An International Journal
computers & mathematics with applications

Computers and Mathematics with Applications 37 (1999) 115-127

Text Compression Using Two-Dimensional Cellular Automata

A. R. KHAN
Department of Electronics and Computer Sciences
University of Kashmir, Srinagar-190006, India

P. P. CHOUDHURY*, K. DIHIDAR AND R. VERMA
Computer Science Unit, Indian Statistical Institute
203, B. T. Road, Calcutta-700035, India
pabitra@isical.ac.in.

(Received July 1997; accepted August 1997)

Abstract—This paper presents an elegant mathematical model using simple matrix algebra for characterising the behaviour of two-dimensional nearest neighbourhood linear cellular automata with periodic boundary conditions. Based on this mathematical model, the VLSI architecture of a Cellular Automata Machine (CAM) has been proposed for text compression. Experimental results of comparisons with adaptive Huffman coding scheme also presented.

Keywords—Cellular automata, Encoding, Decoding, Text compression, Kernel.

1. INTRODUCTION

As the semiconductor technology is moving towards the submicron era, the system designers try to embed complex functions from software domain to hardware blocks on the silicon floor. At the same time, for keeping the design complexity within a feasible limit, the designers are forced to look for simple, regular, modular, cascadable, and reusable building blocks for implementing various complex functions. The homogeneous structure of Cellular Automata (CA) is a right candidate to fulfill all the above objectives. Moreover, the demand for parallel processing architecture has gained importance with the ever increasing need for faster computing. To this end, we are motivated to use the two-dimensional cellular automata model to arrive at the easily implementable parallel processing architecture in VLSI. This parallel architecture built around the CA machine suits ideally for a variety of applications which has been demonstrated in one of our earlier papers [1].

The study of cellular automata (CA) dates back to Von Neumann in the early 1950s. Van Newmann [2] framed CA as a cellular space capable of self-reproduction. Since then, many researchers have taken interest in study of CA for modelling the behaviour of complex system. Wolfram *et al.* [3] studied one-dimensional CA with the help of polynomial algebra. Pries *et al.* [4] studied one-dimensional CA exhibiting group properties based on a similar kind of polynomial algebra.

Later, Das *et al.* [5] extended the characterization of one-dimensional CA with the help of matrix algebra. In recent years, many applications of one-dimensional CA have been reported [6–9]. On the other hand, two-dimensional CA is not yet a well-studied area. Packard *et al.* [10] reported some empirical studies on two-dimensional cellular automata depending on five neighbourhood CA. Chowdhury *et al.* [11] extended the theory of 1DCA built around matrix algebra for characterizing 2DCA. However, emphasis was laid on special class of additive 2DCA, known as Restricted Vertical Neighbourhood (RVN) CA. In this class, the vertical dependency of a site is restricted to either the sites on its top or bottom, but not both. This paper mainly deals with the characterization of 2D periodic boundary nine neighbourhood linear CA. A general framework had been proposed for the study of the state transition behaviour of this class of 2DCA. In this paper, we developed the analytical tool to study all the nearest neighbourhood. 2DCA linear transformation with periodic boundary conditions. One application of periodic boundary 2DCA is also reported.

A CA machine (CAM) has been proposed [1] around the parallel architecture of 2DCA. Such a CAM can be economically built with the available VLSI technology. A wide variety of applications can be developed around the parallel architecture of CAM. Such CAM can serve as a simulation engine to study a wide variety of hybrid CA configurations. Some of these applications have also been reported [1]. The next section briefly describe both the 1D and 2DCA preliminaries. Section 3 highlights a few sample results on the characterisation of uniform periodic 2DCA transformations. Section 4 reports one simple application which deals with text compression. A VLSI architecture of cellular automata machine (CAM) is reported in Section 5 to implement the above application. Section 6 concludes the paper.

2. BASIC CONCEPTS

Prior to introducing the 2DCA framework, a brief introduction on 1DCA is reported below. The 1DCA structure can be viewed as a discrete lattice of sites or cells, where each cell can assume either the value 0 or 1. The next state of a cell is assumed to depend on itself and on its two neighbours (for two-neighbourhood dependency). The cells evolve in discrete time steps according to some deterministic rule that depends only on local neighbourhood. Mathematically, the next state transition of the i^{th} cell can be represented as a function of the present states of the i^{th}, $(i+1)^{\text{th}}$, and $(i-1)^{\text{th}}$ cells:

$$q_i(t+1) = f(q_i(t), q_{i+1}(t), q_{i-1}(t)),$$

where f is known as the rule of a CA.

If the next state function of a cell is expressed in the form of a truth table, then the decimal equivalent of the output is conventionally called the rule number for the cell.

Neighbourhood state:	111	110	101	100	011	010	001	000
Next state:	0	1	0	1	1	0	1	0
Next state:	0	1	1	1	1	0	0	0

The top row gives all the eight possible states of the three neighbouring cells at the time instant 't', while the second and third rows give the corresponding states of the i^{th} cell at time instant $(t+1)$ for two illustrative CA rules. A few definitions are next introduced.

DEFINITION 1. *If the same rule is applied to all the cells in a CA, then the CA is called a uniform or regular CA.*

DEFINITION 2. *If different rules are applied to different cells in a CA, then the CA is called a hybrid CA.*

DEFINITION 3. *If in a CA, the neighbourhood dependence is on EX-OR or EX-NOR only, then the CA is called an additive CA. Specifically, a linear CA employs XOR rules only.*

DEFINITION 4. *A periodic boundary CA is the one in which the extreme cells are adjacent to each other.*

DEFINITION 5. *A null boundary CA is the one in which the extreme cells are connected to logic 0-state.*

DEFINITION 6. *A CA whose transformation is invertible is called a group CA; otherwise, it is a nongroup CA. For a group CA, the dimension of kernel is 0 (that is, the transformation is a full rank one); for a nongroup CA, the dimension of the kernel is nonzero.*

2.1. Mathematical Model for 2DCA

In a two-dimensional nearest neighbourhood linear CA, the next state of a particular cell of the 2DCA is affected by the current state of itself and eight cells in the nearest neighbourhood. Different dependencies are taken into account by means of various CA rules. We take into consideration only the linear rules, i.e., the rules which can be realized by EX-OR operation only. A specific rule convention is envisaged as in the following.

64	128	256
32	1	2
16	8	4

Here, the central box represents the current cell (that is, the cell being considered) and all other boxes represent the eight nearest neighbours of that cell. The number within each box represents the rule number associated with that particular neighbour of the current cell. That is, if the central cell gas got dependency on itself, it is referred to as rule 1, if it depends only on its top neighbour, it is rule 128, and so on. In case the cell has dependency on two or more neighbouring cells, the rule number will be the arithmetic sum of the numbers of the relevant cells. For example, the 2DCA rule $171N(128+32+8+2+1)$ refers to the five neighbourhood dependency of the (central) cell (top, left, bottom, right, and self) under null boundary condition, whereas 171P refers to the same neighbourhood dependency under periodic boundary condition.

The dependency structure can be realized with the help of an elegant mathematical model where we use two fundamental matrices to obtain row and column dependencies of the cells. Let our binary information matrix be X_t—the current state of a 2DCA configured with a specific rule. The next state of any cell will be obtained by EX-OR operation of the states of its relevant neighbours associated with the rule. The transformation associated with different rules can be made effective with the following two fundamental matrices:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = T_1 \quad \text{and} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = T_2.$$

2.2. Characterization of Periodic Boundary 2DCA (PBCA)

The following lemma specifies the value of the next state of a 2D PBCA referred to as

$$X_t + 1,$$

given that its current state is

$$X_t.$$

LEMMA 1. *The next state of all the primary rules with periodic boundary condition can be represented as in [1], except that*

$$T_1 \quad \text{and} \quad T_2$$

are to be replaced by

$$T_{1c} \quad \text{and} \quad T_{2c},$$

respectively, where

$$T_{1c} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \qquad T_{2c} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

In T_{1c}, super diagonal contains all 1's with lower left corner 1 and T_{2c} contains subdiagonal all 1's with upper left corner 1.

Proofs are omitted due to shortage of space.

For our convenience of analysis, we will convert each of the rules (both primary and secondary) into some one-dimensional transformation. We want to look at the transformation T such that T operating on X information matrix of dimension $m \times n$ giving us the new configuration $[X']m \times n$, in the following manner:

$$T(X)m \times n = [T]mn \times mn \begin{bmatrix} X1 \\ X2 \\ \vdots \\ Xm \end{bmatrix}_{mn\times 1} = \begin{bmatrix} X1' \\ X2' \\ \vdots \\ Xm' \end{bmatrix}_{mn\times 1},$$

where $X1, X2, \dots, Xm$ are the rows of X and $X1', X2', \dots, Xm'$ are rows of the new transformed matrix X'.

LEMMA 2. *The matrix for any periodic boundary CA rule (PR) can be represented as*

$$T_{\mathrm{PR}} = \begin{bmatrix} D & U & 0 & 0 & \dots & 0 & 0 & U_c \\ L & D & U & 0 & \dots & 0 & 0 & 0 \\ 0 & L & D & U & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & L & D & U \\ L_c & 0 & 0 & 0 & \dots & 0 & L & D \end{bmatrix},$$

where D, L, U, L_c, and U_c are one of the following $n \times n$ matrices, $[0]$, $[I]$, $[T_{1c}]$, $[T_{2c}]$, $[I + T_{1c}]$, $[I + T_{2c}]$, $[S_c]$, and $[I + S_c]$.

THEOREM 1. *$(T_{1c})_{n\times n}$ and $(T_{2c})_{n\times n}$ are full rank.*

THEOREM 2. *All PBCA primary rules are group rules.*

3. CHARACTERIZATION OF 2D CELLULAR AUTOMATA

Next, we are going to highlight some of the most interesting results for some specific linear transformations *viz.* 170 (periodic boundary condition), i.e., 170P. For analyzing rule 170P, we first convert it into a uniformly partitioned one-dimensional map matrix:

$$\begin{bmatrix} S_c & I & 0 & \dots & I \\ I & S_c & I & \dots & 0 \\ 0 & I & S_c & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ I & 0 & 0 & \dots & S_c \end{bmatrix}.$$

Next, applying matrix algebraic formulations, we can prove the following results.

PROPOSITION 1. *If P be a permutation matrix of order n, $P^{-1} = P^T$ and $(P^n = (P^T)^n) = I$, then $S_c = P + pT$.*

PROPOSITION 2. Rank $((S_c)_{n\times n}) = n-2$, *if* n *is even,* $= n-1$, *if* n *is odd.*

PROPOSITION 3. $p_k(S_c) = (P^k + (P^T)^k) + (P^{k-2} - (P^T)^{k-2}) + (P^{k-4} + (P^T)^{k-4}) + à \cdots + (P^2 + (P^T)^2) + I$, *if* k *is even.* $p_k(S_c) = (P^k + (P^T)^k) + (P^{k-2} - (P^T)^{k-2}) + (P^{k-4} + (P^T)^{k-4}) + à \cdots + (P^3 + (P^T)^3) + (P + P^T)$, *if* k *is odd.*

LEMMA 3. *For all even* n, $p_n(S_c) = I$.

LEMMA 4. *For all even* n, $p_{n+1}(S_c) = S_c$.

LEMMA 5. *For all even* n, $p_{n-1}(S_c) = 0$.

LEMMA 6. *For all odd* n, $p_n(S_c) = p_{n-2}(S_c) = I + p_{n-1}(S_c)$.

LEMMA 7. *For all odd* n, $p_{n+1}(S_c) = S_c + p_{n-1}(S_c) = S_c + I + p_{n-2}(S_c)$.

LEMMA 8. *For all even* n *and* $r = 1,2,3,\ldots$, $p_{rn}(S_c) = I$.

LEMMA 9. *For all even* n *and* $r = 1,2,3,\ldots$, $p_{rn-1}(S_c) = 0$.

LEMMA 10. *For all even* n *and* $r = 1,2,3,\ldots$, $p_{rn+1}(S_c) = S_c$.

LEMMA 11. *For all odd* n *and odd* r, $p_{rn}(S_c) = p_{n-2}(S_c) = I + p_{n-1}(S_c)$.

LEMMA 12. *For all odd* n *and odd* r, $p_{rn+1}(S_c) = S_c + p_{n-2}(S_c) + I$.

LEMMA 13. *For all odd* n *and even* r, $p_{rn}(S_c) = I$.

As a consequence of the above results, we can prove the following result, which was first arrived at by Sutner [12] in connection with σ-game.

THEOREM 3. *The dimension of the kernel of* $T170P$ *is* $gcd(m,n)$ *when* m *or* n *or both are even and* $2gcd(m,n) - 1$ *when both* m *and* n *are odd.*

4. ONE APPLICATION OF 2DCA

4.1. Text Compression

In the previous section, we have seen that the dimension of kernel of $T170$P is $2gcd(m,n)$ when m or n or both are even, and $2gcd(m,n) - 1$ when both m and n are odd. In particular, we have the following tables.

$m = 2$.

n	2	3	4	5	6	7	8	9	10	11
Dimension of Kernel	4	2	4	2	4	2	4	2	4	4

$m = 3$.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Dimension of Kernel	2	5	2	1	6	1	2	5	2	1	6	1	2	5

We were looking for some suitable application where dimension of kernel cyclically comes back as n increases. However, in the absence of any such application, we noticed the following things. For $m = 2$ and $n = 4$, dimension of kernel $= 4$, that is, in the state transition diagram, each reachable state is having $2^4 = 16$ predecessor states.

Referring to Figure 1, we see that all the upper case, lower case appear in the second quadrant. Each quadrant has four characters. Only upper case letters appear in the second quadrant of Figure 1a–d and Figure 1i–l, similarly lower case letters appears in the second quadrant of Figure 1e–h and Figure 1m–p.

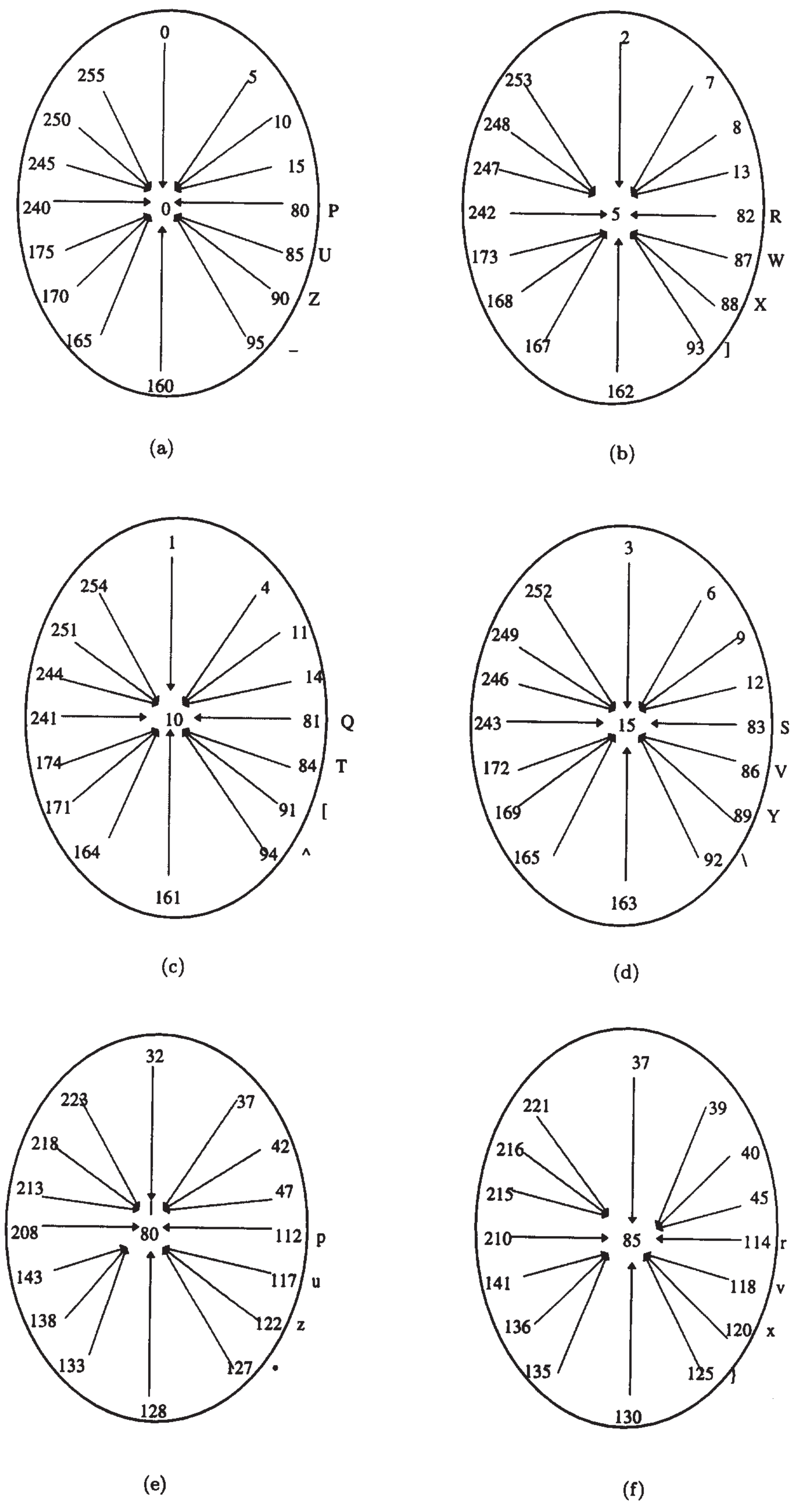
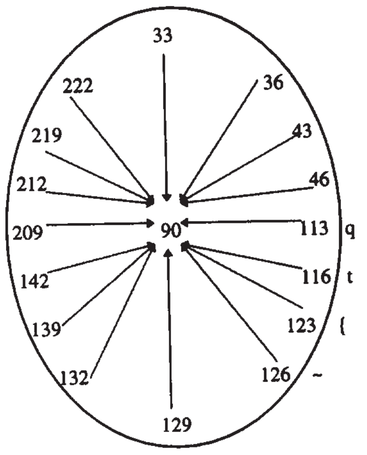
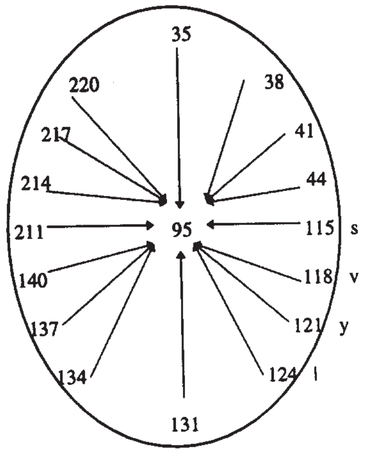


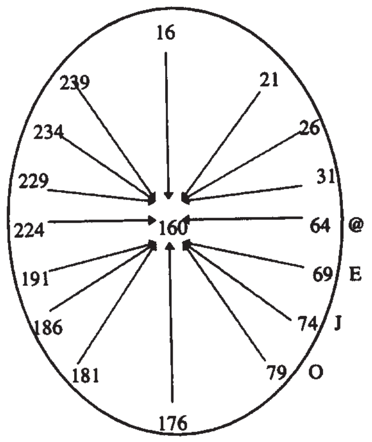
Figure 1. State transition diagram, for the rule 170P on 2 × 4 2DCA.



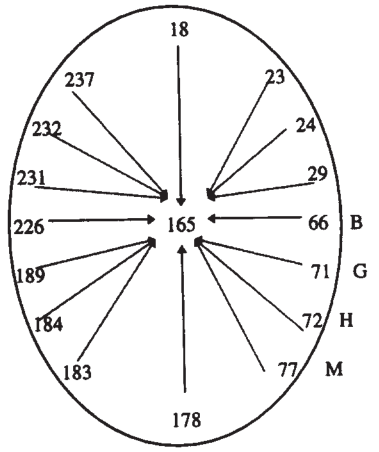
(g)



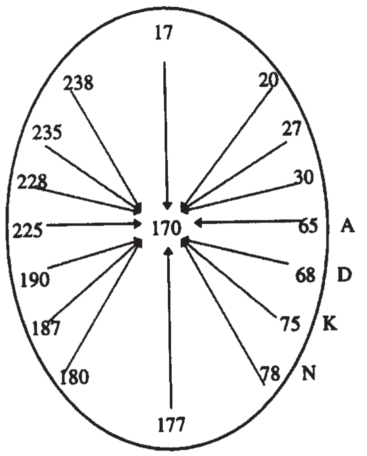
(h)



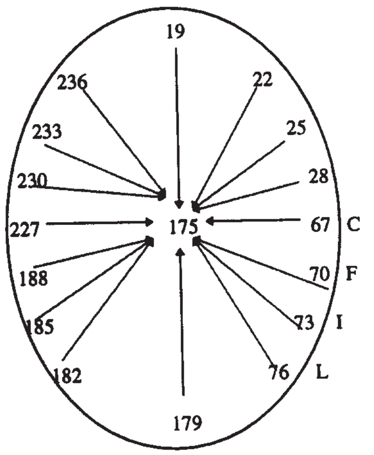
(i)



(j)



(k)



(l)

Figure 1. (cont.)

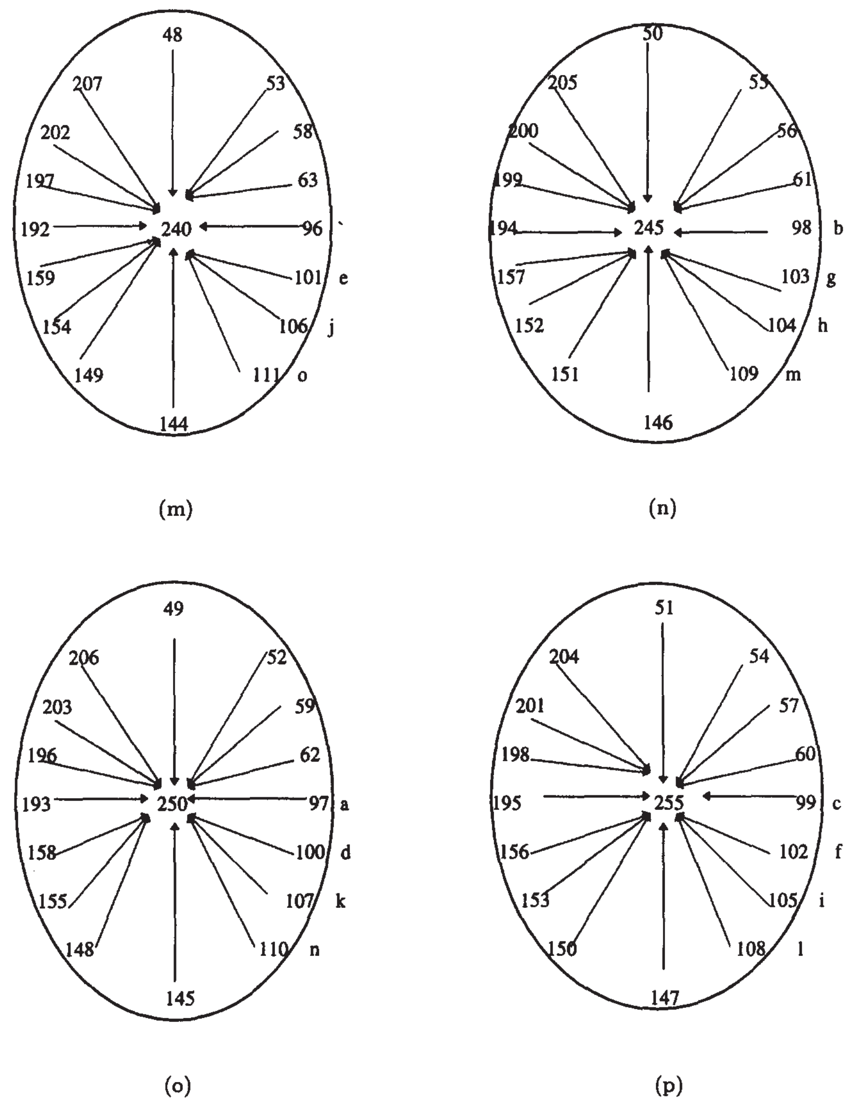


Figure 1. (cont.)

5. ARCHITECTURE OF 2DCA BASED CAM FOR TEXT COMPRESSION

In Section 2, we have proposed a local neighbourhood 2DCA where the next state of each cell depends on the current state of its neighbours *viz.* four orthogonal neighbours, four diagonal neighbours, and itself. In order to design the most general structure, we should be able to configure a cell with a rule out of the 512 available rules. That is, each 2DCA cell is connected through nine switches to its nine nearest neighbours. In order to apply a particular rule, we have to apply 1 or 0 to the corresponding switch control and thereby closing or opening the switches. Thus, a nine bit word is required to control the nine switches corresponding to a single cell. Another bit is required to configure the cell in XOR or XNOR mode. Hence, we have a separate control plane where entry (i, j) stores a ten bit control word corresponding to the particular rule employed to configure the cell (i, j) of the 2DCA. The CAM architecture is shown in Figure 2. In effect, by providing a generalized 2DCA structure, we have incorporated programmability. Hence, we can introduce the concept of a rule program which is a tuple containing the rules to be applied to the cells, the initial seed, and the number of cycles the 2DCA has to be iterated.

The motivation for the parallel architecture of the CAM is derived from following considerations. The reader may recall that computation related to any $m \times n$ 2DCA requires manipulation of $mn \times mn$ binary matrix which is computationally intensive, if not infeasible, when m and n cross 10—such a situation arise in a variety of applications. Therefore, we require an all-purpose hardware simulation engine which is provided by our architecture.

It may be noted that such a machine can achieve a high simulation performance with an appreciably low hardware overhead compared to the universal synthesizer proposed by Toffoli [13].

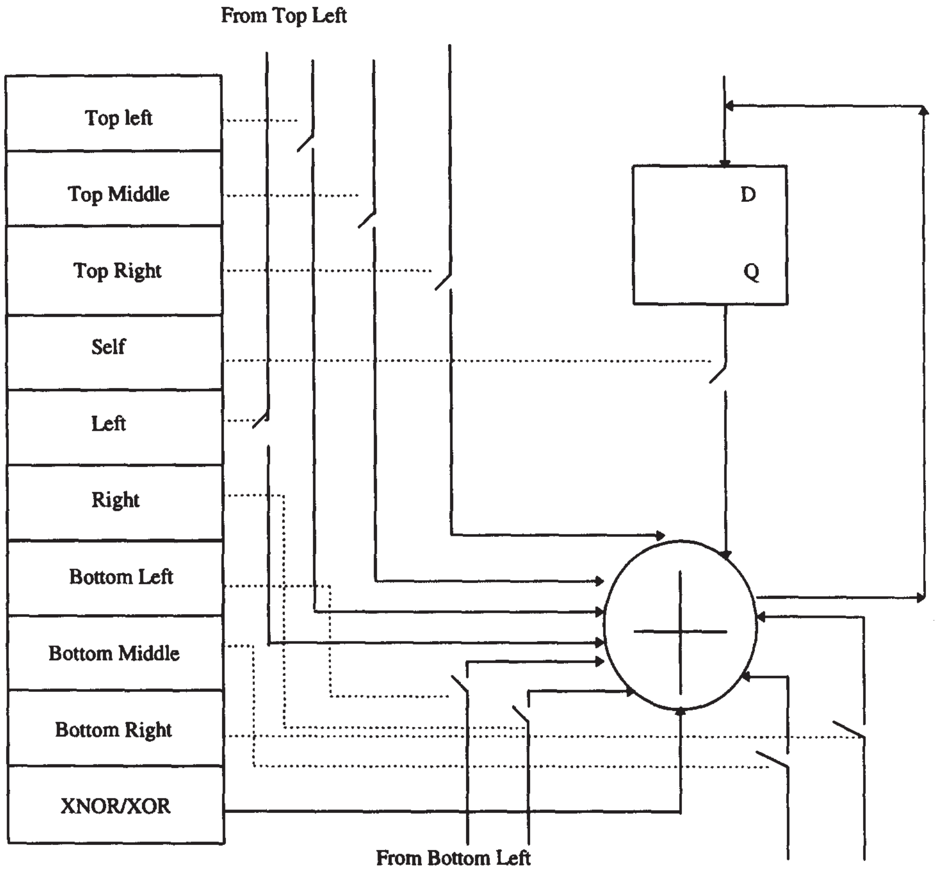


Figure 2. Architecture of the CAM.

5.1. Architectural Details of the CAM

This section provides the architectural details of a CAM corresponding to an $m \times n$ 2DCA. As shown in Figure 2, there are three major blocks.

- An $m \times n$ matrix of CA cells—the details of a representative cell is shown in Figure 2.
- The **Control Memory** is required to configure the 2DCA cells. Since any nearest neighbourhood 2DCA cell has dependencies on nine of its nearest neighbours, a typical control word should contain at least nine bits, moreover, another bit is required to configure the cell in XOR or XNOR mode. Thus, for each cell we require a ten bit control word. Hence, the size of the control memory is $m \times n \times 10$.
- A **Seed Memory** is also kept to store b initial seeds of the 2DCA. For that purpose, the size of the seed memory will be $m \times n \times b$.

If we restrict ourselves to uniform 2DCA, then the size of the control plane reduces drastically to only $9 + (m \times n)$. The term $m \times n$ is kept to take care of the XOR/XNOR for each cell. Thus, a drastic cost reduction is possible with a uniform 2DCA based CAM architecture. However, we will be losing the programmability which is one of the essential requirements for carrying out exhaustive simulations with the CAM. So, we can adopt a hybrid approach where each of the m rows will be configured with the same rule. This reduces the size of the control memory to $m \times 9 + (m \times n)$ while preserving the programmability to the same extent. Next, we report applications of CAM.

5.2. Applications of CAM in Text Compression

In the text compression, we are utilizing rule 170P of 2DCA of dimension 2×4, i.e., to every cell we are applying rule 170P.

To implement our coding and decoding scheme, we will be having 16 blocks of memory locations each for the state from which we can reach to 0 in one step. There will be one main memory block having 16 bytes and all other 16 blocks will be having four bytes. Memory blocks (other than main) contains the values of the second quadrant of their respective state (Figure 3). Main memory block will be having values of the states which can reach to state 0 in one step, starting from first quadrant. The starting address of these blocks other than main can be obtained with the help of some hashing technique. The starting address of main memory block is fixed. As most texts are having lower case letters in abundance than that of upper case and other special characters, so we give stress on efficient encoding of only lower case letters. Generally an upper case letter comes just after the fullstop (.) or full stop and space, so we encode this also with same efficiency. Also the encoding of space, full stop, and comma are done with equal efficiency. Upper case letters other than those starting just after a full stop or full stop and space are encoded with the help of ten-bits and all other special characters are encoded in 13-bit form. But as the probability of existence of these inefficient codeable symbols in a text is extremely less, we will get a good compression ratio.

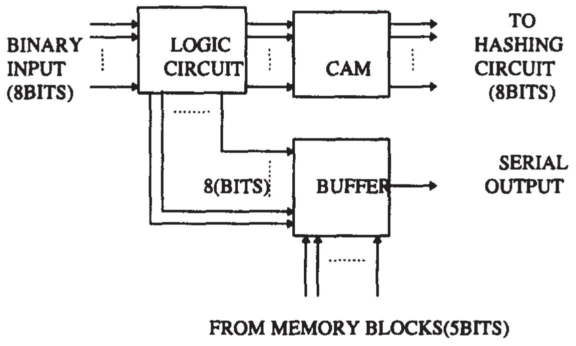


Figure 3. Architecture of CAM encoder.

ENCODING. Our encoder will be having a simple logic to separate codeable and noncodeable symbols. The codeable symbols will be given to CAM which apply rule 170P on the binary value of the symbol to produce a number. We access to the memory block belonging to this number and search for input symbol. The two bit index of it in that block is stored in the buffer. Then we search this number in the main memory block. We get the four-bit index of it in that block and store the 1-MSB and 2-LSBs in the buffer. This is how we encode a symbol (Figure 3).

Our logic circuit will allow only lower case letters and upper case letters to reach to the CAM. Space will be converted to '□', full stop will be converted to '@' or ' ' ', CR will be converted to ' } ' and comma will be converted to '[' before passing it to CAM. As in the memory blocks, we have put the decimal value of full stop in place of '@' and ' ' ', the comma in place of ' [', CR in place of ' } ' and space in place of '□', so we will be able to recover the original values in the decoding phase. Whenever an upper case letter is coming which is not just after a full stop or space, we put a five-bit code 01011(˜) in the buffer and then store the code for the symbol in buffer. Whenever a noncodeable character comes, we put a five-bit code 01111(!) before its actual eight-bit representation in the buffer.

EXAMPLE. Now let us try to compress a word with the help of our technique. Let the word be '.As'. On loading '.' to the encoder, the logic circuit will convert it into '@' or ''' (say '@') and supply it to CAM. CAM will apply rule 170P to give 160. Now we will search the decimal value of input symbol ('.') in the memory block belonging to 160 and find its index which is 00. Then we search 160 in the main memory block and find out its index, which is 1000. Now we omit the third bit from right, the resultant bits are 100, so our complete code will be 10000. As '*A*' is coming after '.', so it is encoded by the same technique as 11000. Next for '*s*' we get 01100.

DECODING. To decode the compressed text, we start taking the five-bit chunks from the stream of bits. If it is 01111(!), then we just output the next eight-bits from the stream. If it is 01011(~), then we take the next chunk of five-bits and decode it as an upper case letter. If it is a full stop, then decode it and output it, now take the next chunk and find whether it is a space, if yes then decode it and check for another space, and so on. In doing so, as we get a nonspace five-bit chunk, decode it as an upper case letter. Thus, by above scheme we will be able to distinguish between lower case and upper case. If code is for lower case and its MSB is 0, then add binary 100 to 3-MSBs, else if it starts from 1, then add 1100, and fetch the value of number at this indexed memory location in the main memory block. Now go to the memory block corresponding to this fetched number and fetch the number indexed by the 2-LSBs of the code. Output the binary equivalent of it. In case of upper case letters, if it starts from 0, then add 0000 to 3-MSB's and if it starts from 1, then add 1000 and do the same procedure as for the lower case letters.

Main	0	5	10	15	80	85	90	95	160	165	170	175	240	245	250	255
0	80	85	90	95												
5	82	87	88	93												
10	81	84	91	44												
15	83	86	89	92												
80	112	117	122	32												
85	114	119	120	125												
90	113	116	123	126												
160	46	69	74	79												
165	66	71	72	77												
170	65	68	75	78												
175	67	70	73	76												
240	46	101	106	111												
245	98	103	104	109												
250	97	100	107	110												
255	99	102	105	108												

Figure 4. Different memory blocks.

5.3. Experimental Data

The following text file was taken for experiment.

File:

This document provides supplementary information on configuring
Btrieve for use with Visual Basic. Before you import,
export, or attach Btrieve tables with Visual Basic, please

read this file.

Contents:

General Considerations

Using Compressed Data Files

Using Btrieve in a Multiuser Environment
Setting Btrieve Options in WIN.INI

Configuring Novell Network LAN Manager (NLM).

General Considerations

When using Btrieve data with Visual Basic, keep the following

considerations in mind:

Btrieve data files must be in Version 5.1x format.

You must have the data definition files FILE.DDF and FIELD.DDF,

which tell Visual Basic the structure of your tables. These

files are created by Xtrieve or another. DDF file-building

program.

You must have the Btrieve for Windows dynamic-link library

WBTRCALL.DLL, which is not provided with Visual Basic.

This file is available with Novell Btrieve for Windows, Novell

NetWare SQL, and some other products for Windows that use Btrieve.

You cannot use Btrieve files that have Xtrieve security. To use

data files with Visual Basic, disable Xtrieve security.

Using Compressed Data Files

If you are using compressed Btrieve files, you must be sure that

the compression buffer Btrieve is using is adequate for your data.

The buffer size must be at least as large as the largest record

in your data files.

To ensure proper operation, set the compression buffer size

option (/u) in the [btrieve] section of your WIN.INI file. The

units for this setting are kilobytes, so if your largest record

is 2 K, you would add /u:2 to the Btrieve options line in WIN.INI.

For more information on setting options, see "Setting Btrieve

Options in WIN.INI" later in this file.

End of File.

No. of 50-Bit Codes	No. of 10-Bit Codes	No. of 13-Bit Codes
1564	125	21

Thus, the total number of bits in the compressed file $= 1564 \times 5 + 125 \times 10 + 21 \times 13 = 9343$ bits. Whereas, total no. of bits in the input file $= 13304$ bits. Therefore, the compression ratio $= 29.77\% \cong 30\%$. Running the same input file by the adaptive Huffman coding scheme (variable length coding), the compression ratio is little over 30%.

6. CONCLUSIONS

In this era of signal processing, text alphabets are considered as real-life signals and online text compression is the need of the day. In this paper, we have developed an analytical tool based on matrix algebra to characterize all the nearest neighbourhood periodic 2DCA transformations and highlighted its application in the field of text compression. We have proposed an architecture of a programmable CAM which suits the VLSI implementation for the text compression. Finally, a comparative study of Huffmann coding scheme is given. It is to be noted that Huffman coding scheme is a variable length coding, whereas our coding scheme is an almost fixed length coding scheme, which is easily implementable in VLSI chip form.

REFERENCES

1. A.R. Khan, P.P. Chowdhury, K. Dihidar, S. Mitra and P. Sarkar, VLSI architecture of a cellular automata machine, *Computers Math. Applic.* **33** (5), 79–94, (1997).
2. J. Von Neuman, *The Theory of Self-Reproducing Automata*, (Edited by A.W. Burks), Univ. of Illinois Press, Urbana, (1996).
3. S. Wolfram, Statistical mechanics of cellular automata, *Rev. Mod. Phys.* **55**, 601–644, (July 1983).
4. W. Pries, A. Thanailakis and H.C. Card, Group properties of cellular automata and VLSI applications, *IEEE Trans. on Computers* **C-35**, 1013–1024, (December 1986).
5. A.K. Das, Additive cellular automata: Theory and application as a built-in-self-test structure, Ph.D. Thesis, I.I.T. Kharagpur, India, (1990).
6. A.K. Das and P.P. Chaudhuri, Vector space theoretic analysis of additive cellular automata and its applications for pseudo-exhaustive test pattern generation, *IEEE Trans. on Computers* **42**, 340–352, (March 1993).
7. D.R. Chowdhury, S. Basu, I.S. Gupta and P.P. Chaudhuri, Design of CAECC—Cellular automata based error correcting code, *IEEE Trans. on Computers* **43**, 756–764, (June 1994).
8. D.R. Chowdhury, S. Basu, I.S. Gupta and P.P. Chaudhuri, CA based byte error correcting code, *IEEE Trans. on Computers* **43**, 371–382, (March 1994).
9. S. Nandi, B.K. Kar and P.P. Chaudhuri, Theory and applications of cellular automata in cryptography, *IEEE Trans. on Computers* **43**, 1346–1357, (December 1994).
10. N.H. Packard and S. WolForm, Two-dimensional cellular automata, *Journal of Statistical Physics* **38** (5/6), 901–946, (1985).
11. D.R. Chowdhury, I.S. Gupta and P.P. Chaudhuri, A class of two dimensional cellular automata and applications in random pattern testing, *Journal of Electronic Testing: Theory and Applications* **5**, 65–80, (1994).
12. K. Sutner, The σ game and cellular automata, *American Mathematical Monthly* **97**, (January 1990).
13. T. Toffoli and N. Margolus, *Cellular Automata Machines*, The MIT Press, (1987).
14. R. Barua and S. Ramkrishnan, The σ game, $\sigma+$ game and two dimensional additive cellular automata, (personal communication).
15. K.B. Dutta, *Matrix and Linear Algebra*, Prentice Hall, India, (1991).
16. B. Elspas, Theory of autonomous linear sequential networks, *TRE Trans. on Circuits* **CT-6**, 45–60, (March 1959).
17. D.R. Chowdhuri, Theory and applications of additive cellular automata for reliable and testable VLSI circuit design, Ph.D. Thesis, I.I.T. Khargpur, India, (1992).
18. D.E. Knuth, *The Art of Computer Programming—Seminumerical Algorithms*, Addison-Wesley, (1981).
19. A. Strole and H.J. Wunderlich, TESTCHIP: A chip for weighted random pattern genaration, evaluation and test control, *IEEE Journal of Solid-State Circuits* **26**, 1056–1063, (July 1991).
20. Y. Fisher, *Fractal Image Compression*, Springer-Verlag, (1994).
21. P.P. Choudhury, On cellular automata of different dimensions and their applications, In *Lecture at CSU Seminar*, ISI, (April 1994).