GF(2^p) CA Based Vector Quantization For Fast Encoding Of Still Images *

Kolin Paul Santashil Pal Chaudhuri+ Ranadeep Ghosal Biplab Sikdar
D. Roy Choudhury+

Department of Computer Science & Technology
Bengal Engineering College (Deemed University)
Howrah, West Bengal, India
+ Department of Computer Science & Engineering
Indian Institute of Technology
Kharagpur, West Bengal, India

Abstract

This paper sets a new direction for *vector quantization (VQ)* employed for image compression. The technology of $GF(2^p)$ CA has been employed for the VQ encoder in order to reduce the encoding time by more than one order of magnitude.

1 Introduction

Transmission of images over communication links faces a major problem in handling the large number of bits transmitted to represent the images faithfully. Hence the emphasis achieve better compression ratio at higher speed with acceptable quality. Also with the increasing demand for video telephony and other such information appliances, the availability of more efficient compression technology targeted specifically for a particular class of images is needed for the Information age . The technique of *Vector Quantization* gives very good image fidelity with a very fast decompression time. However, it suffers from a very high processing overhead at the encoding stage.

In the above background, this paper sets a new direction for use of *Cellular Automata (CA)* technology for **Image Compression**. We have fully characterized a new computation model in terms of **GF(2^p) Cellular Automata** in [1]. Based on this theory, we have developed a scheme for realizing the Vector Quantization encoder where the search time has been significantly reduced. The software version of the scheme achieves a speed up by more than one order of magnitude over the existing schemes. Significant speed with real time performance can be achieved by implementing the hardware version of the scheme on a VLSI chip.

In the above context, we briefly outline the CA preliminaries in Section 2. The GF(2^p) CA based scheme for encoding a 8x8 image block is next presented in Section 3 along with the experimental results. The VLSI architecture of the encoder is noted in Section 4.

2 CA Preliminaries

A *Cellular Automata (CA)* consists of a number of cells arranged in a regular manner, where the state transitions of each cell depends on the states of its neighbors. For an n-cell one dimensional CA, it has been shown in [2] that the linear operator is an $n \times n$ matrix referred to as the T matrix, where the next state of CA is given by $T \times Current\ State$. The complete characterization of CA behavior based on the T matrix has been reported in [2]. A brief introduction to a special class of CA referred to as *Multiple Attractor CA (MACA)* [2] is next introduced as the research reported in this paper employs MACA for Vector Quantization.

2.1 MACA As A Classifier of Binary Patterns

The state transition behavior of a 4-cell MACA is shown in Figure 1(a). In general, an n-bit multiple-attractor CA with k attractors can be viewed as a natural classifier. The generic term *Classifier* is used in the the sense that it classifies the set of n-bit patterns into k distinct classes, each class containing the set of states of an attractor basin [2]. Now, as per the theorem 4.6 ([2], Chapter 4) in the binary patterns of

*This research is partially supported by SAS, Banglore, INDIA

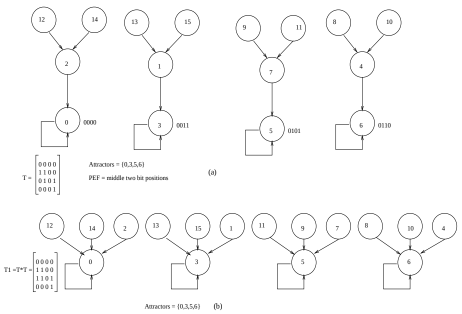

Figure 1: (a) A 2-depth MACA and (b) The corresponding 1-depth MACA

k attractors there exists $\log_2 k$ bit positions that generate pseudo-exhaustive patterns. This field (referred to as **PEF (Pseudo Exhaustive Field)**)will identify the class of patterns falling in a basin uniquely. A special type of multiple attractor 1-depth CA referred to as D1-MACA is employed to develop the classifier for image data.

An MACA having the depth of 1 is referred to as D1-MACA.

The theory and design procedure of building a MACA is described in [2]. This theory has been extended in [1] to develop a GF(2^p) MACA.

3 Application of GF(2^p) CA for Vector Quantization

A vector quantizer [3] maps the n-dimensional vectors in the vector space R^n - range of R is from 0 to 255 - to a finite set of vectors, called the *codebook*. Each vector of this *codebook* is called a *code vector*. Associated with each *code vector* is a nearest neighbor region called the *Voronoi region*. The set of *Voronoi regions* partition the entire vector space R^n into non-overlapping regions. The code vector chosen to represent any arbitrary input vector is the representative vector of the *Voronoi region* it belongs it. This representative vector is called a **codeword**. The set of codewords represents the *codebook*.

A vector quantization basically consists of two operations. The encoder - to encode each block, and a decoder - to get back the block from the code. The encoder, as shown in Figure 2, takes in the input vector and outputs the index of the code vector in the codebook that gives minimum distortion.

Once this index of code vector is found, it is sent to the receiver end of the communication channel. The decoder on receiving this index, replaces it with the

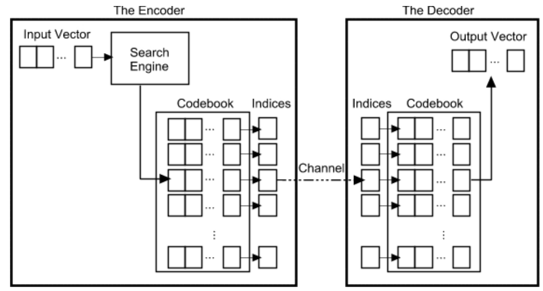

Figure 2: Encoder and Decoder

associated *codeword* found from the codebook kept on the receiver side.

In building the codebook, a training set of 12 pictures are used for the present work . All the pictures are the upper half of the human body.

We have used the standard LBG algorithm [3] to design the codebook. The size of our codebook is 8192 which is constructed using 14900 training vectors collected from the 12 images. Each vector is an image block of $8x8$ pixels (each pixel of 8 bits), that can be represented as a string of symbols (in $GF(2^8)$) of length 64.

3.1 Designing the Codebook

We now discuss the design and working of the **GF(2^p) CA based classifiers which effectively act as a codebook** for our scheme. In order to expose the underlying principle we illustrate the design with a simple example. We assume that each of the pixel value is coded with 2 bits, that is the pixel values range from 0 to 3. We also assume that there are four classes in which all the training image blocks of size 2×2 can be clustered. Each block is represented by a GF(2^2) symbol string of length 4. A few example classes are illustrated below:

Class P1 : {3210,1212,2122,2111} - 1212
Class P2 : {2310,1221,1222,1211} - 1221
Class P3 : {3201,1122,1212,2131} - 1122
Class P4 : {3211,2120,2121,2113} - 2120

Let us assume that the second element in each class represents the codeword of that class. Therefore our codeword array is the set {1212,1221,1122,2120} - this is the *Codebook* for the present case. This is stored in a linear array that can be addressed by two bits as shown in the following table.

00	1212
01	1221
10	1122
11	2120

This table is stored at the decoder. The index to the table is sent from the source after encoding the image block with the index value. The standard Vector Quantization techniques store this codebook at the source and identify the index by searching the codebook. **By contrast in our scheme we store the set of CA configurations that are used to build the multi class Classifier** (Section 2.4) for the 4 classes (P1 to P4) out of which the codebook has been built. The subsequent discussions illustrate this design methodology with reference to the above example.

Since we have used 2×2 blocks with 4 pixel values, we need a 4 cell GF(2^2) CA which will be used in two stages to build the classifier. As discussed earlier, we first obtain the CA T1 which classifies the temporary classes X1 given by {{P1},{P2}} and X2 given by {{P3},{P4}}. Next we find the CA T2 which classifies {P1} and {P2} and also CA T3 which classifies {P3} and {P4}. Figure 3 illustrates the scheme. These CA configurations {T1,T2,T3} are stored in a memory referred to as CA configuration **(CAC) memory**. We also store 0 or 1 in the **PEF (Pseudo Exhaustive Field) Memory** to be accessed with the pseudo exhaustive field of the attractor of the CA chosen as the two class classifier at a particular level — 0 for the class corresponding to the left edge and 1 for the class corresponding to the right edge as shown (Figure 3). Thus the conventional codebook for our case gets replaced with the CA configurations stored in the CAC memory.

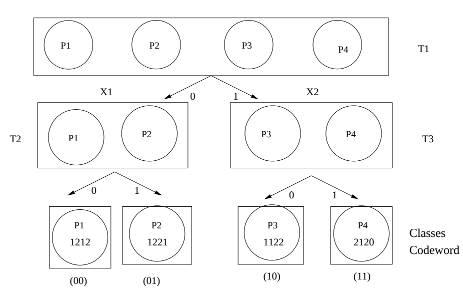

Figure 3: The logical structure of Classifier {T1,T2,T3}

GF(2^8) CA Based Classifiers

Our training set consists of 14900 vectors, each being a 8×8 pixel block. By employing LBG algorithm (noted above), the 14900 blocks are grouped in 8192 classes.

3.2 Searching the codebook

For a given input codeword(c), we need to identify the codebook entry (that is, the codeword) closest to (c).

Suppose we have to identify the class of an input pattern $p_i = 3201$. The Classifier is configured with the CA configuration T1 and allowed to run for one clock pulse with p_i as the initial state. The PEF memory location addressed by the PE field of the attractor of the CA (represented by *T1*) will have a (1) and thus indicates that p_i belongs to class X2. We then reconfigure the CA with configuration T3 and with p_i as the initial state, run it for one cycle. The (0) value of the PEF memory location for the attractor of *T3* indicates that $p_i = 3201$ belongs class P3 addressed by the bits 10 . Thus we need to send the codebook index (10) to the decoder that identifies the input as 1122 from the table that represents the conventional codebook.

The algorithmic steps for the general search scheme to generate the codebook index are as follows:

Experimental Results

The algorithms are coded in C and run on a DEC workstation. The algorithm is tested with many different pictures outside the training set. The results of the experiments are shown below. Table 1 gives the PSNR values of the training set images when compressed and decompressed using this scheme. Table 2 gives the results of employing the scheme on images outside the training set.

Table 1

1.	Lena	33.53
2.	Julie	34.56
3.	Girl1256	32.69
4.	Girl3	37.98
5.	ash	35.98
6.	Michelle	37.89

Table 2

1.	sample 1	25.53
2.	sample 2	25.56
3.	sample 3	26.69
4.	sample 4	27.98
5.	sample 5	25.98

Encoding Speed - We note below a comparison of the time required to run the software version of the proposed algorithm with that of a corresponding Full Search algorithm.

Per Block	CA Search	Full Search
Memory Accesses	$5 \times 64 \times 13$	$64 + 64 \times 8192$
Number of comparisons	1×13	8192×64

These tables establish the claim that the software version of the scheme achieves a speedup of more than one order of magnitude over the existing encoding schemes. The hardwired version achieves a significantly higher speedup achieving real time performance. This claim has been substantiated in the VLSI design of the encoder proposed next.

4 The VLSI Architecture Of the Encoder

The hardwired version of the proposed CA based scheme can be economically realized. The simple, regular, modular and cascadable structure of CA makes it amenable for VLSI designs. The basic nodules of the encoder block, noted in Figure 4 are described below.

1. Input Register: It holds the 8×8 image block to be encoded.

2. Programmable CA (PCA) [2]: A CA with switches (Figure 4(a)) that can be programmed to realize different CA configurations at different time steps. It consists of 4 modules listed below :

   (a) 64 cell 3 neighborhood GF(2^8) CA DFFs — $64 \times 8 = 512$ DFFs

   (b) 512×8 switch array — each DFF can have programmable 8 neighborhood;

   (c) 512×8 bit CA Configuration Register (CACR) — to control the 512×8 switches.

   (d) 512×7 2-input XOR gates — The input to a DFF comes out of a 8 input XOR gate

3. Pseudo Exhaustive Field (PEF) Extractor —- the PE field location can be precomputed (as noted in Chapter 4 of [2]) and stored in the CAC memory(noted below).

4. 512×1 PEF Memory —- It stores a 0/1 to identify the next lower level class the input 8×8 image block belongs to.

5. A 13 bit address generation circuit — This accumulates the 13 address bits that encode the input 8×8 image block.

6. CA Configuration (CAC) Memory — It is a $8192 \times 512 \times 8$ bit DRAM memory organized as 32 banks of 8192×128.

7. The controller — This block controls the sequential operations on the data path.

Execution Time

The steps 2 to 5 are executed 13 times to encode a 8×8 image block with 13 bits. Each of the steps (b) to (e) can be completed in one clock cycle. The slowest operation will be the access time to the DRAM bank. Providing allowance for low cost DRAM with cycle speed of 400nsec and other associated delay of the control circuit, the encoder of Figure 4 can be operated with a 20MHz clock with cycle time of 500 nsec. So for the

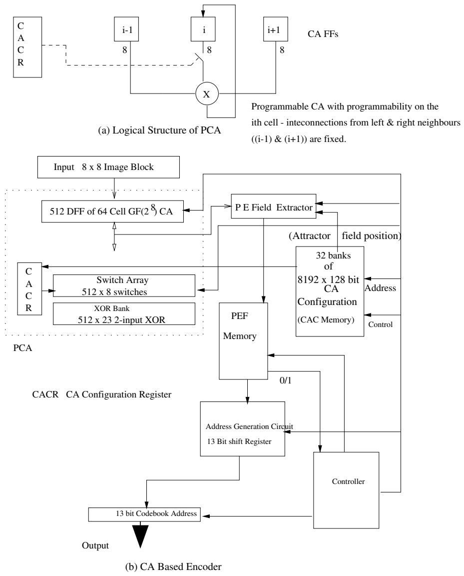

Figure 4: The VLSI architecture of GF(2^p) CA Based VQ Encoder

execution of steps (b) to (e) it needs $4 \times 500 = 2$ microseconds. This has to be repeated 13 times to encode a 8×8 block. That is, the net time to encode a block is 26 microseconds. Hence time to encode a 352×240 image is $26 \times ((352 \times 240)/(8 \times 8)) = 26 \times 1320 = 34.32$ millisecond. Thus the proposed GF(2^p) CA based encoder achieves real time performance in the sense that it can handle close to 30 images per second.

5 Conclusion

This paper has reported a new algorithm for significantly reducing the encoding time of Vector Quantization based image compression schemes.

References

[1] K. Paul, *Theory and Application of GF(2^p) Cellular Automata*. PhD thesis, B. E. College , (Deemed University), Howrah, India, 1999.

[2] P. P. Choudhuri, D. R. Chowdhury, S. Nandi, and S. Chatterjee, *Additive Cellular Automata – Theory and Applications*, vol. 1. IEEE Computer Society Press, California, USA, ISBN 0-8186-7717-1, 1997.

[3] N. M. Nasrabadi and R. A. King, "Image coding using vector quantization," *IEEE Trans. Commun.*, vol. ASSP-36, pp. 957–971, 1988.

[4] B. Ramamurthy and A. Gersho, "Classified vector quantization of images," *IEEE Trans. Commun.*, vol. ASSP-34, pp. 1258–1268, 1986.