tabular transition functions. Control can be passed entirely to the operating system by the MTS command, and as long as no other programs are run, control may be returned to CELL by a RES command. An ECHO (ON)/OFF switch controls the echoing of commands when read in from a file. STOP terminates a CELL run, returning control to MTS.

Cellular Simulation Experience

The CELL system has been tested by using many of the cellular models that are available from the literature. One example that has been particularily useful for demonstration and testing purposes are the Fredkin-Winograd[11] self-replicating spaces. If the states of the system are mapped into the integers 0, 1, 2, ... p-1 (p is prime) then applying a transition rule that takes the sum of the neighbour states modulo p to some initial configuration produces copies of this configuration in the direction of the neighbourhoods some p iterations later. The transition rule for this space is illustrated by figure 2.

Originally, interest in this area was generated by von Neuman's exploration of self-reproducing automata[3], and Ulam's investigations of patterns of growth[4,12]. A recent version of a self-reproducing automaton discovered by Banks[13] has been run on the CELL system along with several of Ulam's spaces. Of course, Conway's game of "life" has been done.

Another classic example of distributed computation is the Firing-Squad Synchronization problem. A solution to this has been successfully implemented in CELL. Malzak[15] has seen this problem as a pattern recognition problem for one-dimensional arrays. He has suggested how complexity measures for patterns, pattern distinguishability, and "local" and "global" properties may be embedded in cellular arrays and arbitrary cellular spaces.

The author's own interest in this area of modelling is in the problem of aggregation and its relationship to the validation of simulations. Closely allied to this is the viability of cellular spaces as an alternative world view, when seen as levels in a heterarchy that is open-ended both at the micro and macro levels.[16] The viability of cellular space models is also being examined in the physics of fields and particles, and highway traffic flow.[17] CELL appears to be a reasonable system with which to explore these possibilities.

Conclusions

The CELL system has demonstrated that a cellular array simulator with reasonable graphic interaction, broad model definition, and array saving capabilities can be implemented within a simple command language (implemented in IBM System 360 Assembler Language). The only graphic features that might improve its performance would be interactive windowing and light-pen interrogation of cell neighbourhoods. A general purpose language and system such as Brender's Cellular Space Language (CSL)[10] need not be attempted.

It appears necessary that such a system be made available in order that people become aware of the modelling potential of cellular arrays, by observing concrete examples as they evolve. Also, it is clear that highly interactive systems are required in order to fully explore and acquire intuition for the complex behaviour that result from these models. Particularly promising is the prospect of developing concepts to deal with complex biological systems, parallel computation, distributed intelligence, and systems exhibiting self-repair.

Most important however, is the making cellular arrays more readily accessible as an alternative world-view for modelling. With some notable exceptions, such as field theories and relaxation methods, practically all of our mathematical and logical models, and especially those implicit in our natural languages force sequential and linear causality, and fragmented universes into our view of the world. In an age where the complexity of artificial, social, and natural systems beseige us from all sides, the holistic and all-at-once parallel characteristics of cellular models offer fresh hope.

Acknowledgments

The author wishes to express his appreciation to the National Research Council of Canada for its financial support and to John Lau who implemented the CELL program.

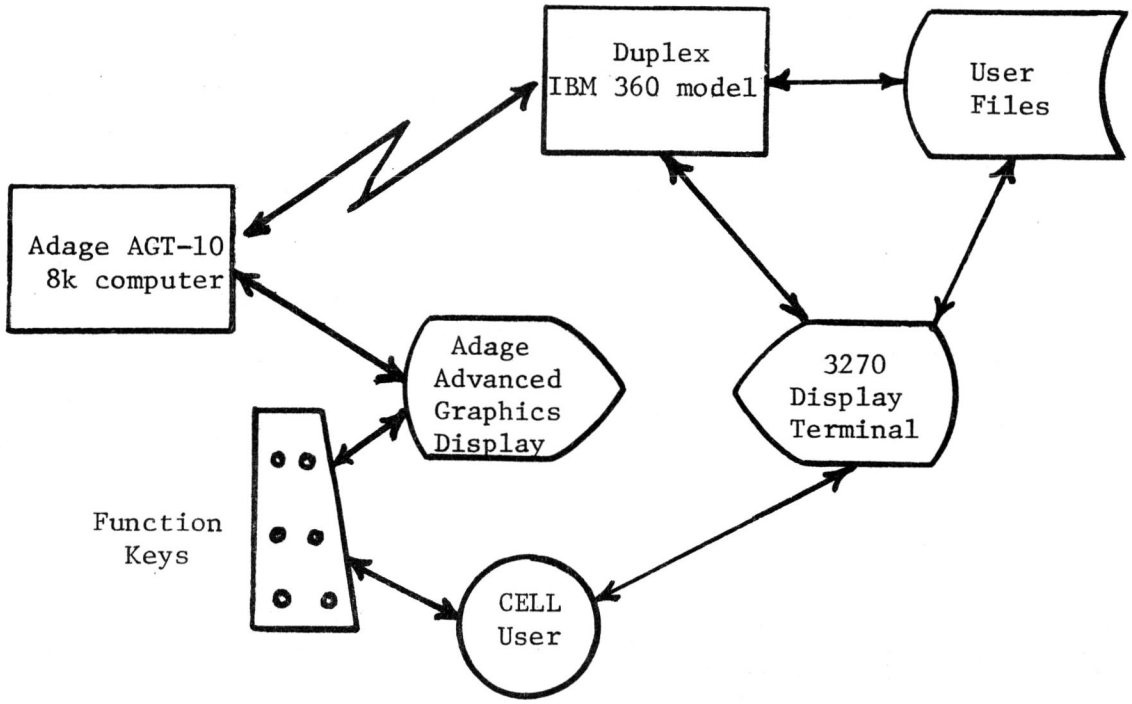

Figure 1. The CELL system; Implemented Physical Configuration

```
      SUBROUTINE S(IR,IC,IST,*)

C
C  ****    WINOGRAD, MODULO-5 SELF-REPLICATING SPACE
C
            IST = IST-240
            MODSUM = 0
            DO 1 J=1,4
            GO TO (2,3,4,5),J
        2   IS = ICELL(0,IP)
            GO TO 1
        3   IS = ICELL(0,-IP)
        4   IS = ICELL(IP,0); GO TO 1
        5   IS = ICELL(-IP,0); GO TO 1
        1   MODSUM = MODSUM+(IS-240)
            IS = MOD(MODSUM,5)
            IST = IS+240
            RETURN
            END
```

Figure 2. Example of a Subprogram Transition Function

References

1. Gardner, Martin, Mathematical Games, in Scientific American, October 1970.

2. Gardner, Martin, Mathematical Games, in Scientific American, February 1971.

3. Burks, W., Von Neumann's Self-Reproducing Automata, in Essays on Cellular Automata, University of Illinois Press, editor A.W. Burks.

4. Ulam S.M., On Some Mathematical Problems Connected with Patterns of Growth of Figures, in Essays on Cellular Automata, University of Illinois Press,

5. Didday, R.L., The Simulation and Modelling of Distributed Information Processing in the Frog Visual System, Technical Report 6112-1, Information Systems Laboratory, Stanford University, August 1970.

6. Flannigan, L.K., A cellular Model of Electrical Conduction in the Mammalian Atrioventicular Node, Ph.D. Thesis, University of Michigan, 1965.

7. Minsky, M. and Papert, S., Perceptrons, the M.I.T. Press, 1969.

8. Avizienis, A., Theory and Design of Fault-Tolerant (Ultra Reliable) Digital Computers: Protective Redundancy, Diagnosis, Self-Repair Techniques, course notes UCLA short course, March 1972.

9. Codd, E.F., Cellular Automata, Academic Press, 1968.

10. Brender, R.F., A Programming System for the Simulation of Cellular Spaces, Technical Report, Computer and Communications Sciences Dept., University of Michigan, January 1970.

11. Winograd, Terry, A Simple Algorithm for Self-Replication, M.I.T. Project MAC Artificial Intelligence Memo no. 197, May 1970.

12. Ulam, S.W. and Schrandt, R.G., On Recursively Defined Geometrical Objects and Patterns of Growth, in Essays on Cellular Automata, University of Illinois Press.

13. Banks, E.R., Information Processing and Transmission in Cellular Automata, M.I.T. Project MAC Report TR-81, January 1971.

14. Seeley, D.A.R. and deKleer, J., CELL, An Interactive Cellular Array System, Proceedings of the Canadian Computer Conference, Session 72.

15. Melzak, Z.A., Companion to Concrete Mathematics, Wiley 1973.

16. Seeley, D.A.R., Epistemics, Holons, and Synchronicity, Progress in Cybernetics and Systems Theory, Gordon & Breach, 1973.

17. Zeigler, B.P., editor, Cellular Space Models for Particle Physics, Biological Development, and Highway Traffic Flow: Some Initial Explorations. Technical Report, August 1972, Dept. of Computer and Communication Sciences, University of Michigan.