NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

ISBN 0-315-53275-0

Canada

Algorithms and Architectures for Image Coding using Vector Quantization

By
S. Panchanathan

A THESIS
submitted to the School of Graduate Studies and Research
in the partial fulfillment of the requirements
for the degree of
DOCTOR OF PHILOSOPHY
in
Electrical Engineering

Ottawa-Carleton Institute of Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
OTTAWA, ONTARIO, K1N 6N5.

Dedicated to my beloved Parents

ABSTRACT

Vector Quantization is a powerful technique for low bit rate image coding. The two basic steps in vector quantization are codebook generation and encoding. The coding performance of vector quantization can be improved by employing adaptive techniques. The applicability of vector quantization is, however, limited by its computational complexity. Recently, architectures which implement vector quantization for image encoding have been reported in the literature, but the maximum speed-up achieved by all these techniques is still not sufficient for codebook generation in real-time. In this thesis, we present a brief review of image coding, vector quantization, architectures for image processing, and architectures which implement vector quantization. Two new adaptive algorithms for image vector quantization and two new architectures to implement vector quantization in real-time are then presented.

In the first algorithm (VQRUC), a reduced set of codewords from the universal codebook is used to code an image. An extension of this algorithm is the VQRUC-1 technique which involves one iteration on the reduced codebook to adapt the codewords to the image to be coded. Simulation results demonstrate that this algorithm provides improved coding performance and gains in computational complexity. The second algorithm (VQMMC) is a super-adaptive technique based on a mini-max criterion. Here, the codebook is generated on the fly from the input vectors to be coded. As it is a single-pass technique, real-time implementation is possible. Simulation results demonstrate good coding performance and the absence of large errors.

The first architecture is based on the concept of systolic arrays. Here, the encoding and the codebook generation operations in vector quantization are overlapped in the same structure. This results in savings in the computation and transfer times for the new codewords, which in turn, makes possible real-time implementation of sub-optimal codebooks. The second architecture is based on an entirely novel approach using a content-addressable memory (CAM). The input vectors are stored in the CAM array and the codewords are used as search arguments and a match is sought by accessing all the input vectors in parallel. This results in a large speed-up as the parallelism in the direction of input vectors is exploited, which in turn, makes possible real-time optimal codebook generation and encoding (< 33 ms). The regular and iterable structure of the two architectures are particularly well suited for VLSI implementation.

ACKNOWLEDGEMENT

It has been an honor and a privilege to work with my supervisor, Dr. Morris Goldberg. It is his constant encouragement, expert guidance, and brilliant insights which has led me to the successful completion of my thesis. He has been a true mentor and mere words cannot express the gratitude I feel.

I would like to record my sincere thanks to the members of my advisory committee, Dr. B.A. Bowen, and Dr. E. Petriu for their valuable guidance and encouragement. My special thanks are also due to Dr. S. Shlien of C.R.C, Ottawa for his valuable suggestions and discussions.

I would also like to thank the staff of the Department of Electrical Engineering, University of Ottawa for their cheerful assistance, help and support.

I wish to thank all my colleagues and friends for their help and encouragement. A special note of thanks is due to Dr. Limin Wang for his valuable suggestions and discussions.

I would like to acknowledge the financial support from the Telecommunications Research Institute of Ontario and NSERC for my research program.

Finally, a special note of thanks goes to my family. It would not have been possible to finish this long-term work without their support.

LIST OF SYMBOLS

Chapter 1

DPCM	Differential pulse code modulation.
VQ	Vector Quantization.
VQUC	Vector quantization with an universal codebook.
FAM	Fuzzy associative memory.
VQRUC	Vector quantization with a reduced universal codebook.
VQRUC-1	Vector quantization with one pass on the reduced universal codebook.
VQMMC	Vector quantization with the mini-max error criterion.
$S_p(NL)$	Speed-up in the directions of N and L.
VLSI	Very large scale integration.
CAM	Content addressable memory.
MSE	Mean square error.

Chapter 2

$P(X_i)$	Probability of X_i.
Q(Y/X)	Conditional probability of Y about X.
T(Y)	Probability of the input message set.
$I_N(X,Y)$	Mutual information between X and Y for a block of length N.
$H_N(X)$	N^{th} order entropy of X.
D	Average distortion per pixel.
R(D)	The rate distortion function.
d(X,Y)	Distortion between X and Y.
$X_{i,j}$	The (i,j)th pixel of the original image X.
$Y_{i,j}$	The (i,j)th pixel of the coded image Y.
NMSE	Normalized mean square error.

Q	Mapping from a vector set to a finite codebook.
V	The input vector.
C	The output vector (codeword).
$\{\Pi_p\}$	The partition of the input vector space.
D_m	The average distortion in the mth iteration.
VQIAC	Vector quantization with an image adaptive codebook.
R_t	The total bit rate.
R_l	The bit rate for the labels.
R_c	The bit rate for the codebook.
SISD	Single instruction single data.
$S_p(1)$	Speed-up (unity) using an SISD machine.
O(KLN)	Complexity of order KLN.

Chapter 3

SIMD	Single instruction multiple data.
TTL	Transistor transistor logic.
MOS	Metal oxide semiconductor.
ECL	Emitter coupled logic.
PE	Processing element.
FFT	Fast fourier transform.
$B_{i,j}$	The (i,j)th bit cell.
W_i	The ith word cell.
M_i	The match signal line corresponding to the ith word.
A_i	The address line corresponding to the ith word.
S	Search argument register.
I_i	The intermediate response register corresponding to the ith word.

R_i	The response register corresponding to the ith word.
T	The set of templates.
C	The set of input patterns.
Δ	The threshold (delta).

Chapter 4

$T_{en(effective)}$	The effective execution time of the encoding algorithm.
$T_{en(SISD)}$	The execution time of the encoding algorithm using a SISD machine.
S_p	Speed-up factor.
T_c	The cycle time of the processor.
T_e	The execution time of the basic (distortion) operation using an SISD machine.

Chapter 5

R_u	The bit rate for usage map transmission.
N_u	The number of codewords in the reduced universal codebook.
R_d	The bit rate for difference codebook transmission.

Chapter 6

LRU	Least recently used codeword.
R_{lp}	The bit rate for primary labels.
R_{ls}	The bit rate for secondary labels.
M_p	The size of the primary codebook.
M_s	The size of the secondary codebook.
N_{lp}	The number of labels corresponding to the primary codebook.
N_{ls}	The number of labels corresponding to the secondary codebook.

VQMMC-P2	Vector quantization with mini-max error criterion with 4-D vectors and a primary codebook of size 2.
VQMMC-P4	Vector quantization with mini-max error criterion with 4-D vectors and a primary codebook of size 4.
VQMMC-P16	Vector quantization with mini-max error criterion with 16-D vectors and a primary codebook of size 2.

Chapter 7

RAM	Random access memory.
V_{in}	The element of the vector input to the cell.
V_{out}	The element of the vector output from the cell.
D_{in}	The distortion input to the cell.
D_{out}	The distortion output from the cell.
V'_{in}	The element of the vector input to the cell (reverse mode).
V'_{out}	The element of the vector output from the cell (reverse mode).
LB_{in}	The label of the vector input to the cell.
LB_{out}	The label of the vector output from the cell.
T_l	The latency using the systolic array.
T_{cen}	The execution time for centroid computation.
T_{tr}	The time for transferring the centroids into the array.
T_g	The execution time for the codebook generation algorithm.
T_{gp}	The execution time for the codebook generation algorithm using the proposed systolic array architecture.

Chapter 8

VQ_CAM	The normal implementation of vector quantization using the CAM.

VQ_CAM_INVERT	The implementation of vector quantization using the CAM by inverting the order of comparison.
IR_i	The index register corresponding to the ith word.
VQIAC-CAM	The VQIAC algorithm implementation using the CAM.
$T_{g(SISD)}$	The execution time for codebook generation using an SISD machine.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1. INTRODUCTION

Image coding is an evolving field of specialization of data compression with applications in the areas of television transmission, video conferencing, transmission of remote sensing images, image data bases, archiving medical images, facsimile transmission of printed material, and graphics images. The goal of image coding is to reduce the bit rate for transmission or storage while maintaining an acceptable image quality. A number of image coding techniques have been developed, examples include DPCM, transform coding and hybrid coding. In these techniques the image data is coded using scalar quantization. Shannon's rate-distortion theory states that it is always possible to achieve, in theory, a better performance in coding vectors instead of scalars, even though the source is memoryless. However, this does not provide a constructive design technique for vector coders. Vector quantization design techniques and various vector quantization structures have only been developed in recent years for speech and image coding applications [1]. Gersho and Cuperman [2] have presented a tutorial paper which reviews vector quantization for speech coding. Gray [1] and Makhoul *et al* [3] have given a broader and more comprehensive review of vector quantization. Nasrabadi and King [4] have recently presented a review of vector quantization for image coding.

In vector quantization (VQ), the first step is to decompose the input image into a set of vectors. A subset of vectors is then chosen as the training sequence from which the codebook is generated, normally using the generalized Lloyd clustering algorithm proposed by Linde, Buzo, and Gray (LBG) [5]. The LBG algorithm generates in an iterative manner a locally optimum codebook of size, N, from a given initial codebook, refining the codebook at each iteration until the distortion is within acceptable limits. In the quantization process, for each input vector, the codebook is searched to obtain the closest match codeword (VQUC). Compression is achieved by transmitting the index (label) of the codeword. Reconstruction of images can be implemented by simple table look-up techniques where the label is used as an address to a table containing the codewords. The coding performance may be improved by employing adaptive techniques which track the image statistics from image to image [6].

The basic computation in a VQ encoder is the distance calculation in which the distortion

between an input vector and a codeword is computed. VQ algorithm is compute intensive making it difficult to implement in real-time. Recently, architectures which implement VQ for speech and image coding applications have been reported in the literature. The first class of architectures is based on efficient execution of the distortion computation by using fast processors [8] or a pipeline of fast processors [9-12,16] , where each processor in the pipeline executes a portion of the distortion computation. This yields sufficient speed-up for real-time speech coding. However, as the throughput rate for image coding is three orders of magnitude greater than that for speech coding, parallelism in other directions has to be exploited. Another class of architectures is based on parallel and pipeline execution of the VQ algorithm. For example, Sun and Hsu have reported on a parallel and pipelined architecture [15] in the directions of vector and codeword dimensions, respectively. Davidson, *et al* , [7] have proposed a family of architectural techniques based on the concept of systolic arrays to implement vector quantization. These include: word-level architecture, bit-level architecture, two-dimensional array architecture, sub-linear and super linear array architectures. Abut *et al* have presented an architecture based on Fuzzy associative memory (FAM)[17] chips[16] where multiple input vectors are processed in parallel. A third class of architectures implement a modified form of VQ. For example, if a tree-search VQ [14,16] or multi-stage VQ [13] is used, a smaller number of codewords can be searched resulting in reductions in computational complexity. Nasrabadi and Feng [18] have reported using neural networks for codebook generation in image coding. The maximum speed-up achieved by all these implementations is still not sufficient for executing the codebook generation algorithm for image coding in real-time, since it involves several iterations of the encoding algorithm. The objective of this thesis work is to design new adaptive algorithms and architectures which implement VQ for image coding in real-time.

In this thesis, two new adaptive techniques and two new architectures for image coding using vector quantization are presented: 1) *Algorithms for Image Adaptive Vector Quantization* (VQRUC and VQRUC-1) [80]; 2) *A Mini-max Error Criterion based Algorithm for Image Adaptive Vector Quantization* (VQMMC) [81]; 3) *A Systolic Array Architecture for Vector Quantization* [82,83]; and 4) *A Content-addressable Memory Architecture for Vector Quantization* [84,85].

In VQRUC, a subset of codewords in the universal codebook which are used by the input vectors of the given image are identified which form the reduced codebook. The input vectors are coded using the reduced codebook and the corresponding labels are determined. An usage map is then created to indicate the codewords used. An entropy coder [19,20] is used to transmit the labels. This technique results in considerable savings in the bit rate for label transmission for the same coding performance. In VQRUC-1, a new codebook is generated using one iteration of the LBG algorithm on the reduced codebook. The new codebook is now well adapted to the input image resulting in improved coding performance. The difference between the new and the old codewords are transmitted as overhead using an entropy coder. Simulation results demonstrate that the two proposed adaptive techniques [80] provide a good compromise between coding performance and computational complexity resulting in a very good performance at a reduced complexity.

VQMMC is an intra-image adaptive technique [81] which employs a criterion that minimizes the maximum error. Here, the codebook is generated on the fly from the input vectors to be coded. A primary codebook of size, 2,4, 8 or 16 is typically used to store the frequently used codewords. A larger secondary codebook is used to store the less frequently used codewords. Both the transmitter and receiver maintain identical codebooks and hence keep track of any changes without any overhead information. An entropy coder is used to transmit the labels and the codewords. The simulation results demonstrate excellent rate-distortion performance and also confirm the absence of large errors when compared to the VQUC technique. As it is a single-pass technique, real-time implementation is possible. This algorithm can be mapped onto stack based architectures.

In the systolic array architecture, the encoding and codebook generation operations are overlapped in the same structure. A basic systolic cell is designed with two modes of operation (forward and reverse). In the forward mode, the cell executes the basic operation in a VQ encoder, namely, distortion computation. In the reverse mode, the cell executes the new codeword (centroid) computation operation. An array of L x N cells are connected in parallel and pipeline in the directions of vector dimension, L, and codeword dimension, N, respectively. This results in a speed-up $S_p(NL)$. The two modes of operation take place simultaneously with a delay buffer used to synchronize the operations. The important features of this architecture are as follows: (i) There

is no need for separate hardware to compute the new centroids; (ii) There is no need for a high-speed interface to transfer the new centroids into the systolic array; and (iii) There are no delays involved in the computation and transfer of new centroids. This makes possible sub-optimal codebook generation and encoding in real-time. The regular and iterable structure is well suited for VLSI implementation of the architecture.

In the content-addressable memory (CAM) architecture, the input vectors are stored in the CAM array and the codewords are used as search arguments and a match is sought by accessing all the input vectors in parallel. This means that matching must be performed from the perspective of the codewords; namely, for a given codeword, all input vectors are evaluated in parallel. Hence, parallelism in the directions of vector dimension, L, and the direction of the number of input vectors, K are exploited resulting in a speed-up of order $S_p(KL)$. This speed-up coupled with the gains in the execution time for the basic distortion operation, implies that even optimal codebook generation is possible in real-time (< 33 milliseconds). Note that in using the CAM, the conventional MSE measure is substituted by the absolute difference measure. The use of this measure causes little degradation and, in fact, the simulation results shows that the occurrence of large errors is limited. The regular and iterable architecture is particularly well suited for VLSI implementation.

The rest of the thesis is organized as follows. Chapter 2 presents a review of image coding and vector quantization. The review on architectures for image processing is presented in chapter 3. This follows in chapter 4, with a review of the research contributions on architectures for vector quantization. Chapters 5-8 present our research contribution to the thesis. Chapters 5 introduces new image adaptive techniques for vector quantization. In chapter 6, a mini-max error criterion algorithm for vector quantization is presented. This follows with a systolic array architecture for vector quantization in chapter 7. An entirely novel approach is presented in chapter 8 using a content-addressable memory (CAM). This technique provides an elegant method of implementing VQ in real-time by exploiting the parallelism in the direction of the input vectors. Finally, chapter 9 briefly summarizes the main results, lists the current research work, and provides some suggestions for further investigation.

2. REVIEW OF IMAGE CODING AND VECTOR QUANTIZATION

In this chapter, a short review of image coding techniques is first presented. Distortion measures used are then discussed in section 2.2. This is followed by a review of vector quantization in section 2.3. In section 2.4, the bit rate calculations are presented. This is followed by an analysis of computational complexity issues in section 2.5. A brief review of methods to reduce computational complexity is then presented in section 2.6 followed by a summary in section 2.7.

2.1 IMAGE CODING

The goal of image coding is to reduce the bit rate for transmission or storage while maintaining an acceptable fidelity or image quality. Typical applications include, television transmission, video conferencing, transmission of remote sensing images obtained from satellites and reconnaissance aircraft, facsimile transmission of printed material and graphics images; storing multispectral images, finger prints, and archiving medical images. A number of bandwidth compression techniques have been developed to exploit the psychovisual as well as statistical redundancies in the image data. These techniques can be broadly classified as lossless coding and lossy coding techniques.

Lossless Coding : Lossless coding removes, or at least reduces, the statistical redundancy of the image in such a way that redundancy reduction operation is a reversible process; i.e., the original image can be recovered exactly. Two popular lossless coding techniques are Huffman coding and run-length coding. The Huffman coding procedure results in a variable length code in which the number of bits for a source sample varies approximately inversely with the corresponding source probability. In run-length coding, a run is defined as a sequence of consecutive pixels of identical values along a specified direction, for example, the horizontal scan line. If the runs are long, significant reduction in the average bit rate may be achieved by simply transmitting the start and

length of the run.

Lossy Coding : In lossy coding, the objective is to reduce the transmission bit rate subject to some quality constraint. Historically, lossy image coding techniques have generally followed two paths: predictive coding and transform coding. Both predictive coding and transform coding possess relative advantages and limitations [21-28]. From an implementation point of view, predictive coding systems have much lower complexity both in terms of the memory requirements and the number of operations to be performed compared to transform coding systems. However, transform coding achieves a relatively higher image fidelity reconstruction at lower bit rates (less than 1 bit/pixel) while predictive coding results in a superior coding performance at higher bit rates (around 2 bits/pixel) [22]. A second difference is that the errors due to quantization in transform coding are distributed over the entire image (during inverse transformation) whereas the errors in predictive coding are locally concentrated. Furthermore, predictive coding is more sensitive to changes in statistics of the image than transform coding. Hybrid transform/predictive coding provides an effective compromise of the advantages and disadvantages of transform and predictive coding. Good results are reported at a moderate bit rate (around 1-2 bits/pixel)[22].

One of the major disadvantages with all of these techniques is that the quantization is performed on individual real-valued samples of waveforms or pixels of images. Furthermore, these techniques are not optimal since the processed samples are still somehow correlated or dependent. Shannon's rate-distortion theory states that it is always possible to obtain a better performance in coding vectors instead of scalars, even though the source is memoryless. This provided the impetus for the design of vector coders. However, Shannon's theory has limited impact on the actual system design because 1) the theory does not provide constructive design techniques for vector coders, and 2) traditional scalar coders often yield satisfactory performance with enough adaptation and fine tuning. Extensive studies of vector quantizers or multidimensional quantizers have recently been performed by many researchers [29-33]. The design of optimal vector quantizers from empirical data were proposed and extensively studied by Linde, Buzo and Gray [5] using a clustering approach which will be discussed in detail in section 2.3.

2.2 RATE DISTORTION FUNCTION

The rate distortion function R(D) is the minimum average rate (bits/element), and hence minimum channel capacity, required for a given average distortion level D. To describe this in a more quantitative way, suppose the source is a sequence of intensity values of discrete picture elements (pixels), and that these levels are encoded by successive blocks of length N. Each block of elements is then described by one of a denumerable set of messages $\{X_i\}$ with probability function $P(X_i)$. Any given system is now described mathematically by the conditional probability $Q(Y_j / X_i)$, of a message $\{Y_j\}$, at the output of the decoder, given a source input, X_i. The probability of the output message set is, therefore,

$$T(Y_j) = \sum_i P(X_i) Q(Y_j/X_i) \qquad (2.1)$$

The information transmitted is called the average mutual information between Y and X, and is defined, for a block of length N, as follows:

$$I_N(X,Y) = \sum_i \sum_j P(X_i) Q(Y_j / X_i) \log\{ Q(Y_j / X_i) / T(Y_j) \} \qquad (2.2)$$

In the case that encoding is error free so that Y = X, then

$$Q(Y_j / X_i) = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases} \qquad (2.3)$$

$$T(Y_j) = P(Y_j)$$

and

$$I_N(X,Y) = \sum_i \sum_j P(X_i) \log\{ P(X_i) \} = H_N(X) \qquad (2.4)$$

the Nth-order entropy of the source. The optimum-error free coding requires at least $H_N(X)$ bits [34].

If, as in the more general situation, source coding results in error so that $Y \neq X$ at least some of the time, then

$$I_N(X,Y) = H_N(X) - H_N(X/Y) \qquad (2.5)$$

where $H_N(X/Y)$ is the entropy of the source, given the observed decoder output Y. As the entropy is a positive quantity, the source entropy is the upper bound to the mutual information; that

is,

$$I_N(X, Y) \leq H_N(X) \qquad (2.6)$$

Let $d(X, Y)$ be the average distortion between X and Y, then the average distortion per pixel is given by,

$$D(Q) = \frac{1}{N}\{E[d(X, Y)]\} = \frac{1}{N}\{\sum_i \sum_j d(X_i, Y_j) P(X_i) Q(Y_j / X_i)\} \qquad (2.7)$$

The set of all conditional probability assignments, $Q(Y/X)$, that yield average distortion less that or equal to D^*, can be written as:

$$\{Q: D(Q) \leq D^*\} \qquad (2.8)$$

The N-block rate distortion function is then defined as the minimum of the average mutual information, $I_N(X, Y)$, per pixel :

$$R_N(D^*) = \min_{Q:\, D(Q) \leq D^*} \{\frac{1}{N}[I_N(X, Y)]\} \qquad (2.9)$$

The limiting value of the N-block rate distortion function is called simply the rate distortion function,

$$R(D^*) = \lim_{N \to \infty} \{R_N(D^*)\} \qquad (2.10)$$

It should be clear from the above discussion that Shannon's rate distortion function $R(D)$ is a lower bound on the transmission rate required to achieve average distortion D. The rate at which a source produces information subject to a requirement of perfect reproduction is called the entropy of the source. It follows that the rate distortion function is a generalization of the concept of entropy. Shannon's coding theorem states that one can design a coding system with rate only negligibly greater than $R(D)$ which achieves the average distortion D. In other words, the rate distortion function $R(D)$ specifies the minimum possible rate required to transmit an image with average distortion level D. A plot of a typical rate distortion function is shown in Fig. 2.1. It is seen to be monotonically non-increasing, which is reasonable in that the higher information-rate representations should lead to the smaller (strictly, non-increasing) average distortion. It is also clear that the distortion at rate $R = 0$ should be finite if the input variance is finite [35]. Let us now look at a suitable distortion measure for a coding system.

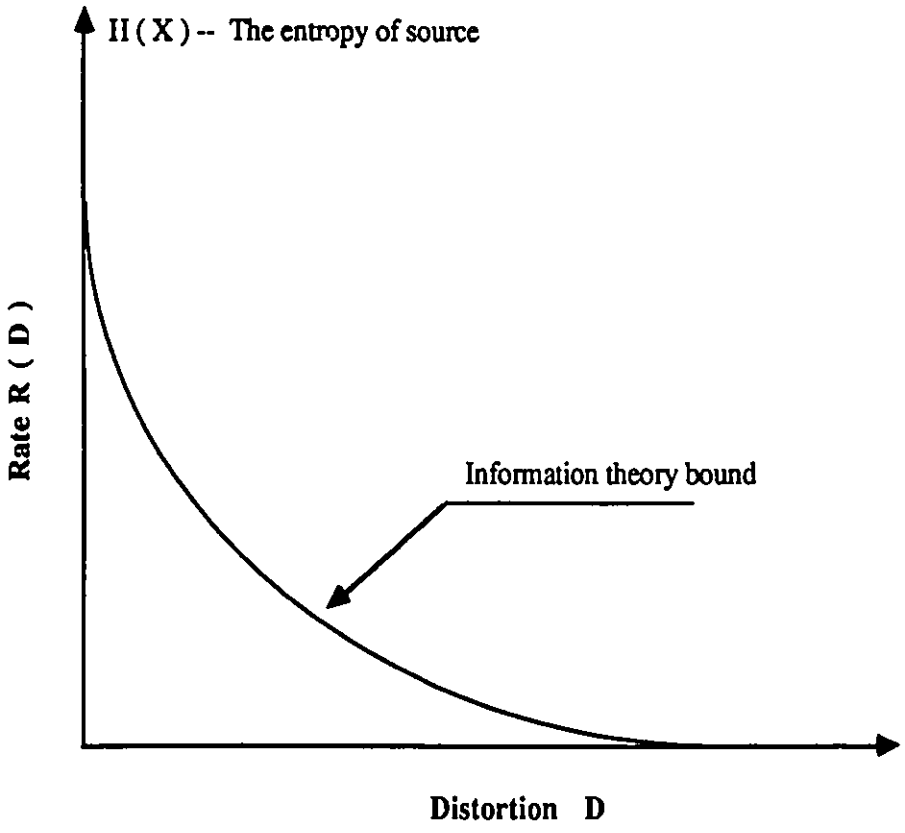

Fig. 2.1 An example of the rate distortion function for a discrete-amplitude source

2.2.1 DISTORTION MEASURE

A distortion measure for a coding system assigns a cost d(x, y) for reproducing an input, x, by an output y. The performance of the coding system can be evaluated using the average distortion E[d(X, Y)],

$$E(d(X,Y)) = \sum_{\text{all } x,\, y} d(x,y)\, p(x,y) \qquad (2.11)$$

To be of value for both design and comparison, a distortion measure should be:

1) Computable, so that it can be efficiently evaluated in real-time;

2) Subjectively meaningful, so that it correlates well with human visual observations; and

3) Tractable, so that it is amenable to mathematical analysis.

Perhaps, the most popular distortion measure used in image coding is the average mean square error (MSE) [22], defined as,

$$MSE = 1/(M \times M) \{\sum E (X_{ij} - Y_{ij})^2\}, \; i,j = 0, 1, 2, ..., M-1 \qquad (2.12)$$

where, X_{ij} and Y_{ij} are, respectively, the (i,j)th element of the original image and the coded image. Experimentally, the average mean square error is often estimated by,

$$MSE = 1/(M \times M) \{\sum (X_{ij} - Y_{ij})^2\}, \; i,j = 0, 1, 2, ..., M-1 \qquad (2.13)$$

The mean square is often normalized by the average energy of the original image (NMSE),

$$NMSE = \sum (X_{ij} - Y_{ij})^2 / \sum X_{ij}^2, \; i,j = 0, 1, 2, ..., M-1 \qquad (2.14)$$

The major advantages of the normalized mean square error (NMSE) are its intuitive appeal (large errors are given more importance than small errors), its ease of computation, and its mathematical tractability. A variant form, more easily calculable, is the normalized absolute error between the original and coded images, defined as [22],

$$e = \sum |X_{ij} - Y_{ij}| / \sum |X_{ij}|, \; i,j = 0, 1, 2, ..., M-1 \qquad (2.15)$$

However, neither the mean square error nor the absolute error is found to correlate very well with subjective ratings obtained by presenting the coded images to human observers. Two images having the same mean square error or absolute error may give very different subjective evaluations. The significant features of the human visual system with regard to (monochrome) image coding

evaluations are its logarithmic sensitivity to light intensity and its nonuniform response to spatial frequencies [22]. There has been considerable effort to yield a distortion measure that is better correlated with subjective results. One approach is to preprocess both the original and coded images prior to formation of the mean square error [22]. Examples of preprocessing include point transformations, such as, logarithmic and power-law transformations, and spatial transformations, such as, Laplacian, gradient and convolution operations [22]. Another approach investigated [28,36] is to pass the error image between the original and coded images through an operator designed to model the processing steps for the human visual system.

Although the preprocess steps seem to improve the correlation with subjective rating [22], there is, up to now, still no accepted standard. In this thesis, the normalized mean square error (NMSE) between the original image X and coded image Y, is adopted as the distortion measure because it is analytically tractable, computable in real-time and also subjectively relevant. It should be emphasized that even though the NMSE does not completely match with the subjective rating, small and large NMSE do correspond to good and bad subjective qualities, respectively [22].

2.3 VECTOR QUANTIZATION

Vector quantization (VQ) concerns the joint quantization of a block of input data. In VQ [1], the input vector, $V = \{V_i\}$, drawn from an L-dimensional Euclidian space is mapped into a finite set (codebook) of reproduction vectors (codewords), $C = \{C_p,\ p = 1, 2, ..., N\}$, contained in the space. In other words, the input vector, V, is represented by C, the vector quantized version of V. This can be written as,

$$q(V) = C \qquad (2.16)$$

where $q(\bullet)$ is the quantization operation.

In designing a codebook with N levels (codewords), the input vector space is partitioned into N Voronoi regions $\{\Pi_p:\ p = 1, 2, ...N\}$, where each, Π_p, is associated with a representative vector, or codeword, C_p. The quantizer then assigns, C_p, if and only if the input, V, is in, Π_p,

$$q(V) = C_p \qquad \text{iff } V \in \Pi_p. \tag{2.17}$$

An optimal quantizer should minimize the average coding distortion over all other N-level quantizers. If the distortion is defined as follows,

$$D = E[\, d(V, C)\,] = \sum_{i=1}^{N} P(V \in \Pi_p)\, E[\, d(V, C_p)\,] \mid V \in \Pi_p\,] \tag{2.18}$$

then the optimal quantizer must satisfy two necessary conditions [29]:

1) The optimal quantizer must use a minimum-distortion or nearest neighbor selection rule,

$$q(V_i) = C_p, \qquad \text{iff } d(V_i, C_p) \leq d(V_i, C_n) \text{ for all } p \neq n \tag{2.19}$$

where $q(\cdot)$ is the quantization operator. In other words, the quantizer chooses the code vector that results in the minimum distortion with respect to V.

2) The codewords, $\{C_p\}$, must be chosen to minimize the average distortion in each cell, Π_p,

$$D_p = E[\, d(V, C) \mid V \in \Pi_p\,], \qquad p = 1, 2, \ldots\ldots, N \tag{2.20}$$

In other words, $\{C_p\}$ are simply the centroids of the corresponding Voronoi regions.

The basic principle of VQ is shown in Fig. 2.2. The steps involved in VQ as applied to image coding are vector formation, training set generation, codebook design, quantization and decoding.

2.3.1 VECTOR FORMATION

Vector formation is the first step in vector quantization. For an image, it typically consists of a blocking operation, where the input data is organized into small blocks of pixels followed by a preprocessing operation. The features or values are then extracted from the blocks of pixels and then formed into vectors; in other words, the image is decomposed into a set of vectors. Many different decompositions have been proposed; examples include, the intensity values of a spatially contiguous block of pixels[37,38]; intensity values normalized by the mean and variance[39]; colour components of a pixel[40]; the transformed components of a block of pixels[41,42] and adaptive linear predictive coding(LPC) coefficients for a block of pixels[43]. One decomposition of a 256 x 256 image into vectors of dimension 16 is illustrated in Fig. 2.3.

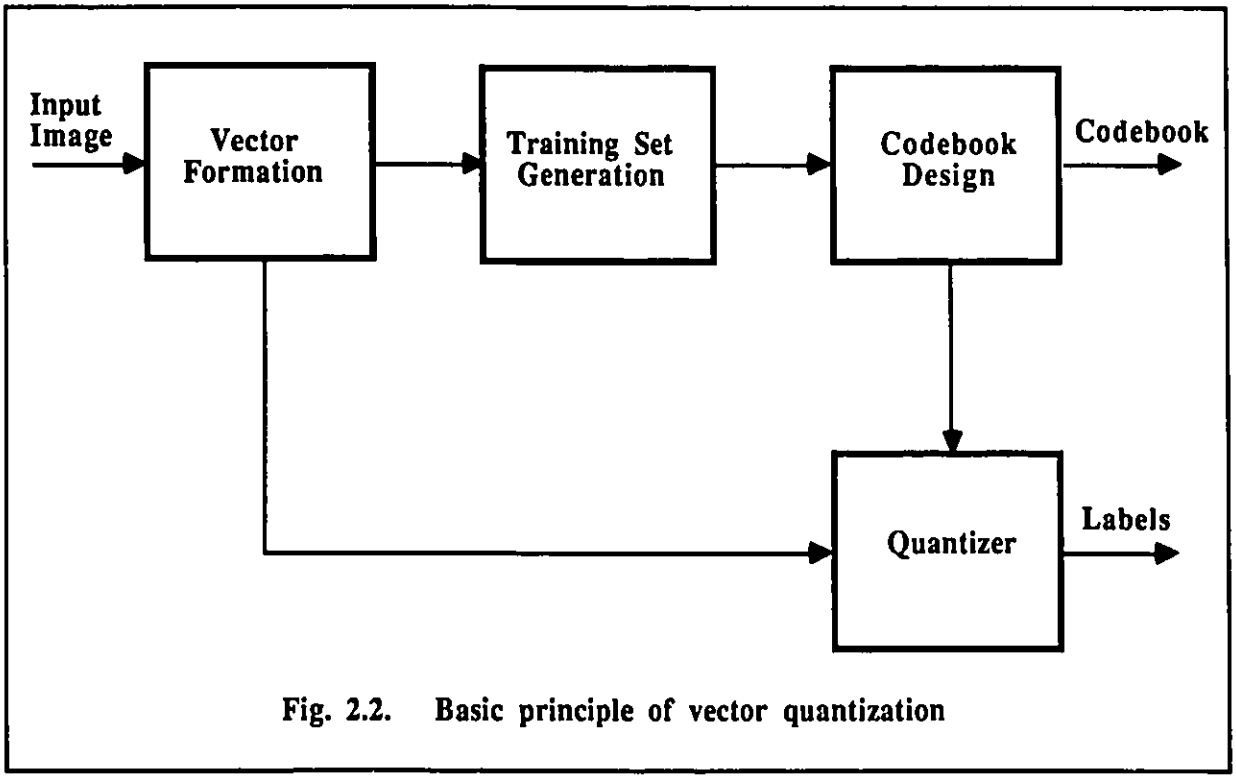

Fig. 2.2. Basic principle of vector quantization

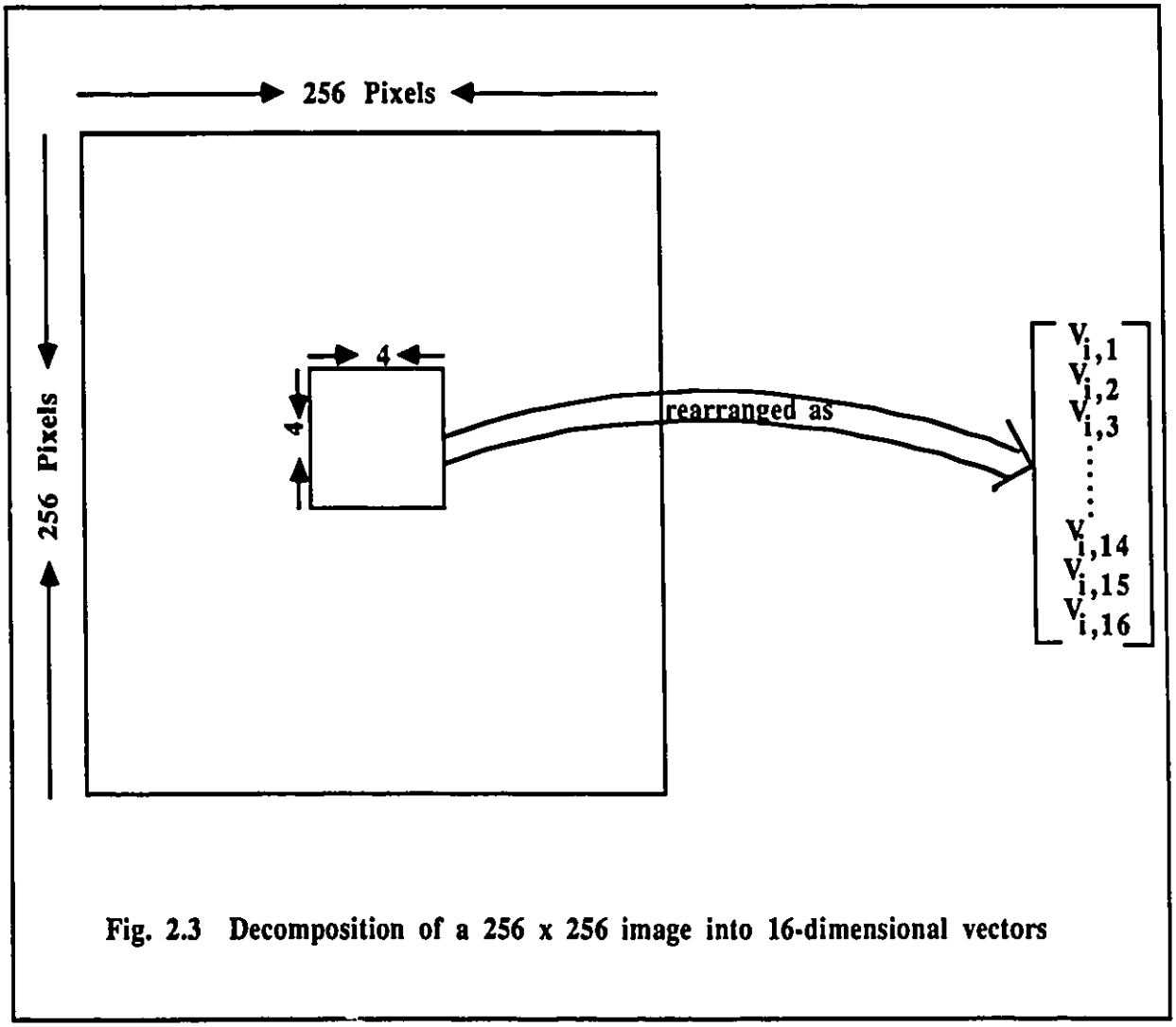

Fig. 2.3 Decomposition of a 256 x 256 image into 16-dimensional vectors

2.3.2 TRAINING SET GENERATION

This step involves the selection of a set of vectors from a cross-section of representative images, or from the image to be coded. An optimal vector quantizer should ideally match the statistics of the input source [5]. We now discuss two methods of training sequence generation: universal and adaptive. In the universal method (VQUC) of [37,44], the vectors for the training sequence are drawn from many different kinds of images. The resultant universal codebook can then be fixed at the transmitter and the receiver and therefore, there is no overhead required for transmitting the codebook. There are two problems in the universal method. First of all, to ensure a good fit for all images, a large codebook is needed, thus, increasing the bit rate. Secondly, images outside the training sequence may not always be represented well.

The first problem is overcome by reducing the codebook size using techniques that exploit the local image statistics. Ramamurthi and Gersho[45] classify the vector source into edge and shade vectors and separate training sets are generated. They call this approach a classified vector quantizer (CVQ). Since the consecutive vectors are statistically dependent, better performance can be achieved if inter-vector correlation is incorporated into the encoder. This technique results in a codebook of smaller size by classifying the input vectors. Aravind and Gersho[46] propose the use of a state transition function F to predict the state of each input vector from the previous reproduction vectors. The state variable, s, identifies which sub-codebook in the codec's supercodebook to use to encode the input vector. As the sub-codebooks are smaller in size less bits are needed for the labels; furthermore, no side information is required for indicating the choice of the codebook. Cuperman and Gersho[47] developed a predictive vector quantizer to exploit the inter-vector correlation where the coder consists of a vector predictor and an error vector quantizer. An error vector signal is formed by subtracting the predicted vector (estimated from the previous vectors) from the present vector and is then coded using the error codebook. This technique also results in reduced size codebooks.

The second problem can be overcome by using adaptive techniques which track the local statistics from image to image[6]. One such scheme is image adaptive codebook generation (VQIAC) technique presented by Goldberg and Sun [42] in which the training sequence and the codebook are adapted to each input image. For each input image, a subset of vectors formed from

the image is taken as the training sequence and a codebook is generated. One further refinement is to break up the input image into subimages, and design a codebook for each subimage [6]. The local codebooks better match the local statistics, thus, a smaller codebook can be used. However, an overhead is required for transmitting the codebook for each input image. Gersho and Yano [48] have presented a technique in which the adaptation involves changing only a small part of the codebook rather than the entire codebook. The algorithm monitors the distortion for each input vector and if it is larger than a predetermined threshold, then the input vector is added to the codebook as a new vector. Yeh [49,50] has presented a codeword replacement scheme for both still and image sequences. This scheme adaptively replaces the codewords that cause excess quantization error due to mismatch. Sun and Goldberg[42] have proposed a replenishment technique for image sequences; here, the changes in the successive frames are tracked by replenishing the labels and the codebook.

2.3.3 CODEBOOK DESIGN

A procedure which has been applied successfully to codebook design is an iterative clustering algorithm, referred to as the LBG algorithm (Linde, Buzo and Gray [5]), or the generalized Lloyd algorithm [33]. This algorithm produces a quantizer such that the two necessary conditions of optimality are satisfied. The LBG algorithm generates in an iterative manner a locally optimum codebook of size, N, from a given initial codebook, refining the codebook at each iteration until the change in distortion is within acceptable limits. One important feature of the LBG algorithm is that the codebook can be designed without requiring explicit knowledge of the source statistics. Furthermore, this algorithm has been shown to converge to a local optimum [5]. The steps involved in this algorithm are as follows,

Step 0 : ***Initialization :*** Given a distortion threshold $\varepsilon \geq 0$, an initial codebook, $C_0=\{C_{p,0}; p = 1,2,....N\}$, and a training sequence of input vectors, $\{V_i, i = 1,2,....,T\}$. Set m=0 and $D_{-1} = \infty$.

Step 1 : ***Partition:*** Each training vector is assigned to a Voronoi region [1] using the nearest neighbour rule,

$$V_i \in \Pi_{p,m}, \quad \text{iff} \quad d(V_i, C_{p,m}) \leq d(V_i, C_{n,m}) \quad \text{for all } p \neq n. \qquad (2.21)$$

Step 2 : ***Codebook Updating:*** The codewords of the (m + 1)st codebook are formed,

$$C_{p,m+1} = \frac{1}{T_{p,m}} \sum_{V \in \Pi_{p,m}} V, \qquad p = 1, 2, ..., N. \qquad (2.22)$$

Here, $T_{p,m}$, is the number of vectors in, $\Pi_{p,m}$, and the new codewords are the centroids of the corresponding Voronoi region.

Step 3 : ***Termination:*** Set m=m+1, and compute the average distortion

$$D_m = \frac{1}{T} \sum_{i=1}^{T} \min_{C \in C_m} d(V_i, C) \qquad (2.23)$$

If $(D_{m-1} - D_m)/D_m \leq \varepsilon$, stop and let, C_m, be the final codebook. Otherwise, return to step 1. In practice, we have found that as many as 100-200 iterations may be required to generate a "locally" optimum codebook. One of the shortcomings of the LBG algorithm is that it can be easily trapped in a local minima of the distortion resulting in a suboptimal codebook. A stochastic optimization technique called simulated annealing [51-53] can be used to avoid local minimums by introducing a degree of randomness in the search algorithm.

The locally optimum codebook generated using the LBG algorithm is not unique [3], but depends upon the choice of the initial codebook. There are two basic approaches to obtain the initial codebook [1]:

1) The first approach is to start with a codebook of the required size. An example is to take the first N vectors in the training sequence as the initial codewords [1]. A modification is to select suitably spaced vectors from the training vectors [1].

2) The second approach is to start with a small codebook and recursively construct a larger one; an example is the *binary splitting* method [5]. In this method, the starting point is one codeword, the mean vector of the training vectors. The codeword is split by a fixed perturbation vector to form two codewords and the LBG algorithm is applied to yield a codebook of two levels. Each of the two codewords are then split and the LBG algorithm is again applied to produce a codebook with four levels. The procedure continues until the required number of codewords is generated.

2.3.4 QUANTIZATION

During this step, for each input vector, V_i, the closest codeword, C_p, is found using equation 2.19. The corresponding index (label) of the codevector is transmitted to the receiver.

2.3.5 DECODING

This is a simple table look-up procedure, where the label is used as the address to the appropriate codeword as shown in Fig. 2.4.

2.4 BIT RATE CALCULATION

We now turn to the computation of the bit rate. In the case of VQUC, only the label is transmitted so that,

$$R_t = R_l = (\mathrm{Log}_2 N) / L \qquad (2.24)$$

where, N, is the number of codewords in the universal codebook, and, L, is the dimension of the vector. In the case of VQIAC, there is an additional overhead for transmitting the codebook, and hence the total bit rate R_t, is the sum of the two individual components R_c and R_l, for codebook transmission and label transmission, respectively. Thus,

$$R_t = R_c + R_l \qquad (2.25)$$

where,

$$R_c = (N * B) / K \qquad (2.26)$$

and R_l is given by Eqn. 2.24. Here, B, is the number of bits per representative vector component and K, is the number of vectors in the image.

2.5 COMPUTATIONAL COMPLEXITY

2.5.1 BASIC DISTORTION OPERATION

The basic computation in a VQ encoder is the distance calculation between an input vector and a codeword (Eqn. 2.18). In other words, it is the distortion that would be introduced by replacing

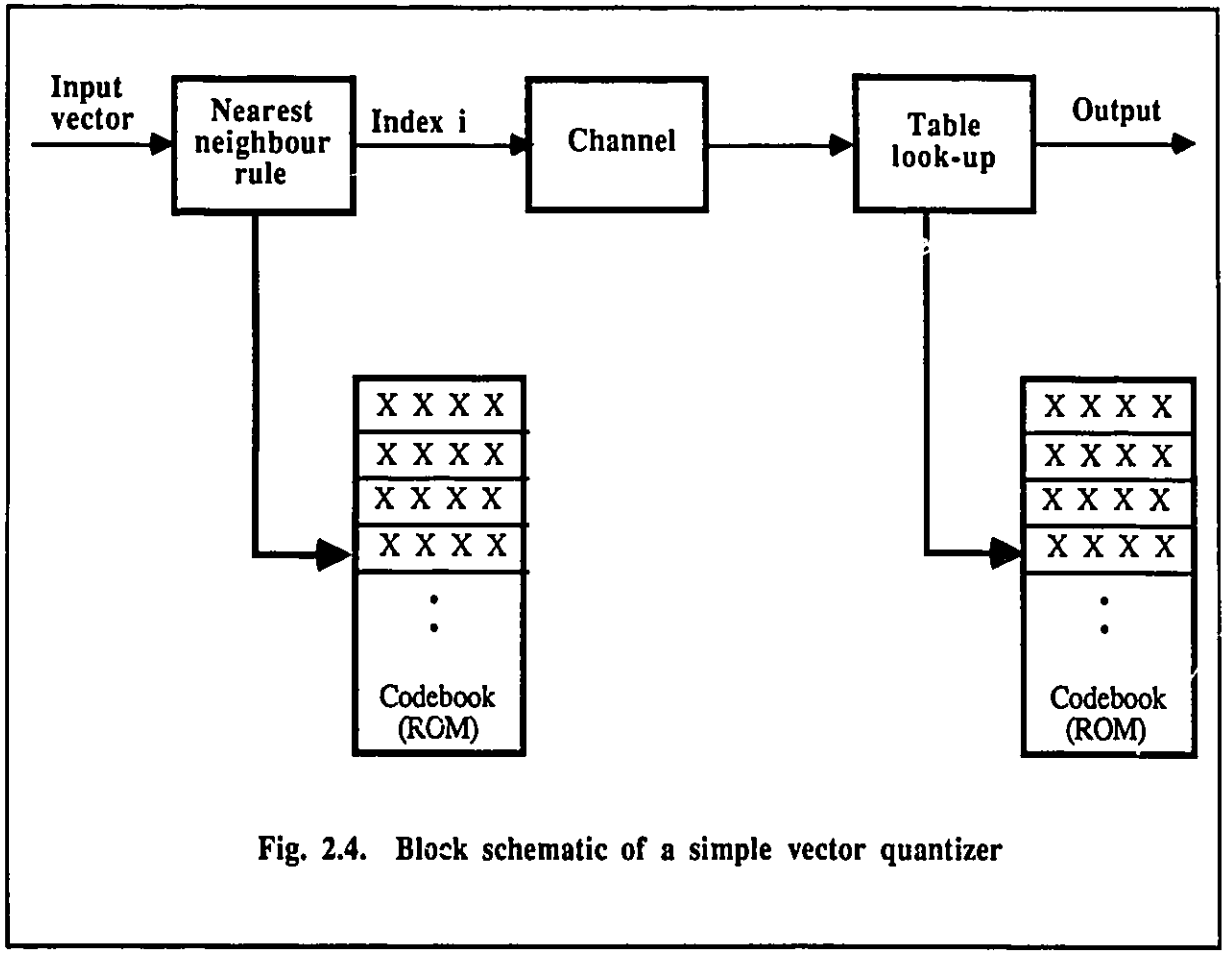

Fig. 2.4. Block schematic of a simple vector quantizer

the input vector with the codeword. If the squared error measure is used,

$$d(V_i, C_p) = \frac{1}{L} \sum_{j=1}^{L} [(V_{i,j} - C_{p,j})^2] \qquad (2.27)$$

this requires the computation of L basic distortion operations. Here, we define the basic distortion operation as the computation of the square of the difference between an element of the input vector and the corresponding element of the codeword.

2.5.2 ENCODING COMPLEXITY

As a baseline for comparison purposes, we consider a full search N-level VQ encoder operating sequentially, i.e., one input vector and one codeword at a time. The pseudo code is as follows.

```
PROCEDURE VQ_SEQUENTIAL (In: Input_vectors, Codebook; Out: Labels )
  BEGIN
    DO ( For Input_vectors 1,..., K )
      DO ( For codewords 1,..., N )
        DO ( For elements of the vector 1, ..., L )
          Compute the distortion
        END DO
      END DO
      Label of the Input_vector = Index of the closest (least distortion) codeword
    END DO
  END
END PROCEDURE
```

The encoding complexity is O(KLN) = K*L*N basic operations. For example, a 256 x 256 image with a vector dimension of 16 encoded using a codebook of size 256 (bit rate of 0.5 bits/pixel) requires approximately 16 million operations. The corresponding value for 512 x 512 and 1024 x 1024 images are 64 and 256 million operations, respectively. We use the implementation on an Single Instruction Single Data (SISD) machine as a baseline for comparison. This machine has an unity speed-up $S_p(1)$, as the operations are performed sequentially.

2.5.3 CODEBOOK GENERATION COMPLEXITY

For each iteration in the codebook generation, there are two steps: an encoding process (Eqn. 2.21) of complexity O(KLN) and a centroid computation process (Eqn. 2.22) of complexity O(KL) = K*L accumulation operations. As the centroid computation complexity is much smaller compared with the encoding complexity, it is ignored in the analysis below. If the number of iterations of the codebook generation algorithm is I, then the computation complexity for codebook design is O(KLNI).

2.6 METHODS FOR REDUCING COMPLEXITY

Although vector quantization can achieve significant performance advantage over scalar quantization, the advantage is obtained at the expense of considerable computational and storage costs. For full search vector quantization, the computational complexity is exponentially proportional to the dimensionality of the vectors and the number of bits per dimension. Several techniques have been proposed to reduce the computational complexity in a full search vector quantization.

2.6.1 TREE SEARCH VECTOR QUANTIZATION

Instead of using a full search, quantization can be operated on a binary tree codebook [54]. In binary tree-structured VQ (Fig. 2.4), the M-dimensional space is first divided into two regions, then each region is further divided into two subregions, and so on, until the space is divided into N regions, or cells. In quantizing each input vector, a decision is made between one of the two subregions at each layer of the binary tree and the representative codeword is found at the last layer. In other words, an input vector is quantized by traversing (searching) the tree along a path that gives the minimum distortion at each node in the path. The computational complexity is $O\{KL\ (\log_2 N)\}$.

2.6.2 MULTISTAGE VECTOR QUANTIZATION

Multistage vector quantization was first proposed by Juang [55]. Fig. 2.5 shows a two-stage example of multistage vector quantization. The input vector X is first vector quantized using a

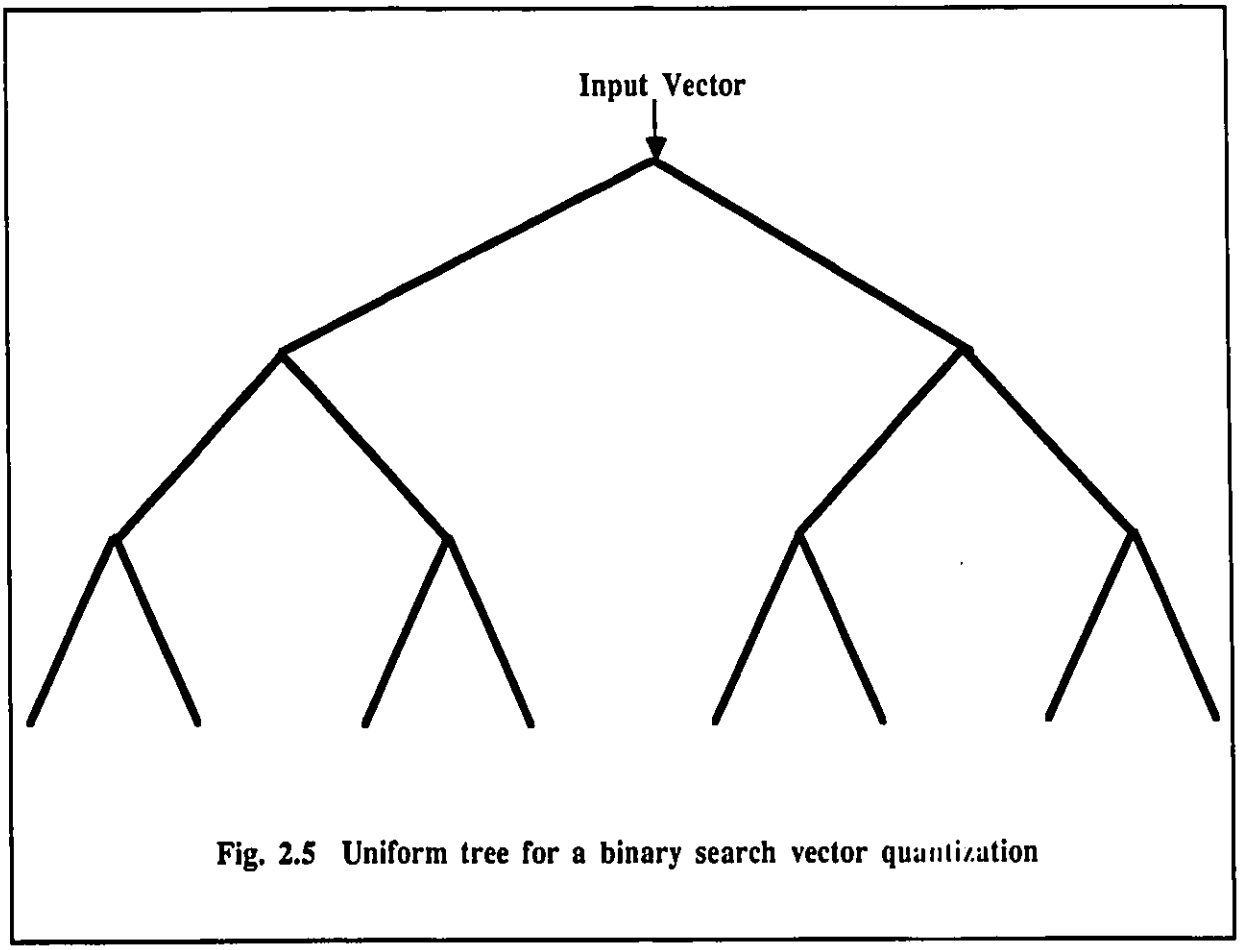

Fig. 2.5 Uniform tree for a binary search vector quantization

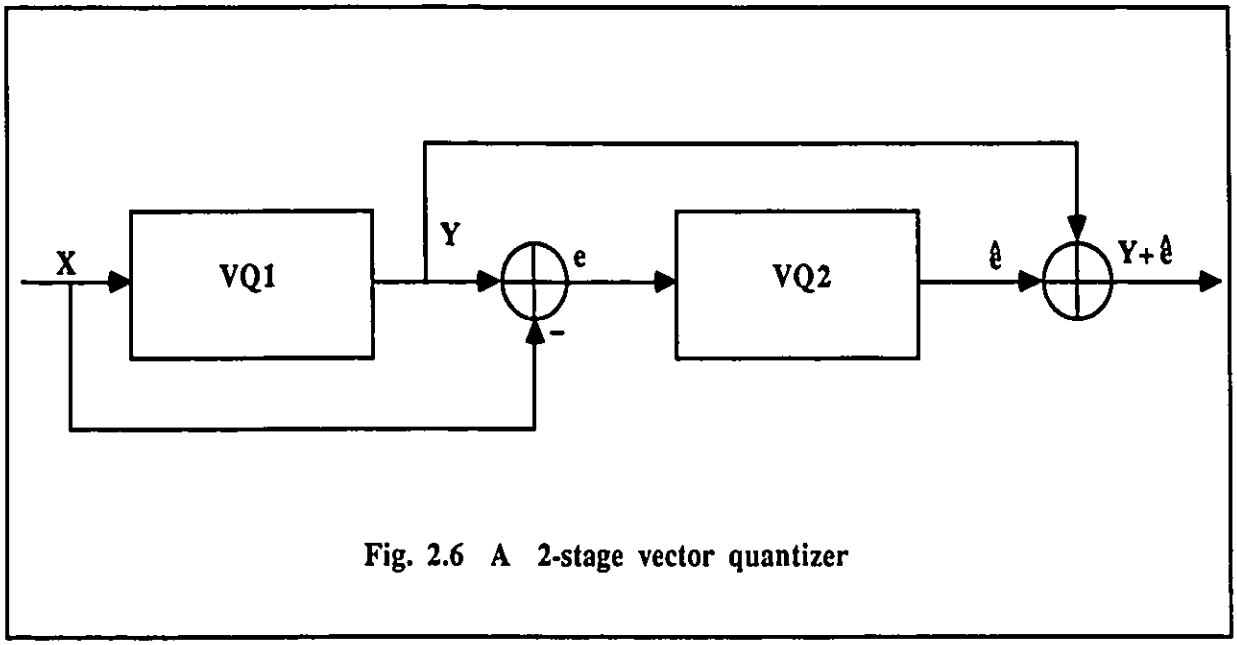

Fig. 2.6 A 2-stage vector quantizer

vector quantizer with N_1 levels. The residual error between the input X and its vector quantized version Y is then used as the input to a second stage vector quantizer with N_2 levels. The final quantized version of the input X is simply the sum of the outputs of two stages. The process can be repeated by feeding the residual errors from the second stage into the third stage, and so on. The computational complexity is proportional to $O(KL\sum N_i)$ instead of $O(KL\prod N_i)$ for a single stage vector quantization, where the index i refers to the number of stages.

2.6.3 PRODUCT VECTOR QUANTIZATION

In product vector quantization [54,56,57], the Euclidian space of the input vector is first decomposed into two or more subsets, jointly representing all points in the space. If we let L_i and N_i be the dimensionality and codebook size of subset i, then the dimensionality and codebook size of product code are, respectively, given by,

$$L = \sum L_i \qquad (2.28)$$

$$N = \prod N_i \qquad (2.29)$$

If the subsets are separately vector quantized with their respective codebooks, the corresponding computational complexity is $O(K\sum N_i \times L_i)$. The corresponding computational complexity for a single stage vector quantization with vector dimension L and codebook size N is $O(KLN) = O(K\sum L_i \prod N_i)$.

An example of a product vector quantization is the Gain/Shape vector quantization [56]. In this technique, the input vector is first normalized, prior to encoding by the removal of the gain defined as energy of the vector. The resultant normalized input vector is referred to as the shape vector. The gain and shape are then separately quantized; in particular, the gain is scalar quantized and the shape vector quantized. Another example of product vector quantization is the Mean/Shape vector quantization [57]. The mean of each vector is subtracted to yield a new shape vector with zero mean. The shape vector is then vector quantized while the mean is separately scalar quantized.

2.7 SUMMARY

In this chapter, we first reviewed image coding. Lossless coding techniques such as, Huffman and run length coding were then discussed followed by the lossy coding techniques such as, predictive coding, transform coding and hybrid coding. Secondly,the concept of rate distortion function was reviewed. A review of vector quantization as applied to image coding was then presented. This was followed by the bit rate calculations and an analysis of computational complexity for vector quantization. Finally, the methods to reduce the complexity of vector quantization were discussed. Two new adaptive techniques for vector quantization at a reduced complexity will be presented in chapter 5. The first technique uses a subset of the universal codebook (VQRUC) while the second technique is basically a one pass algorithm on this reduced codebook (VQRUC-1). Both these techniques result in an improved coding performance over the universal codebook at a substantially reduced computational complexity. A super adaptive algorithm based on a mini-max error criterion algorithm will be presented in chapter 6. This algorithm has the advantage that the codebook is generated on the fly and is adaptive. It is easily implementable using a stack based architecture.

3. REVIEW OF ARCHITECTURES

The image processing community has always bemoaned the inevitably high demands their work makes on the computing resources. Images contain vast amounts of data and seem to require highly complex (and therefore computationally expensive) analytical programs. These demands, coupled with the rapidly increasing availability of cheap computer hardware, have led to a proliferation of novel processor architectures.

In this chapter, a review of different architectural concepts applicable to image processing and pattern recognition applications is presented. First of all, the need for architectures is discussed in section 3.1. This is followed by the classification of the architectures in section 3.2. The methods to speed-up image processing algorithms are then discussed in section 3.3. This follows with a discussion on special purpose architectures such as systolic arrays, pyramid architectures, content-addressable memories, and neural networks in section 3.4. Finally, a summary is presented in section 3.5.

3.1 NEED FOR ARCHITECTURES

The performance of conventional computers in the rapidly expanding field of image processing (including image analysis) has been found to be quite inadequate for the majority of the applications which have been attempted or envisaged [58]. The typical data rates involved in processing a television sequence image of size 512 x 512 sampled at 8 bits/pixel with a frame rate of 30 frames/second is of the order of 60 Million bits/second. In practice, even very modest image analysis tasks will often require some hundreds of the so-called local neighbourhood operations which implies a computation rate of 6 Billion operations/second.

Faced with these demands for impossibly high computation rates, the image processing community has always been driven to explore ways and means of achieving greatly improved performance. In general, it has been realized that either the processors must become more powerful, or else, systems must be designed in which many processors can be assembled into a single, powerful, composite structure. The complete solution involves the design of dedicated

architectures onto which are mapped the algorithms. In the following section, the different classes of architectures applicable for image processing operations are discussed.

3.2 CLASSIFICATION OF ARCHITECTURES

The architectures for image processing applications can be classified into one of the following categories: (i) serial, parallel, and pipeline architectures, (ii) array processors, and (iii) vector processors.

3.2.1 SERIAL, PARALLEL, AND PIPELINE ARCHITECTURES

Serial architectures:

Some of the image processing operations are intrinsically serial in the sense that the order in which the data are processed is important. Examples include, searching, measurement and classification. Searching may involve enumerating objects in a scene or finding the exact location of some feature starting from a knowledge of its approximate location. Classification of a whole image or of objects within an image may involve a series of measurements and a sequence of decisions.

Serial systems generally offer more flexibility in total system cost particularly allowing small cheap systems to be developed more easily [58]. It is also easier to develop high level language software tools which map efficiently onto the hardware because of the less committed nature of a serial machine. It also provides a more efficient mapping of software to hardware. The main drawback is speed and is a potential weakness of a serial system when image to image transformation is a major part of an applications program.

Parallel architectures :

There are some image processing operations which are intrinsically parallel in the sense that the same rule must be applied to each of many data, and the order in which the data are processed does not matter. Examples include, single pixel operations such as contrast manipulation or thresholding and neighbourhood operations such as convolution, erosion / dilation and non-linear filtering. These algorithms are important because, although the basic algorithm may be simple and the image transformation involved may be a small step in the complete solution, very large amounts

of data are involved.

Parallel systems exploit the spatial parallelism in the image. Examples include: pixel-bit parallelism, neighbourhood parallelism, image parallelism, and operator parallelism. The inherent advantage of a parallel machine is the speed-up achievable, which implies a high throughput resulting in real-time processing, however, at the cost of increased amount of processors.

Pipeline processors :

Pipelining refers to temporal parallelism and in the context of image processing usually implies concatenation of a number of operators performing logical and arithmetic operations on a small neighbourhood. Although a pipeline processor offers the advantage of parallelism, it has the inherent latencies involved in computing the result; in other words, the first output is available after N clock cycles where, N is the number of processors in the pipeline, while the subsequent outputs are available at intervals of one clock cycle.

3.2.2 ARRAY PROCESSORS

Array processors are a set of processors executing the same set of instructions driven by a host computer. The processing cells are connected in parallel and pipeline as shown in Fig.3.1. This belongs to the class of Single Instruction Multiple Data (SIMD) processors. The processing cells operate in a synchronized fashion under the control of a common clock (in the lock-step mode). Typically, data (images) are loaded into the array along one edge of the array and propagated along the interconnections until the data set is in place, at which time processing commences. Because of the large number of interconnections and processing elements, processor implementation is usually bit-serial. In the limit, an array architecture contains one processing element per pixel [59]. The main advantage offered by the array architectures is the ability to apply massive parallelism to image computation . On the other hand, they have the same drawbacks as concerns real-time applications, namely that they cannot begin processing until the array is loaded, and cannot extract information from the processed image until it is unloaded.

3.2.3 VECTOR PROCESSORS

Vector processors are often known as "array processors" although they are simply "processors

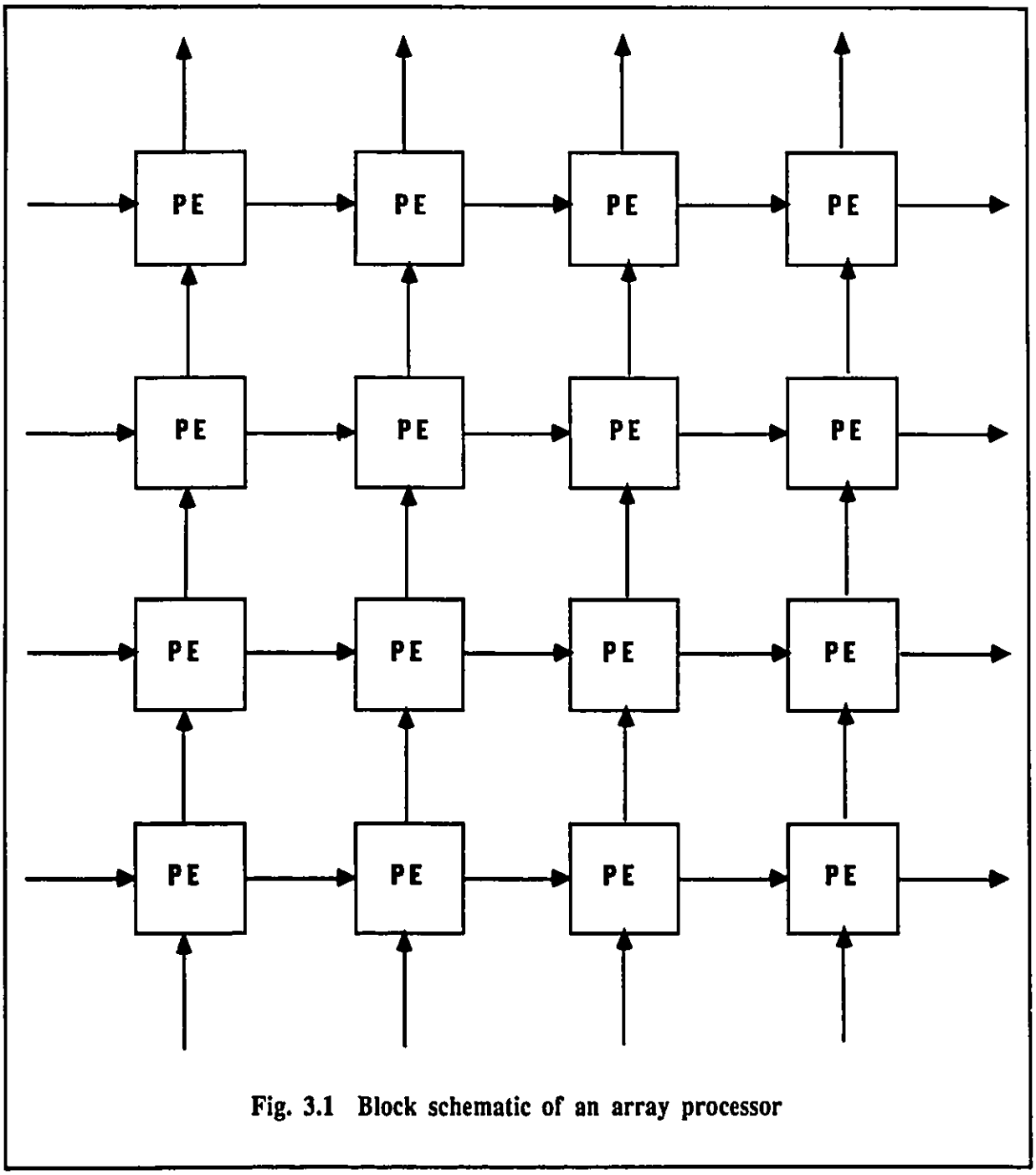

Fig. 3.1 Block schematic of an array processor

of arrays" rather than simply SIMD "arrays of processors". They provide the host computer with the ability to carry out high speed vector operations such as convolution and vector multiplication and thus operate on vector instructions. Vector processors need to perform the same operation on different data sets repeatedly. One obvious advantage of vector processing over scalar processing is the elimination of the overhead caused by the loop control mechanism.

These processors are often described as horizontal architectures since a number of parallel resources, such as adders and multipliers, are controlled by individual fields in a horizontal microcode instruction. Since identical processes (or functions) are repeatedly invoked many times, each of which can be subdivided into sub-processes (or subfunctions), vector processors have pipeline structures. For image processing applications, the data elements are usually arranged in array, vector, or matrix forms and computations on the data could be done using a vector processor.

3.3 FAST TECHNOLOGY AND NEW DEVICES

The speed-up factors provided by all of the above architectures is still not enough to satisfy the real-time requirement for most image processing algorithms. One method of achieving further gains in speed-up (smaller cycle time) is to use fast technology and new devices. The use of faster technology is an interesting point, since technology is evidently progressing rather rapidly [60]. However, it is moving mainly towards greater levels of integration which is roughly quadruple every two years, especially in memory design and processor technology. The clock rates of the processors have increased from 1 Mhz to 50 Mhz and the memory cycle times have decreased from 500 ns to 20 ns in the last 10 years. TTL, MOS and ECL are approaching their limits while GaAs and Josephson devices are due to appear by the end of the decade. The research in optical circuits and surface acoustic wave devices seem to be promising. In spite of these developments, we find that it takes considerable effort and expense to achieve 5-10 times speed-up.

There is a certain mileage to be gained from using more advanced technologies, but there are strict limits to what is achievable. By and large it therefore seems that one has to look to the degree of hardware parallelism, and to the specific architecture to be used for major gains in speed of image processing algorithms. In other words, the solution could be considered as one of designing dedicated architectures which map the algorithms to structures optimally. The advent of

VLSI has made possible integration of several thousand devices on a single chip, thus justifying the design of special purpose architectures even though they are complicated, and involve a large number of processors. In the following section, some of the novel architectural concepts will be reviewed.

3.4 NOVEL ARCHITECTURES FOR IMAGE PROCESSING

This section summarizes some of the latest developments in image processing architectures. These architectures efficiently map algorithms resulting in significant speed-up factors which in turn makes possible real-time implementation. First of all, a brief description of a special purpose parallel and pipelined architecture called systolic arrays is presented. This will be followed by a brief description of a multi-layered structure known as pyramid architectures. The concept of content-addressable memories and its application to pattern matching is then discussed. Finally, an overview of neural networks is presented. Note that emphasis is placed on systolic array and content addressable memory concepts because of their potential applications for implementation of vector quantization which will be discussed in detail in chapters 7 and 8.

3.4.1. SYSTOLIC ARRAYS

Computational tasks can be conceptually classified into two families: (i) compute-bound, and (ii) I/O-bound computations. In a computation, if the total number of operations is larger than the total number of input and output elements, then the computation is compute-bound, otherwise it is I/O bound. Any attempt to speed-up an I/O-bound computation must rely on an increase in memory bandwidth. A balance between the two kinds of computations is critical in image processing applications [60]. Systolic architectures provide an elegant solution to achieve this balance. They belong to the class of array processors which are parallel and pipelined.

The systolic architectural concept developed by H.T.Kung *et al.*, [61] is a general methodology for mapping high-level computations into hardware structures. In a systolic system, data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory. The crux of the systolic array approach is to ensure that once

a data item is brought out from the system memory, it can be used effectively at each processing cell it passes. This basic principle of a systolic architecture is illustrated in Fig. 3.2. By replacing a single processing element (PE) with an array of PE's, a higher throughput is achieved without increasing the memory bandwidth. In the systolic system, the data flow may be at multiple speeds in multiple directions as against the classical pipelined system with unidirectional data flow, making possible the implementation of a variety of computations [62]. Other advantages, include modular expandability, simple and regular data and control flows, use of simple and uniform cells, and elimination of global broadcasting and fan-in [61]. All these properties are highly desirable for VLSI implementations [62]. A systolic array based design to implement vector quantization will be detailed in chapter 7.

3.4.2 PYRAMID ARCHITECTURE

A "Pyramid" consists of a large array that forms its base, plus a series of successively smaller layers or regions, of arrays or other topologies [58]. A "complete" pyramid ends in the pyramid's single "apex" cell (which is a 1 x 1 array). Pyramids have unusually good global characteristics of passing messages between distant nodes(O[logn] as compared to O[n] for arrays). They are based upon and can be viewed as trees. Pyramids also have unusually good local structural characteristics for several important classes of problems, since the near-neighbour linkage of their array-grid structure is appropriate for handling the near-neighbour interactions in image processing and pattern recognition algorithms. They also have other desirable global features, that allow a programmer to code highly parallel, and hence very fast, programs that successively transform, abstract, combine, and take into account successively more global aspects of the scene. A typical pyramid that is being examined today for image processing applications is 3-dimensional, with a 2-dimensional array at its base and an apex that contains a single cell. The theory is that each level of processor can deal with the data at a higher level of abstraction, until the desired information can be extracted from the top-most processor. In this architecture, iterative algorithms can be implemented in which information flows up and down the pyramid. More specifically, pyramid architectures hold promise for implementing binary tree and quad tree based algorithms.

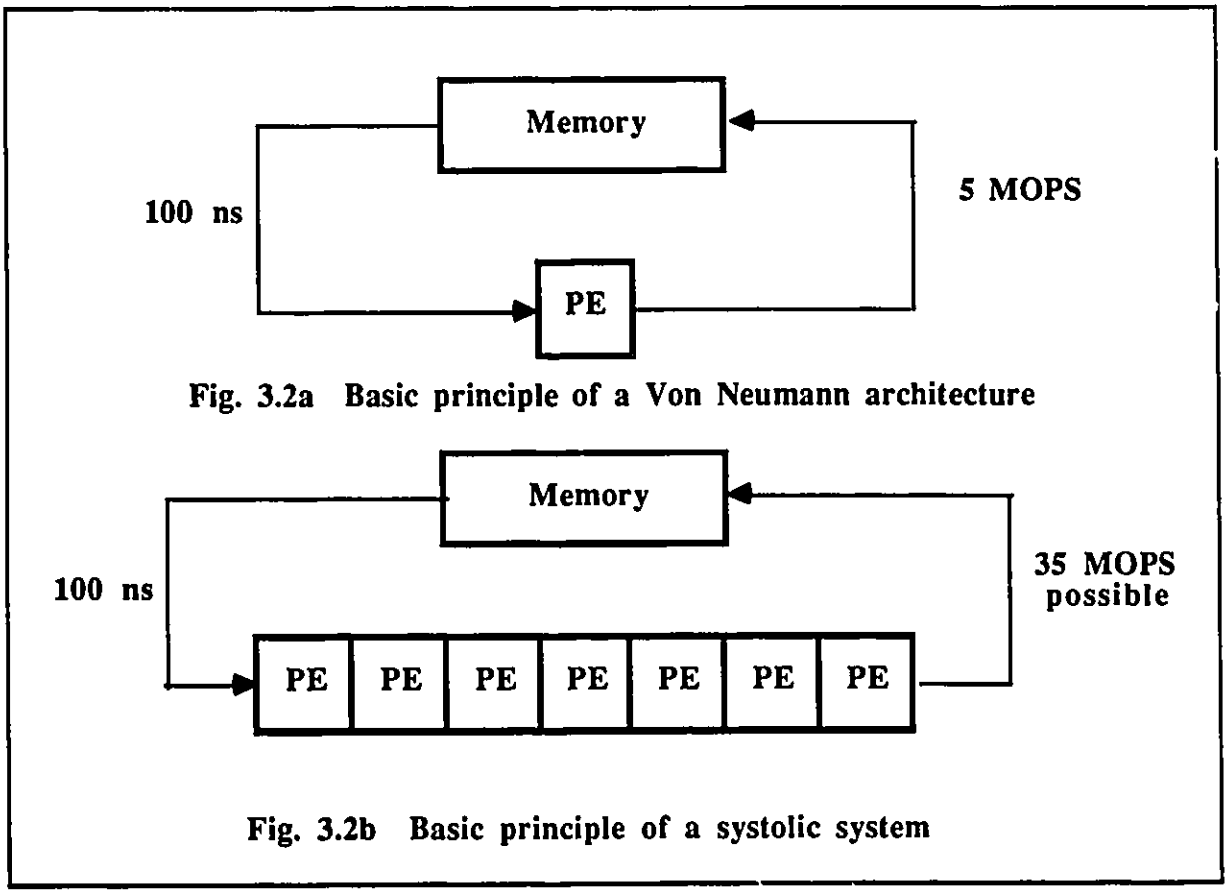

Fig. 3.2a Basic principle of a Von Neumann architecture

Fig. 3.2b Basic principle of a systolic system

3.4.3 CONTENT ADDRESSABLE MEMORY

Content-addressable memories (CAM's) are collections or assemblages of elements having data storage capabilities, which can be accessed simultaneously and in parallel on the basis of their data content rather than by the specific address or location[63]. The parallelism is proportional to the number of elements stored in the CAM. CAM's are particularly interesting as database machines, and can be effectively used for operations such as N-point FFT, template matching in pattern recognition etc., as they offer fast and concurrent access capabilities. The access mechanism is based on an external search argument which is compared with the data stored in all the cells. The block schematic of a CAM system is shown in Fig. 3.3. The main component is a CAM array, $\{B_{ij} : i = 1, 2, ..., n ; j = 1, 2, ..., b\}$ consisting of n x b cells organized as n words with b bits/word; $\{W_i : W_i = (B_{i1}, B_{i2}, ..., B_{ib}), i = 1, 2, ..., n\}$. Each bit cell, B_{ij}, consists of a flip-flop with associated logic gates for pattern matching and read/write control.

Bit cell : The circuit diagram of the basic unit of a CAM, the bit cell , is shown in Fig. 3.4 and is built with NAND gates N1 - N6 [64]. The bit cell effectively compares the input bit with the stored bit and reports the result as a match signal. The bit, B, is stored in the flip-flop (gates N3 and N4) and is entered through the input lines, I1(0) and I1(1). The input lines, I2(0) and I2(1), are used to compare the search bit with the stored bit value. The input line, A, serves to activate the bit cell. The match line, M, indicates the result of the comparison operation. The output of gate N3 represents the stored bit value. The gates N5 & N6 (having open-collector outputs capable of performing wired-AND logic) are connected to M. The details of writing/store and compare operations are as follows.

To write/store a bit and mask writing, I1(0) and I1(1) have the following values,

(i) I1(0) = 0, I1(1) = 1, and A =1 ; stored bit = 1,

(ii) I1(0) = 1, I1(1) = 0, and A =1 ; stored bit = 0,

(iii) I1(0) = 0, I1(1) = 0 ; writing is masked.

To perform comparison/content-addressable reading and masking, I2(0) and I2(1) have the following values,

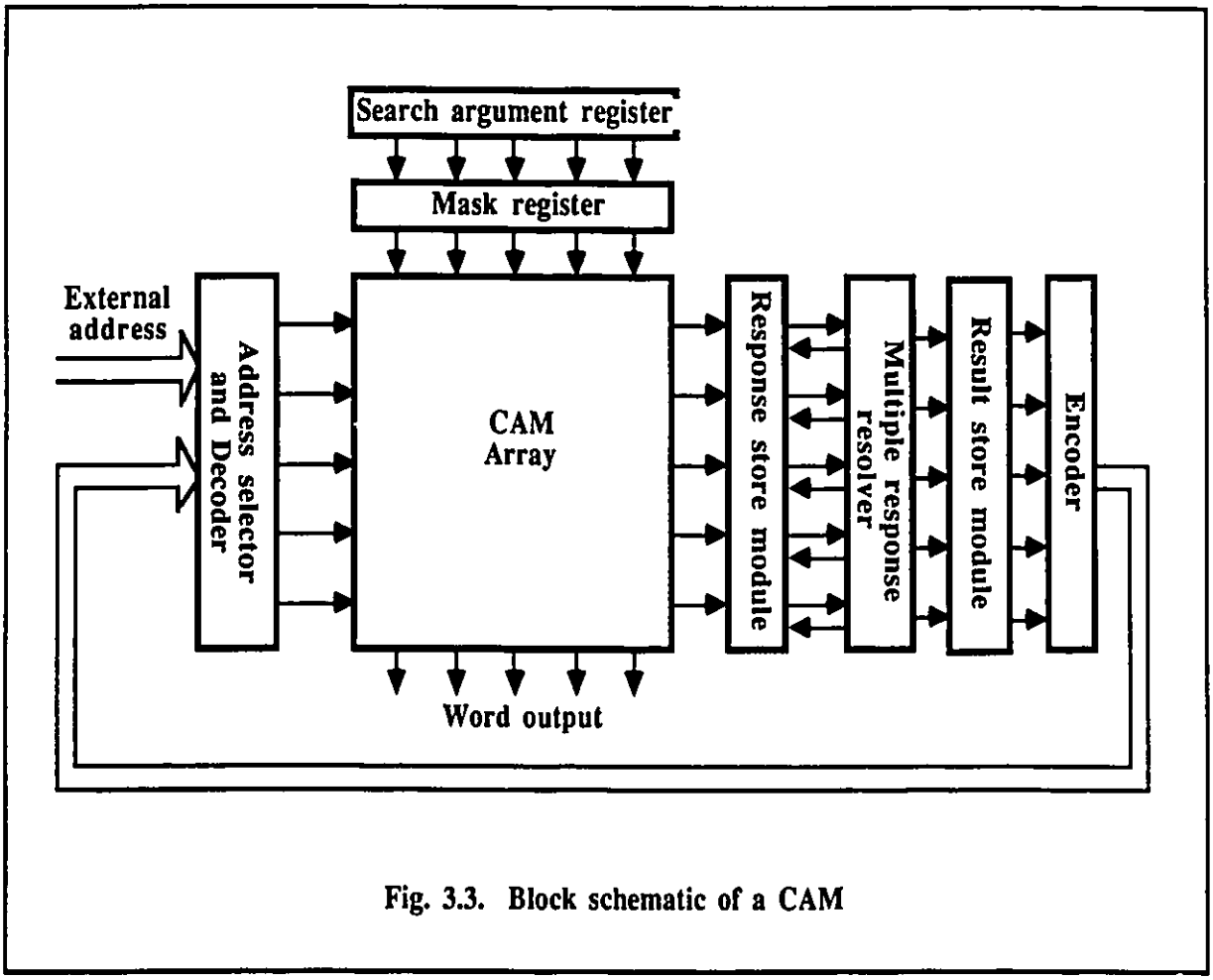

Fig. 3.3. Block schematic of a CAM

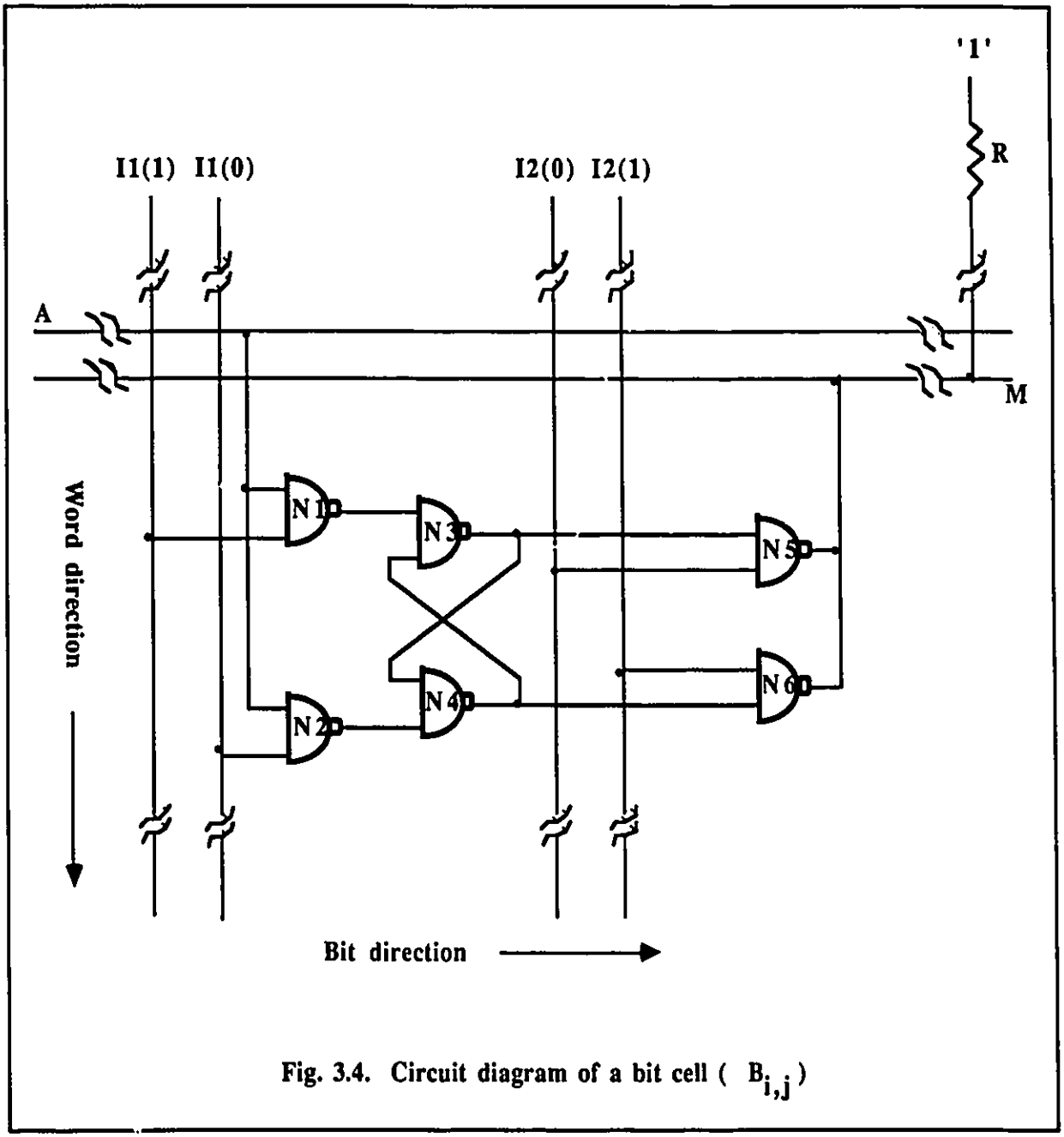

Fig. 3.4. Circuit diagram of a bit cell ($B_{i,j}$)

(i) I2(0) = 0[1] , I2(1) = 1[0] ; content-addressable reading is initiated with,

M = 1 if the stored bit was a 1[0], and

= 0 otherwise.

(ii) I2(0) = 0, I2(1) = 0 ; bit is masked for comparison.

The block schematic of a word cell is shown in Fig. 3.5. The match signal, M_i, and the address signal, A_i, are common to all bits in W_i. The word cell is used to compare a word with an external search argument stored in the search argument register, S. To facilitate searching on selected attributes of the search argument, the masking register, MR, is used to enable or disable a specific bit. Whenever, there is a match in all of the bit positions of W_i, then $M_i = 1$. The results of M_i are stored in the response register, R_i, in the response store module. In the event of multiple matches in a search operation, the multiple match resolver is used to order the outputs according to some priority. In addition to these basic blocks, the CAM has associated computing devices and additional circuits for interconnection and control. The encoder and the decoder serve to reduce the number of external and internal feedback lines.

CAM Array for Pattern Matching :

Traditionally, CAM arrays have been employed in the design of high speed pattern recognition machines[63-69]. The set of templates, T, are stored as words in the CAM array and the input pattern, C, is used as the search argument. The search argument is compared in parallel with all the words (templates) and if an exact match is obtained with a particular word, the corresponding output line is activated and the result recorded by the response register. This algorithm is expressed as follows.

PROCEDURE ***Exact_Match*** (**In:** Input_pattern, Templates; **Out:** Response_register)

Initialize all bits of Response_register, $R_i = 0$.

Initialize Search_arg. := Input_pattern

BEGIN

DO (For all Templates in parallel)

IF < Template = Search_arg. > **THEN**

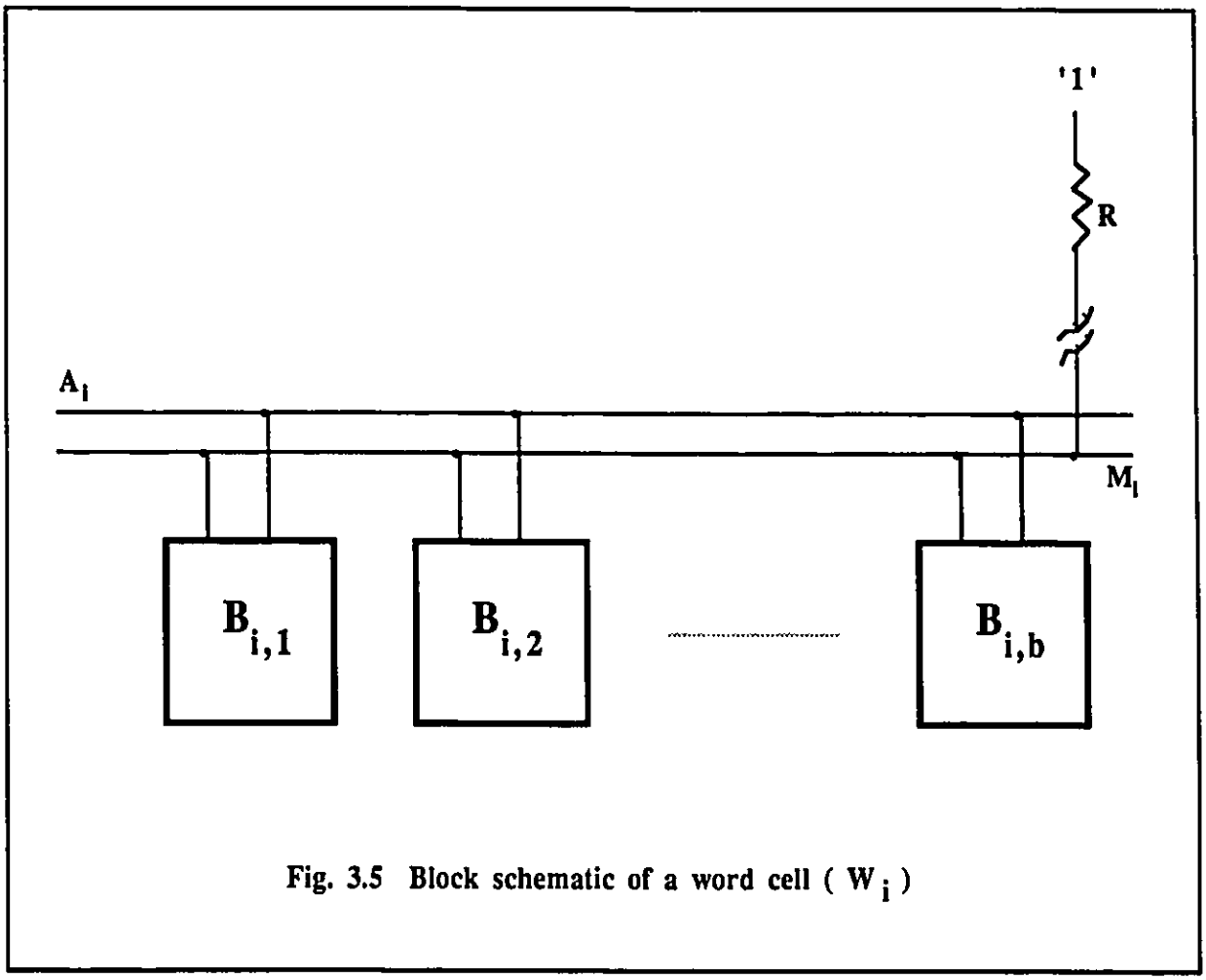

Fig. 3.5 Block schematic of a word cell (W_i)

Set corresponding bit of R_i to 1

END DO

END

END PROCEDURE

The output of the algorithm is recorded in the response register: a value of 1 indicates the templates that match the input_pattern.

Inexact Pattern Matching (" *within* Δ" search):

The CAM can also be used for inexact pattern matching applications[68, 69]. Here we now seek all words which are within a specified threshold Δ of the input pattern :

$$C - \Delta \leq T \leq C + \Delta \quad (14)$$

This is referred to as a *"within Δ"* operation and is executed by performing a "greater than" search followed by a "less than" search and computing the intersection of the results. Chu [69] has proposed an algorithm to perform the "greater than" ["less than"] search using the CAM. The algorithm is executed in b iterations (for a b-bit word) with the search argument being masked with different bit patterns stored in the mask register MR at each iteration. For completeness sake, we repeat the algorithm. To start with the response register is initialized to 0. The search argument is scanned from left to right and the first 0[1] is chosen as the target bit. This target bit is changed to a 1[0] and the bits to the right of the target bit are then masked to form a new search argument. An Exact_Match is sought with the new search argument. It can be observed that those words which have a 1 in the response register are certainly greater than the search argument, whereas nothing can be said about other words until the algorithm terminates. The target bit is now changed back to a 0[1] and the search argument is scanned further to the right for a target bit and the procedure is repeated. Since the results of the search operations for the individual passes have to be combined to get the final result, we require an intermediate response register, I_i, in addition to R_i, in the response store module. In pseudo code the algorithm can be written as follows.

PROCEDURE ***Greater[Less]_Than*** **(In:** Input_pattern,Templates; **Out:** Response_register)

Initialize all bits of Response_register, $R_i = 0$.

Initialize Search_arg. := Input_pattern

BEGIN

Scan the Search_arg. from left to right and choose the 0[1] as target bit.

REPEAT

Change target bit to 1[0] and mask all bits to the right to form a New_search_arg.;

Exact_Match **(In:** New_search_arg., Templates; **Out:** Response_register);

Change target bit to 0[1] and scan the Search_arg. further to the right for a 0[1].

UNTIL (all target bits are exhausted)

END

END PROCEDURE

The response register records a value of 1 in the bit position corresponding to templates which are Greater[Less]_than the input pattern.

The Inexact_Match procedure is now presented in pseudo code form:

PROCEDURE ***Inexact_Match*** **(In:** Input_pattern,Templates,Delta; **Out:** Response_register)

Initialize all bits of Response_register, $R_i = 1$;

Initialize all bits of Intermediate_response_register, $I_i = 0$.

BEGIN

Greater_Than **(In:** Input_pattern - Delta, Templates; **Out:**

Intermediate_response_register)

Copy Intermediate_response_register to Response_register { R_i := R_i **AND** I_i };

Initialize all bits of Intermediate_response_register to 0.

Less_Than **(In:** Input_pattern + Delta, Templates; **Out:**

Intermediate_response_register)

Combine results of the Greater_Than search and Less_Than search.{R_i := R_i **AND** I_i}

END

END PROCEDURE

The response register records a value of 1 in the bit positions corresponding to templates which are within the threshold (Delta) of the input pattern. A novel architecture based on a CAM to implement vector quantization in real-time is presented in chapter 8.

3.4.4 NEURAL NETWORKS

Conventional digital methods have run into a serious speed bottleneck due to their serial nature. To overcome this problem, a new computation model called "neural networks", has been proposed, which is based on some aspects of neurobiology and adapted to integrated circuits [70-74]. The key features of neural networks are asynchronous parallel processing, continous-time dynamics, and global interaction of network elements. Neural network models have received more and more attention in many fields where high computation rates are required. Examples include, optimization, linear and nonlinear programming, associative memory, pattern recognition and computer vision. Recently, combining the concepts of cellular automata and neural network, Chua and Yang [70,71] proposed the cellular neural network model and demonstrated its application to image processing. Hopfield [72] showed that the neural network can perform some signal processing tasks, such as the signal decomposition/decision problem, i.e. detecting the presence or absence of a basis in the presence of other waveforms and noise. Chua and Lin [73] have recently proposed a neural network approach for image transform coding. It is expected that these neural network models have significant potential applications to information processing and computing [74].

3.5 SUMMARY

Image processing involves vast amounts of data and computationally expensive processing. The performance of conventional computers has been found to inadequate for majority of image processing applications. The speed-ups obtained using parallel and pipeline processors, array processors and vector processors are still insufficient for most of the applications. This points to the need for special purpose architectures which more efficiently map the algorithms onto the computing structures.

Systolic arrays belong to the class of special purpose parallel and pipelined architectures. They provide an elegant solution to obtain a balance between compute bound and I/O-bound problems. Pyramid architectures are suitable for iterative algorithms where the information flows up and down the pyramid i.e. binary tree and quad tree based algorithms. Content-addressable memories have the advantage of parallel search capability based on the content of the location. These are particularly useful for database machines and pattern matching applications. Neural networks hold promise in many fields where high computation rates are required. Recently, some neural network models have been proposed which can have significant potential applications to information processing and computing.

In the following chapter, a review of the architectures for vector quantization is presented. Our research on architectures has focussed on (i) a systolic array architecture design for vector quantization, detailed in chapter 7, and (ii) a content-addressable memory architecture for real-time vector quantization, presented in chapter 8.

4. REVIEW OF ARCHITECTURES FOR VECTOR QUANTIZATION

The computational complexity of VQ has been the bottleneck in implementing the algorithm thus limiting its practical applications. Recently, several dedicated structures have been proposed to implement vector quantization in real-time for speech and image coding applications. The basic concept is to increase the speed-up factor (S_p), thereby reducing the effective execution time of the encoding algorithm.

$$T_{en(effective)} = T_{en(SISD)} / S_p \qquad (4.1)$$

where, $T_{en(SISD)}$ is the execution time using an SISD machine. Typical values for L, N, and K for a 512 x 512 image using a codebook of size 256 (equivalent to a bit rate of 0.5 bits/pixel for VQUC) are, respectively, 16, 256, and 16384. Assuming that the time taken to execute the basic operation using an SISD machine ranges from 100 - 200 ns [13-16], the speed-up required to satisfy real-time requirement (< 33 ms) ranges from 200 - 400. However, for real-time codebook generation (assuming 100-200 iterations of the LBG algorithm), the speed-up required ranges from 20000 - 80000.

The architectures reported in the literature to speed-up the execution time can be classified into one of three categories: high-speed architectures, parallel and pipelined architectures, and modified VQ based architectures.

4.1 HIGH-SPEED ARCHITECTURES

The first class of architectures is based on efficient execution of the distortion computation by using fast processors or a pipeline of fast processors.

(i) Fast Processors :

The first approach is to use fast processors with a short cycle time, T_c, to execute the basic

operation. This results in a speed-up factor of

$$S_p = T_e / T_c \qquad (4.2)$$

where, T_e, is the time to execute the basic operation using an SISD machine.

Tao, *et al* , [8] have designed a full search VQ system for speech coding based on this approach using LSTTL and CMOS devices. Here, the basic operation is executed in 500 ns. This implementation has a large device count and the speed-up factor is not high.

(ii) Pipeline Processors :

A second approach is to use a pipeline of fast processors, where each processor in the pipeline executes a portion of the distortion computation. The pipeline architecture results in a reduced cycle time (1/n compared to the previous approach) and the speed-up factor is given by,

$$S_p = n * [T_e / T_c] \qquad (4.3)$$

where, **n** is the number of processors in the pipeline. Davidson *et al* , [9,11] have reported on a high-speed architecture where the difference, squaring and accumulation operations in the distortion computation are executed in a pipeline, using a semi-custom VLSI pattern matching chip (PMC). Recently, Chase and Gersho [12] have reported on a real-time codebook generation hardware for speech coding based on the PMC, running in parallel with a TMS320C25 signal processor. The PMC chip computes the basic distortion operation in 366 ns.

This class of architectures yield sufficient speed-up for real-time speech coding. Since there are limitations on the smallest cycle time achievable, the two approaches discussed are not suitable for image coding applications as the throughput rate for an image is three orders of magnitude greater than that for speech.

4.2 PARALLEL AND PIPELINED ARCHITECTURES

It can be observed from equation 2.26 that the distortion computations can be processed independently. The second class of architectures exploit this property based on parallel and/or pipeline execution of the VQ algorithm.

(i) SIMD Processors :

A single instruction multiple data (SIMD) processor can be used to execute the distortion

computation in parallel in the direction of the vector dimension L. This yields an additional speed-up of L:

$$S_p = n * [T_e / T_c] * L \qquad (4.4)$$

An example of this processor is the systolic array dual-distortion processor module (DPM), reported by Abut, *et al* [16] where L basic operations in the distance computation are executed in parallel. The basic distortion operation itself is executed in 100 ns using a pipeline processor. The speed-up achieved using this approach is still not sufficient for image encoding in real-time as T_d still depends on N and K.

(ii) MSIMD Processors :

A different approach to speed-up the encoding operation is to parallelize the distortion computation in the direction of the codebook size N by using an Multiple SIMD (MSIMD) processors. The speed-up factor is given by,

$$S_p = n * [T_e / T_c] * L * N \qquad (4.5)$$

Sun and Hsu [15] have designed a special purpose wavefront array for full search VQ based on this approach. The design involves a rectangular array of (L+2)N basic processing cells with regular and local interconnections. The cells are organized in parallel in the direction of vector dimension L, and in pipeline in the direction of the codebook size N. The upper part (L x N) of the array serves as the computing medium and the lower part of the array is used to compare the distortion and select the label of the closest codeword. Once again, the basic operation is executed in 100 ns using a pipeline processor. This design has a latency of N+K+1 time units. Abut, *et al*, [16] have also reported on an architecture using VHISC technology based on fuzzy associative memory (FAM) chips which provides an increased speed-up by exploiting parallelism in the direction of L and partially in the direction of K. Davidson, *et al* , [7] have proposed a family of architectural techniques based on the concept of systolic arrays to implement vector quantization. These include: word-level architecture, bit-level architecture, two-dimensional array architecture, sub-linear and super linear array architectures for speech and image coding applications. These offer efficient computation of the distance measure for the encoding operation. In general, the use of multiple SIMD processors yields sufficient speed-up for real-time image encoding, but not

real-time codebook generation as T_d is still proportional to K which is much greater than N and L. One additional problem is the large number of processors required.

4.3 MODIFIED VQ BASED ARCHITECTURES

The number of processors can be reduced and T_d can be decreased by designing architectures to implement modified forms of VQ. For example, architectures which implement tree-search VQ and multi-stage VQ have been reported in the literature.

(i) Tree-search VQ Processors :

The encoding time T_d can be made proportional to logN instead of N by using a tree search VQ [54] instead of a full search VQ. This entails an additional speed-up of N/Log N making possible image encoding in real-time. The architecture reported by Abut, *et al* [16] can also be used with tree-structured codebooks. Dianysian and Baker [14] have designed a programmable SIMD machine that vector quantizes image sequences in real-time using tree-structured codebooks. The basic operation is executed in 100 ns.

(ii) Multi-stage VQ Processors :

Multi-stage VQ [55] makes possible the use of moderate size codebooks at each stage instead of a large sized codebook as in single stage VQ. Ramamoorthy and Potu [13] have proposed a bit-serial systolic chip set using multi-stage VQ for real-time image encoding. The design involves an inner product processor to compute the distortion and a comparator-address generator to determine the label. The bit-serial architecture enables integration of a large number of PE's on a chip.

(iii) Neural Networks :

Kohonen [75,76] introduced a neural network clustering technique for codebook generation for speech coding. Here, the weights (synaptic strengths) between neurons are adaptively sensitized to the prevailing input patterns. This clustering technique is not iterative and does not segment the training sequence into partitions before calculating the centroids. Instead, the asymptotic values of the weights define the codebook entries. Nasrabadi and Feng [18] have used this algorithm to design a codebook for vector quantization of images. The neural network can be implemented in

parallel by a two-dimensional grid of interconnected processors.

4.4 SUMMARY

In this chapter, a review of architectures for VQ was presented. The first class of architectures is based on efficient execution of the distortion computation by using fast processors or a pipeline of fast processors. The second class of architectures exploit the possibility of independent processing of distortion computation by using parallel and/or pipeline architectures. The third class of architectures implement a modified form of vector quantization. The maximum speed-up achieved by all these implementations is still not sufficient for executing the codebook generation algorithm for image coding in real-time, since it involves several iterations of the encoding algorithm. Chapters 5-8 now presents the research contribution to the thesis. The adaptive techniques described in chapters 5 and 6 are computationally less expensive and are easily mappable to architectures for real-time implementation. In chapter 7, a systolic architecture is presented where, the codebook generation and encoding operations are overlapped in the same structure. A novel architecture which makes possible real-time codebook generation using a content-addressable memory is presented in chapter 8.

5. ALGORITHMS FOR IMAGE ADAPTIVE VECTOR QUANTIZATION

5.1 INTRODUCTION

In this chapter, two new adaptive algorithms for vector quantization of images are presented. We recall from section 2.3.2, that VQ using a fixed or universal codebook (VQUC) has two problems which result in degraded coding performance. First of all, as the training set must be chosen to be representative of the class of images, to maintain a low distortion over all possible images, a large codebook may, therefore, be required. In turn this entails a high bit rate for label transmission. Secondly, if an image not represented in the training set is transmitted, then no guarantee can be given with regards to the distortion. The coding performance can be improved by using techniques which exploit the inter-vector correlation [45-47] resulting in a smaller codebook size. But these fixed codebooks cannot guarantee a good performance for a variety of images. An alternative strategy is to design a codebook which more closely matches the local image statistics. The codevector replacement schemes proposed by Gersho *et al* [48] and Yeh [49,50] provide partial adaptation by changing a small part of the codebook. Goldberg *et al* [6,42] have presented an adaptive technique where a new codebook is designed for each input image or parts of the image. These techniques provide an improved coding performance at the expense of increased computational complexity making it difficult to implement VQ algorithm in real-time. We propose two adaptive algorithms [80] which provide a compromise between coding performance and computational complexity resulting in a good performance at a reduced complexity.

5.2 REDUCED CODEBOOK ALGORITHM (VQRUC)

Even though a large codebook may be required to guarantee a low distortion over all possible images, only a subset of these codewords is required for coding a specific image. In other words, only some of the Voronoi regions are occupied as shown in Fig. 5.1. If these regions can be determined, then only the corresponding codewords can be used as the sub-codebook resulting in

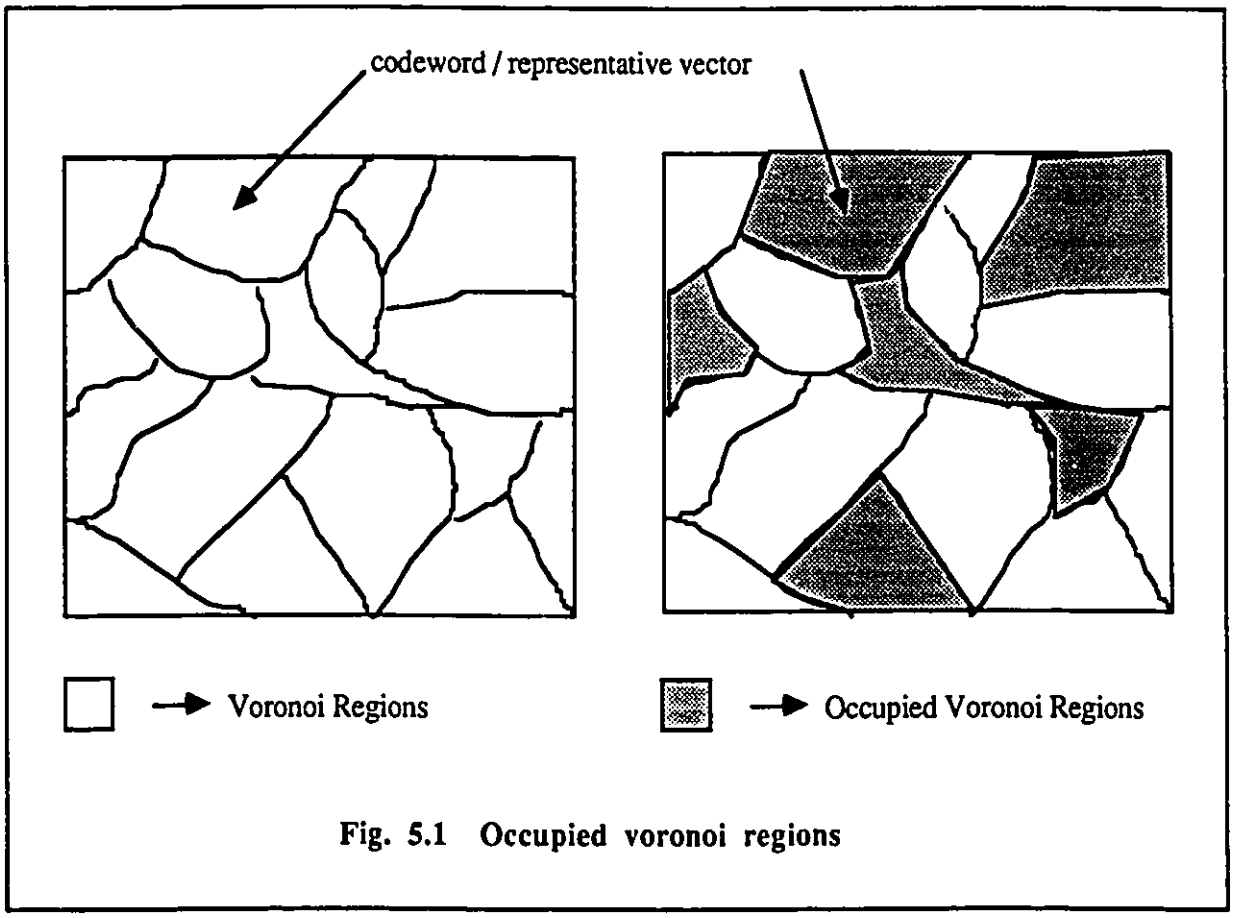

Fig. 5.1 Occupied voronoi regions

considerable savings in the bit rate for label transmission. We call this approach VQ with a reduced universal codebook (VQRUC). The input vectors are coded using the universal codebook and the corresponding labels are determined. An usage map is then created to indicate the codewords used. A reduced codebook is formed as shown in Fig. 5.2 consisting of only the subset of codewords used in the universal codebook resulting in a lower bit rate for label coding. The algorithm is expressed as follows,

PROCEDURE ***VQRUC*** (**In:** Universal_codebook, Input_vectors ; **Out:** Reduced_codebook, Usage_map, Labels)

Initialize all bits of Usage_map to 0.

DO

For each input vector

BEGIN

Determine the closest match codeword;

Label := Index of the closest match codeword;

Set Usage_map bit corresponding to the closest match codeword to a 1.

END

END DO

DO

For each codeword

IF < Usage_map bit := 1 > **THEN**

Transfer codeword to Reduced_codebook.

END DO

Translate labels to correspond to the reduced codebook.

END PROCEDURE

The new labels and the usage map (overhead) are then transmitted. The total bit rate, R_t, consists of two components: one for label transmission R_l, and one for the usage map transmission R_u. A variable bit rate coder is used to transmit the labels.

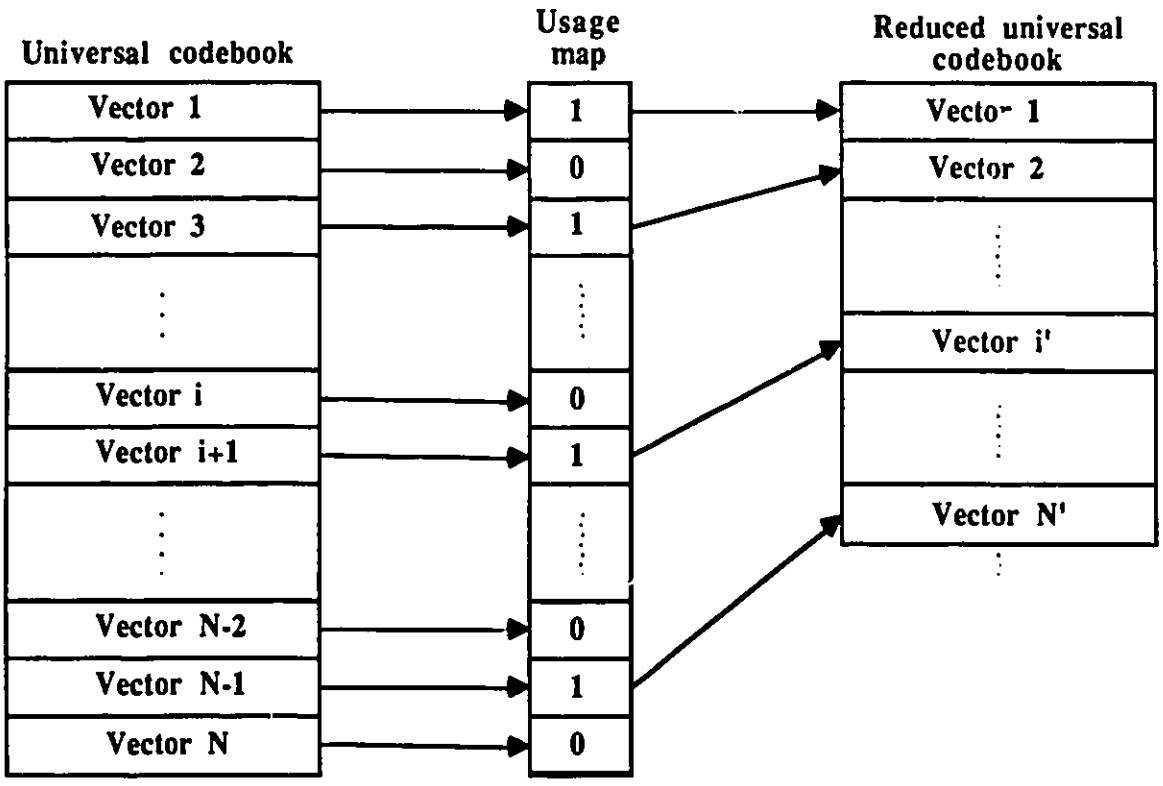

Fig. 5.2 Formation of the reduced codebook

Thus,

$$R_t = R_l + R_u \qquad (5.1)$$

where,

$$R_u = \{ N/(M \times M) \} \qquad (5.2)$$

Here,

M x M is the number of pixels in the image, and

N is the number of codewords in the universal codebook.

5.3 ONE-PASS ALGORITHM (VQRUC-1)

It has been reported that adaptive techniques which track image statistics from image to image are useful [6,42]. For example, an image adaptive codebook generation (VQIAC) can be used, but, this procedure is computationally expensive. A trade-off between computational complexity and coding performance has to be made to implement the algorithm in real-time. VQRUC can be improved somewhat by optimizing the codebook to the local image. This essentially involves one iteration of the LBG algorithm using the universal codebook as the initial codebook, and where the vectors are drawn from the new image. The new codebooks are now adapted to the input image. The difference between the new and old centroids must be transmitted. We note that as in VQIAC, the extra overhead information to be transmitted has to be factored against the smaller label size. We call this technique VQRUC-1, where 1 refers to one iteration of the LBG algorithm. In algorithmic form it is expressed as follows,

PROCEDURE ***VQRUC -1*** (**In:** Universal_codebook, Input_vectors;

Out: Difference_codebook, Labels)

DO

For each input vector

BEGIN

Assign vector to a Voronoi region using the nearest neighbour rule (Eqn. 2.21).

END

END DO

```
END DO
DO
    For each occupied Voronoi region
    BEGIN
        Calculate the centroid of the region (using Eqn. 2.22);
        New_codeword := Centroid of the region;
        Difference_codeword := New_codeword - Universal codeword.
    END
END DO
DO
    For each input vector
    BEGIN
        Determine the closest match New_codeword;
        Label := Index of the closest match New_codeword.
    END
END DO
END PROCEDURE
```

The labels and the error codebook(overhead) are then transmitted. In this case, the total bit rate, R_t, consists of two components: one for label transmission R_l, and one for the difference codebook transmission R_d (overhead). An entropy coder [19-20] operating on a pixel-by-pixel basis is used to losslessly encode the labels and the error codevectors.

$$R_t = R_d + R_l \qquad (5.3)$$

5.4 SIMULATION AND DISCUSSION OF RESULTS

Computer simulations are carried out on two test images of size 256 x 256 pixels with 8 bits/pixel. The first image is a Face (Lena) image (Fig.5.8) and the second image is a radiographic (Chest) image (Fig.5.7). Simulations have been performed for two block sizes.

In the first case, the images are preprocessed to yield blocks of 4 x 4 pixels which are then

performed. In the first set of experiments, a universal codebook is designed from 10 different radiographic images of size 256 x 256 not including the test Chest image (without the training set). These images are subsampled (Fig.5.3) versions of the higher resolution (2048 x 2048) digitized radiographs. The VQUC, VQIAC, VQRUC, and VQRUC-1 techniques are applied to the test Chest image. In the second set of experiments, a universal codebook is designed from a training set of four (Frank, Boat, Table, and Girl) CCITT standard images not including the test Face (Lena) image (without the training set). The four techniques are applied to the Face image.

In the second case, the images are preprocessed to yield blocks of 2 x 2 pixels which are then rearranged as 4-dimensional vectors. Three sets of experiments have been performed. The first two sets of experiments are identical to that for 16-dimensional vectors. In the third experiment, the test chest image is coded using the VQRUC technique with a universal codebook designed from 10 different radiographic images including the test Chest image (within the training set). We call this technique VQRUC-W. The Chest image is also coded using VQRUC with the codebook generated in the first experiment from the four (Frank, Boat, Table, and Girl) CCITT standard images (without the training set and without class). We call this technique VQRUC-WC.

5.4.1 CODING PERFORMANCE

The coding performances are compared using the rate distortion curves. The bit rate R_t for VQUC, VQIAC, VQRUC and VQRUC-1 are computed using equations 2.24, 2.25, 5.1 and 5.3, respectively. The distortion is computed as the normalized mean square error (NMSE) and is evaluated using equation 2.14.

The results of the two sets of experiments for the first case with 16-dimensional vectors are tabulated in Tables 5.1a - 5.1d and 5.2a - 5.2d, respectively for the Chest and Face (Lena) images. The corresponding rate distortion curves are shown, respectively in Fig. 5.4a & 5.4b. The rate distortion curves show the improvement in performance in using the VQRUC and VQRUC-1 techniques. It can be observed that the best coding performance is obtained using VQRUC-1 over the range of bit rates from 0.2 - 0.6 bits/pixel for the Chest image. However, for the Face image, VQRUC provides the best coding performance in the range of 0.2 - 0.6 bit/pixel.

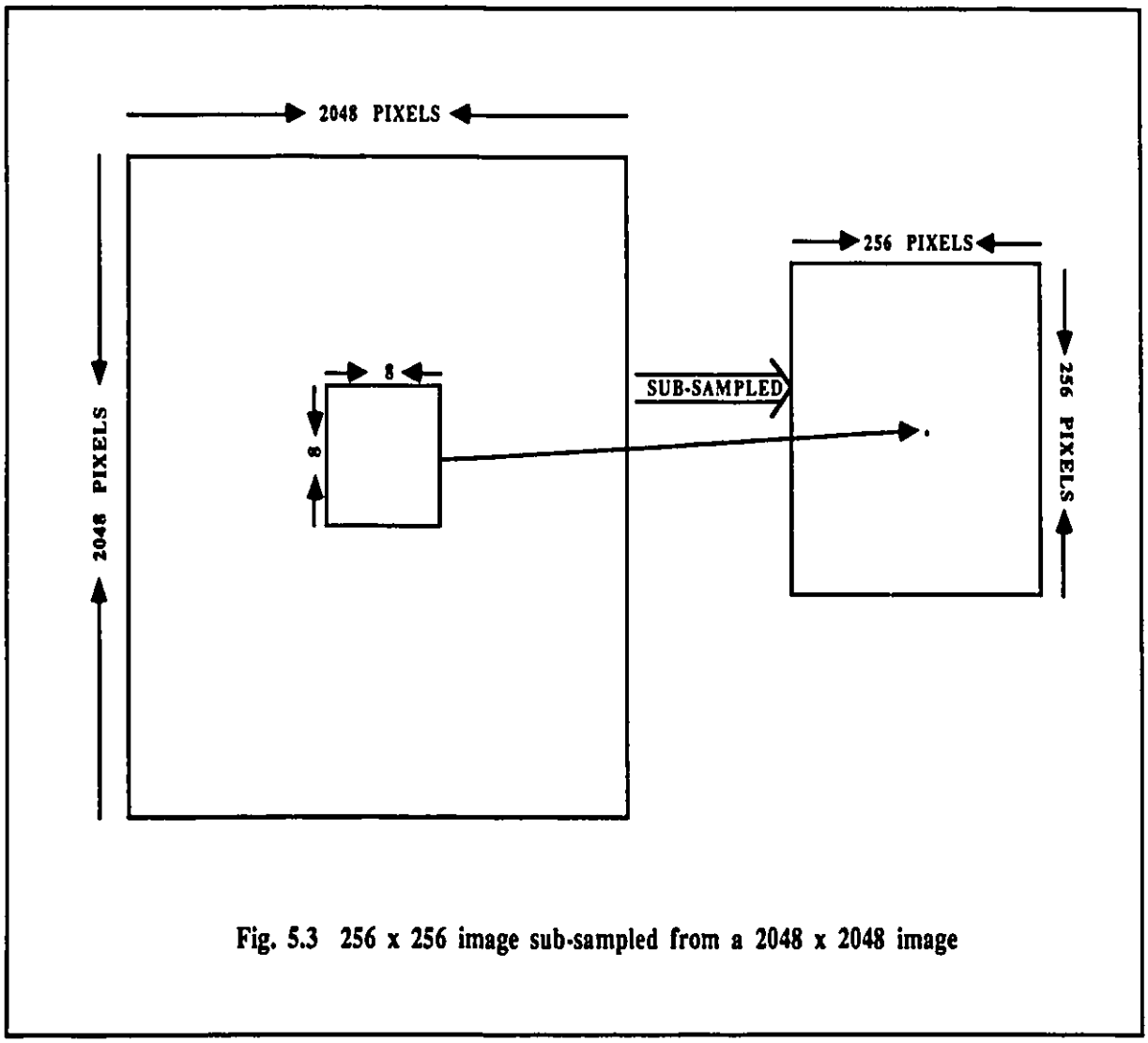

Fig. 5.3 256 x 256 image sub-sampled from a 2048 x 2048 image

TABLE 5.1a

Simulation results of coding the chest image with 16-D vectors using the VQUC technique

No. of codewords	Total bit rate	% NMSE	Computational complexity
8	0.19	1.31	8
16	0.25	0.87	16
32	0.31	0.63	32
64	0.38	0.48	64
128	0.44	0.37	128
256	0.50	0.31	256
512	0.56	0.26	512
1024	0.63	0.22	1024
2048	0.69	0.20	2048

TABLE 5.1b

Simulation results of coding the chest image with 16-D vectors using the VQIAC technique

No. of codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE	No. of iterations	Computational complexity
8	0.19	0.01	0.20	0.64	29	261
16	0.25	0.03	0.28	0.32	68	1156
32	0.31	0.07	0.38	0.20	92	3036
64	0.37	0.13	0.50	0.12	110	7150
128	0.44	0.25	0.69	0.079	129	16641
256	0.50	0.50	1.00	0.054	146	37522

TABLE 5.1c

Simulation results of coding the chest image with 16-D vectors using the VQRUC technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for used codewords info.	Total bit rate	% NMSE	Computational complexity
8	6	0.125	0.00012	0.13	1.31	8
16	9	0.163	0.00024	0.16	0.87	16
32	14	0.192	0.00049	0.19	0.63	32
64	27	0.230	0.00098	0.23	0.48	64
128	44	0.265	0.002	0.27	0.37	128
256	65	0.310	0.004	0.31	0.31	256
512	94	0.34	0.008	0.35	0.26	512
1024	145	0.385	0.016	0.40	0.22	1024
2048	206	0.423	0.032	0.46	0.20	2048

TABLE 5.1d

Simulation results of coding the chest image with 16-D vectors using the VQRUC-1 technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE	Computational complexity
8	6	0.125	0.006	0.13	1.1	14
16	9	0.163	0.009	0.17	0.72	25
32	14	0.192	0.015	0.21	0.43	46
64	27	0.230	0.032	0.27	0.27	91
128	44	0.265	0.050	0.32	0.19	172
256	65	0.310	0.069	0.39	0.14	321
512	94	0.34	0.095	0.45	0.11	606
1024	145	0.385	0.140	0.53	0.089	1169
2048	206	0.423	0.190	0.62	0.073	2254

TABLE 5.2a

Simulation results of coding the face (Lena) image with 16-D vectors using the VQUC technique

No. of codewords	Total bit rate	% NMSE	Computational complexity
8	0.19	3.26	8
16	0.25	2.59	16
32	0.31	2.07	32
64	0.38	1.80	64
128	0.44	1.56	128
256	0.50	1.39	256
512	0.56	1.25	512
1024	0.63	1.14	1024
2048	0.69	1.08	2048

TABLE 5.2b

Simulation results of coding the face (Lena) image with 16-D vectors using the VQIAC technique

No. of codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE	No. of iterations	Computational complexity
8	0.19	0.01	0.20	3.10	31	279
16	0.25	0.03	0.28	2.41	57	969
32	0.31	0.07	0.38	1.92	86	2838
64	0.37	0.13	0.50	1.53	115	7475
128	0.44	0.25	0.69	1.12	136	17544
256	0.50	0.50	1.00	0.91	152	39064

TABLE 5.2c

Simulation results of coding the face (Lena) image with 16-D vectors using the VQRUC technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for used code-words info.	Total bit rate	% NMSE	Computational complexity
8	8	0.181	0.00012	0.18	3.26	8
16	16	0.231	0.00024	0.23	2.59	16
32	32	0.278	0.00049	0.28	2.07	32
64	63	0.329	0.00098	0.33	1.80	64
128	123	0.384	0.002	0.39	1.56	128
256	230	0.435	0.004	0.44	1.39	256
512	408	0.484	0.008	0.49	1.25	512
1024	660	0.528	0.016	0.54	1.14	1024
2048	953	0.568	0.032	0.60	1.08	2048

TABLE 5.2d

Simulation results of coding the face (Lena) image with 16-D vectors using the VQRUC-1 technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE	Computational complexity
8	8	0.181	0.007	0.19	3.13	16
16	16	0.234	0.016	0.25	2.48	32
32	32	0.279	0.035	0.32	1.91	64
64	63	0.331	0.074	0.41	1.59	127
128	123	0.385	0.152	0.54	1.30	251
256	230	0.437	0.297	0.73	1.05	486
512	408	0.485	0.544	1.03	0.81	920
1024	660	0.531	0.887	1.42	0.63	1684

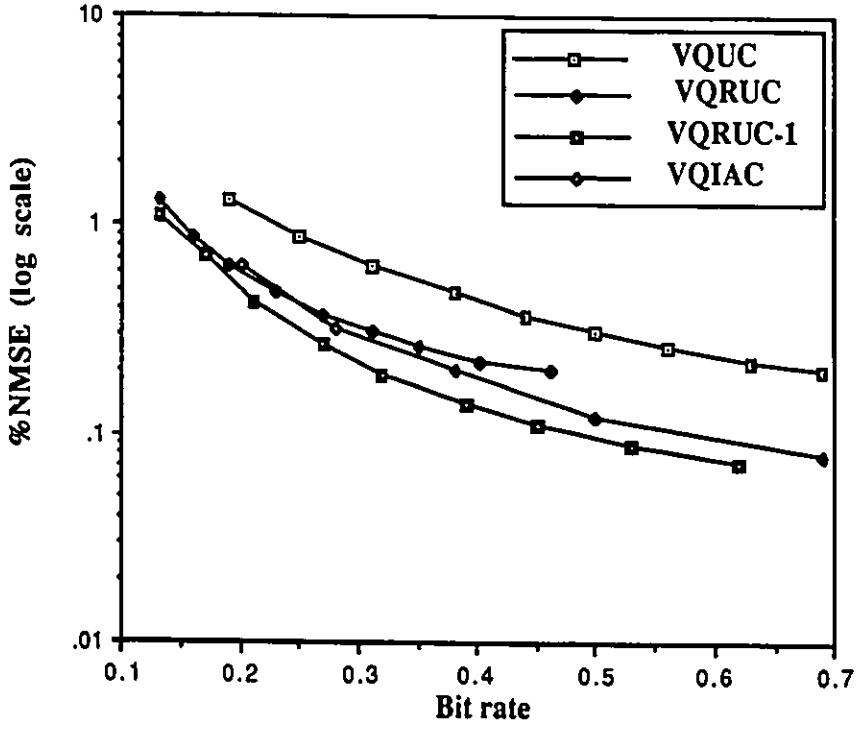

Fig. 5.4a Rate distortion curves for the Chest image with 16-D vectors

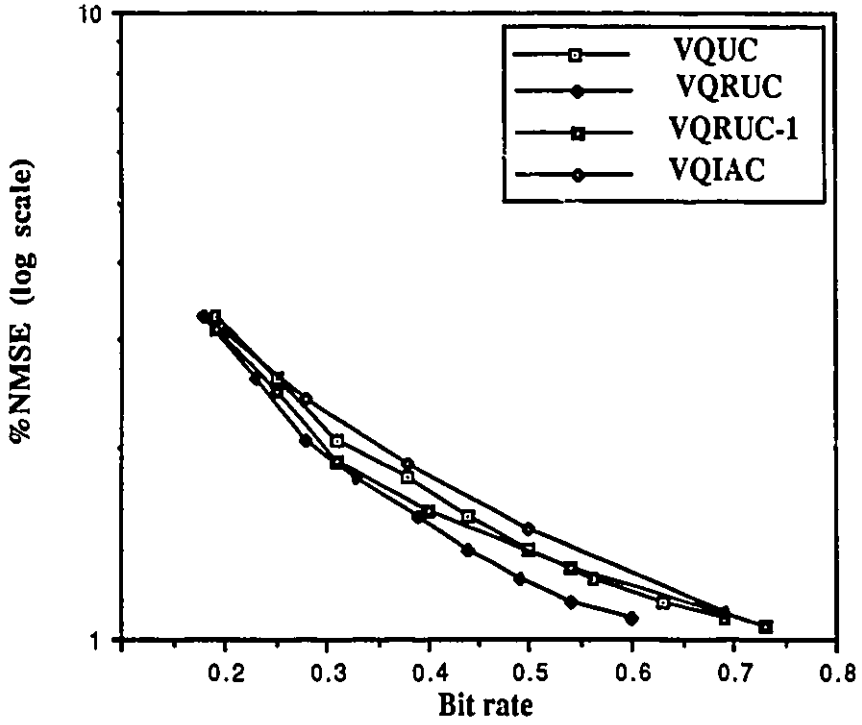

Fig. 5.4b Rate distortion curves for the Face image with 16-D vectors

The results of the first two sets of experiments for the second case with 4-dimensional vectors are tabulated in Tables 5.3a - 5.3d, and 5.4a - 5.4d for the two images, respectively. The corresponding rate distortion curves are shown in Fig. 5.5a & 5.5b. The rate distortion curves confirm the performance improvement in using the VQRUC and VQRUC-1 techniques. It can be observed that the best coding performance is obtained using VQRUC-1 over the range of bit rates from 0.6 - 2.0 bits/pixel for both Chest and Face images. The results of the third set of experiments is tabulated in Tables 5.5a & 5.5b and the corresponding rate distortion curves is graphed in Fig.5.6. It can be seen that VQRUC-W outperforms VQRUC, which in turn, outperforms VQRUC-WC.

The reconstructed Chest image using the VQRUC-1 at a bit rate of 1.5 bits/pixel is shown in Fig. 5.9. The corresponding error image (magnified by a factor of ten) is shown in Fig. 5.12. The reconstructed Face image using the VQRUC-1 at a bit rate of 1.42 bits/pixel (16-D vectors) and 1.72 bits/pixel (4-D vectors) are shown in Fig. 5.10 and Fig. 5.11, respectively. The corresponding error images (magnified by a factor of five) are shown in Fig. 5.13 and 5.14, respectively. It can be seen that very good quality reconstructed Face and Chest images can be obtained at a bit rate of approximately 1.5 bits/pixel. The error images illustrate the concentration of errors near the edges.

5.4.2 COMPUTATIONAL COMPLEXITY

We recall from section 2.5.2 and 2.5.3, that the computational complexity of the encoding and codebook generation operations for K input vectors of dimension L and a codebook of size N is K*L*N and K*L*N*I basic operations, respectively. Hence, the computational complexity of VQUC and VQRUC techniques is K*L*N basic operations. Note that the complexity to form the reduced codebook is very small and is, hence ignored. The computational complexity of VQRUC-1 and VQIAC are $K*L*(N+N_u)$ and K*L*N*(I+1) basic operations, respectively. For the present set of experiments, L and K have values of 16 and 4096, respectively. The complexity involved in using VQUC, VQIAC, VQRUC, and VQRUC-1 techniques for the Chest and Face images, with 16-dimensional vectors, are tabulated in Tables 5.1a - 5.1d and 5.2a - 5.2d, respectively. Note that the complexity values are normalized by a factor of 64000 for the sake of simplicity.

TABLE 5.3a

Simulation results of coding the chest image with 4-D vectors using the VQUC technique

No. of codewords	Total bit rate	% NMSE
8	0.75	1.183
16	1.00	0.649
32	1.25	0.260
64	1.50	0.141
128	1.75	0.086
256	2.00	0.062
512	2.25	0.040
1024	2.50	0.030
2048	2.75	0.027

TABLE 5.3b

Simulation results of coding the chest image with 4-D vectors using the VQIAC technique

No. of codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE
8	0.75	0.004	0.75	0.372
16	1.00	0.008	1.01	0.155
32	1.25	0.02	1.27	0.090
64	1.50	0.03	1.53	0.062
128	1.75	0.06	1.81	0.041
256	2.00	0.13	2.13	0.033
512	2.25	0.25	2.50	0.021

TABLE 5.3c

Simulation results of coding the chest image with 4-D vectors using the VQRUC technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for used code-words info.	Total bit rate	% NMSE
8	5	0.483	0.00012	0.48	1.183
16	9	0.612	0.00024	0.61	0.649
32	14	0.811	0.00049	0.82	0.260
64	27	1.058	0.00098	1.06	0.141
128	54	1.239	0.002	1.24	0.086
256	150	1.538	0.004	1.54	0.062
512	297	1.841	0.008	1.85	0.040
1024	577	2.113	0.016	2.13	0.030
2048	1039	2.333	0.032	2.36	0.027

TABLE 5.3d

Simulation results of coding the chest image with 4-D vectors using the VQRUC-1 technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE
8	5	0.519	0.001	0.52	0.934
16	9	0.628	0.002	0.63	0.517
32	14	0.825	0.002	0.85	0.302
64	27	1.062	0.003	1.07	0.109
128	54	1.249	0.008	1.26	0.067
256	150	1.547	0.021	1.57	0.045
512	297	1.849	0.039	1.89	0.025
1024	577	2.117	0.064	2.18	0.016
2048	1039	2.333	0.104	2.44	0.013

TABLE 5.4a

Simulation results of coding the face (Lena) image with 4-D vectors using the VQUC technique

No. of codewords	Total bit rate	% NMSE
8	0.75	2.48
16	1.00	1.97
32	1.25	1.76
64	1.50	1.65
128	1.75	1.56
256	2.00	0.54
512	2.25	0.43
1024	2.50	0.35
2048	2.75	0.31

TABLE 5.4b

Simulation results of coding the face (Lena) image with 4-D vectors using the VQIAC technique

No. of codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE
8	0.75	0.004	0.75	1.76
16	1.00	0.008	1.01	1.20
32	1.25	0.02	1.27	0.86
64	1.50	0.03	1.53	0.63
128	1.75	0.06	1.81	0.47
256	2.00	0.13	2.13	0.28
512	2.25	0.25	2.50	0.20

TABLE 5.4c

Simulation results of coding the face (Lena) image with 4-D vectors using the VQRUC technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for used code-words info.	Total bit rate	% NMSE
8	8	0.549	0.00012	0.55	2.48
16	16	0.675	0.00024	0.68	1.97
32	28	0.732	0.00049	0.73	1.76
64	50	0.668	0.00098	0.67	1.65
128	55	0.828	0.002	0.83	1.56
256	198	1.668	0.004	1.67	0.54
512	452	2.008	0.008	2.02	0.43
1024	895	2.315	0.016	2.33	0.35
2048	1768	2.567	0.032	2.60	0.31

TABLE 5.4d

Simulation results of coding the face (Lena) image with 4-D vectors using the VQRUC-1 technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for codewords	Total bit rate	% NMSE
8	8	0.568	0.002	0.57	2.31
16	16	0.696	0.004	0.70	1.75
32	28	0.763	0.006	0.77	1.62
64	50	0.714	0.012	0.73	1.53
128	55	0.854	0.012	0.87	1.45
256	198	1.680	0.043	1.72	0.44
512	452	2.026	0.095	2.12	0.31
1024	895	2.329	0.179	2.51	0.23

TABLE 5.5a

Simulation results of coding the chest image with 4-D vectors using the VQRUC (within training set) technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for used code-words info.	Total bit rate	% NMSE
8	5	0.484	0.00012	0.49	1.180
16	9	0.613	0.00024	0.61	0.597
32	16	0.837	0.00049	0.84	0.216
64	28	1.059	0.00098	1.06	0.123
128	52	1.251	0.002	1.25	0.079
256	149	1.539	0.004	1.54	0.053
512	301	1.845	0.008	1.85	0.031
1024	580	2.113	0.016	2.13	0.022
2048	1054	2.339	0.031	2.37	0.018

TABLE 5.5b

Simulation results of coding the chest image with 4-D vectors using the VQRUC (without training set and without class) technique

No. of codewords	No. of used codewords	Bit rate for label	Bit rate for used code-words info.	Total bit rate	% NMSE
8	3	0.324	0.00012	0.33	5.378
16	4	0.340	0.00024	0.34	2.369
32	6	0.406	0.00049	0.41	2.055
64	6	0.412	0.00098	0.41	2.041
128	7	0.431	0.002	0.43	0.149
256	66	1.204	0.004	1.21	0.138
512	180	1.527	0.008	1.53	0.092
1024	315	1.730	0.016	1.75	0.062
2048	507	1.906	0.032	1.94	0.044

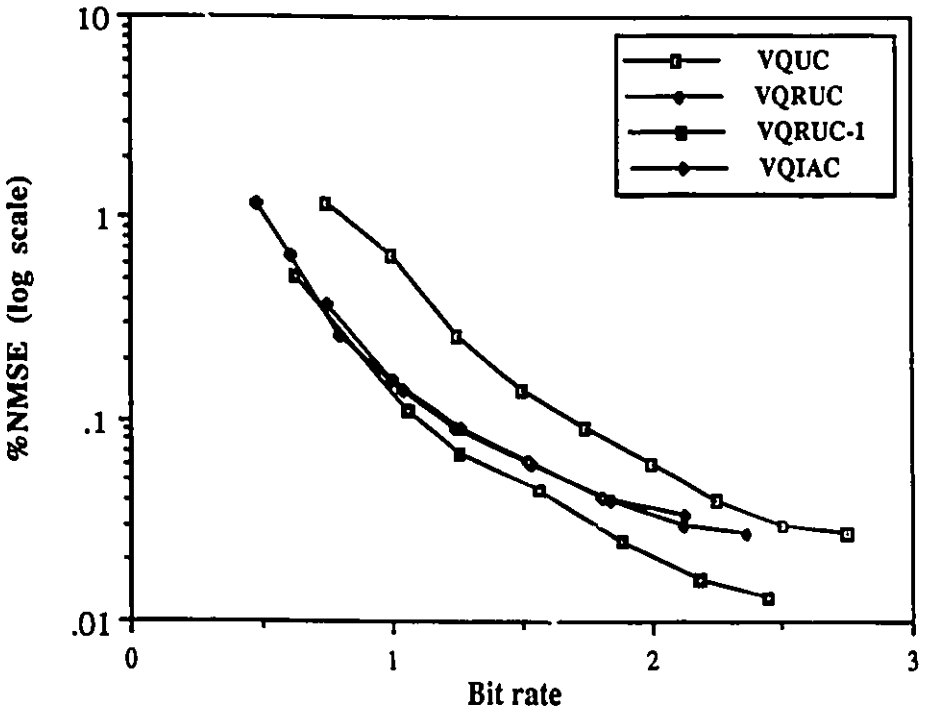

Fig. 5.5a Rate distortion curves for the Chest image with 4-D vectors

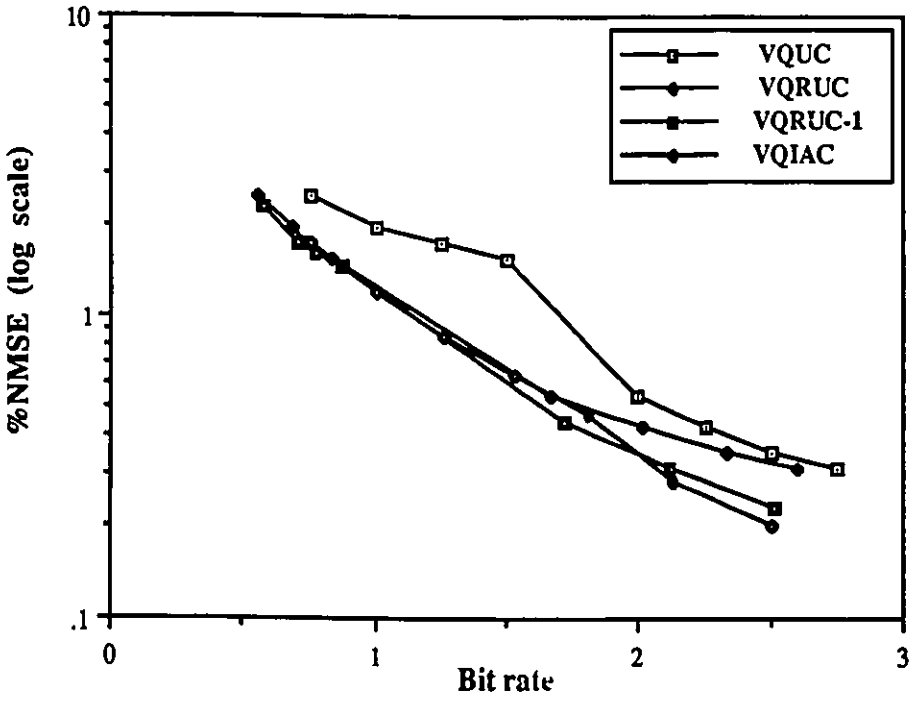

Fig. 5.5b Rate distortion curves for the Face image with 4-D vectors

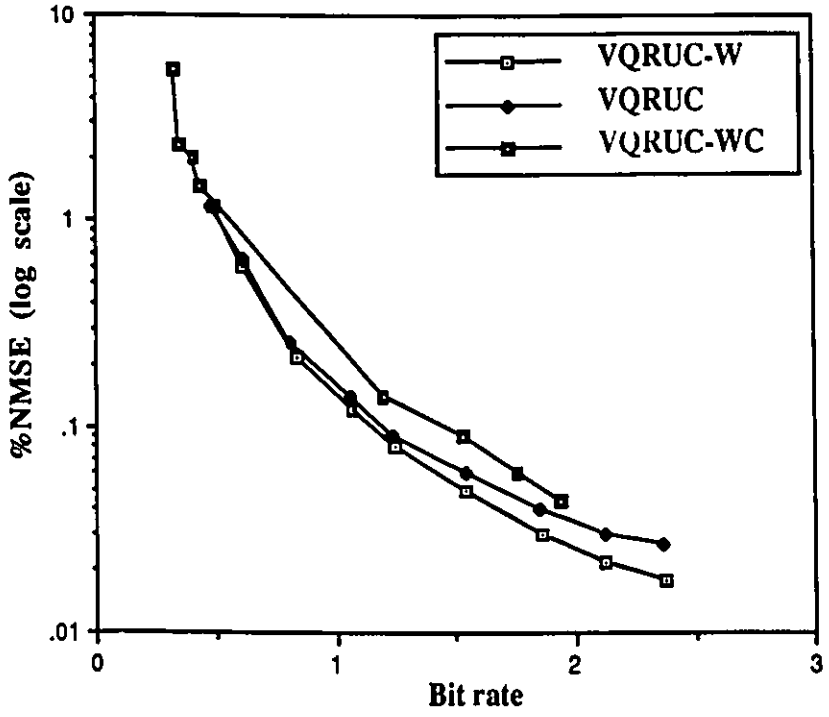

Fig. 5.6 Rate distortion curves for the Chest image using the VQRUC technique

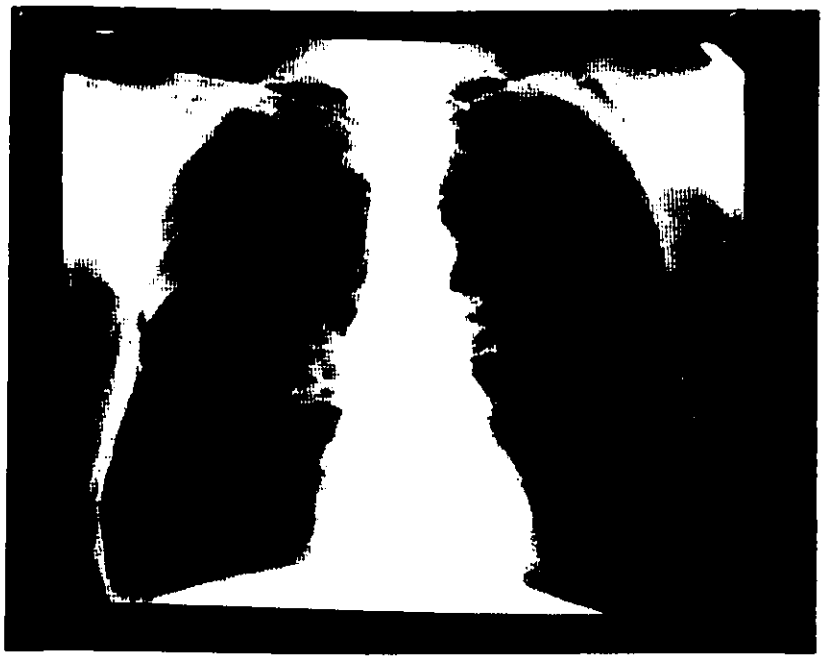

Fig. 5.7 The original Chest image of 256 x 256 pixels with 8 bits/pixel

Fig. 5.8 The original Face (Lena) image of 256 x 256 pixels with 8 bits/pixel

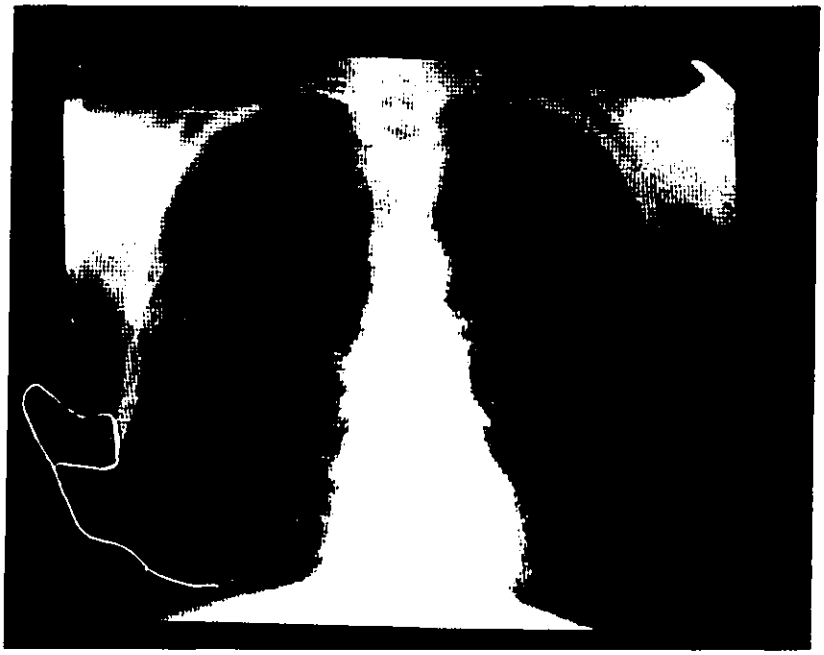

Fig. 5.9 The reconstructed Chest image using VQRUC-1 with 4-D vectors at a bit rate of 1.5 bits/pixel

Fig. 5.10 The reconstructed Face (Lena) image using VQRUC-1 with 16-D vectors at a bit rate of 1.42 bits/pixel

Fig. 5.11 The reconstructed Face (Lena) image using VQRUC-1 with 4-D vectors at a bit rate of 1.72 bits/pixel

Fig. 5.12 The error Chest image using VQRUC-1 with 4-D vectors at a bit rate of 1.5 bits/pixel

Fig. 5.13 The error Face (Lena) image using VQRUC-1 wi h 16-D vectors at a bit rate of 1.42 bits/pixel

Fig. 5.14 The error Face (Lena) image using VQRUC-1 with 4-D vectors at a bit rate of 1.72 bits/pixel

5.4.3 DISCUSSION OF THE RESULTS

The results of the two sets of experiments for 4-D and 16-D vectors clearly demonstrate the superior coding performance of VQRUC and VQRUC-1 techniques. It can be observed that for almost the same computational complexity, VQRUC provides a better performance than VQUC, as a smaller number of codewords are addressed, which in turn, results in savings in bit rate for label transmission. The VQRUC-1 technique provides an improvement in coding performance over VQRUC for a small increase in computational complexity as only one iteration of the algorithm needs to be carried out. VQRUC-1 outperforms VQIAC at a substantially reduced computational complexity. The performance improvement is the result of needing only to send the changes to the receiver, rather than an entirely new codebook.

Tables, qualitatively comparing the four techniques for 4-D and 16-D vectors on the Chest and Face images, are shown in Tables 5.6a & 5.6b, respectively. Note that a scale of ten is used to express the relative performance, where a value of ten corresponds to the technique with the best performance. The tabulated values are for two different bit ranges: low bit rate (0.2 - 0.7 bits/pixel) which corresponds to the simulations with 16-D vectors, and medium bit rate (0.7 - 2.0 bits/pixel) which corresponds to simulations with 4-D vectors. The "Not applicable" entries in the low bit rate column results from the large bit rates required for label transmission with 4-D vectors. For the medium bit rate rate column, the problem is the practical difficulty in generating large size codebooks.

It can be observed from Table 5.6a that for the Chest image, a large performance improvement is obtained by employing adaptivity. This implies that the individual radiographic images are statistically quite different. VQRUC-1 provides the best performance at both low and medium bit rates for a marginal increase in computational complexity. The performance improvement is 60-80% over VQUC, and 10-50% over VQRUC at twice the complexity. This is in contrast to a 25-50% improvement in performance using VQIAC (over VQUC), but at the expense of 25-150 times the complexity. The large computational complexity of VQIAC is the result of the large number of iterations required for optimum codebook generation. The reconstructed Chest image in Fig. 5.9 shows that excellent quality coded images can be obtained at a bit rate of 1.5 bits/pixel.

TABLE 5.6a

Performance comparison chart for the Chest image
(a scale of 10 is used to express the relative performance)

Algorithm	Vector dimension	Low bit rate 0.2 - 0.7 bpp	Medium bit rate 0.7 - 2.0 bpp	Complexity
VQUC	4-D	Not applicable	2 - 4	1
	16-D	3 - 4	Not applicable	
VQRUC	4-D	Not applicable	5 - 8	1.01
	16-D	6 - 9	Not applicable	
VQRUC-1	4-D	Not applicable	10	1.5 - 2
	16-D	10	Not applicable	
VQIAC	4-D	Not applicable	5 - 7	25 - 150
	16-D	8 - 9	Not applicable	

TABLE 5.6b

Performance comparison chart for the Face image
(a scale of 10 is used to express the relative performance)

Algorithm	Vector dimension	Low bit rate 0.2 - 0.7 bpp	Medium bit rate 0.7 - 2.0 bpp	Complexity
VQUC	4-D	Not applicable	3 - 6	1
	16-D	8 - 8.5	Not applicable	
VQRUC	4-D	Not applicable	8.5 - 9	1.01
	16-D	10	Not applicable	
VQRUC-1	4-D	Not applicable	10	1.8 - 2
	16-D	9 - 9.5	Not applicable	
VQIAC	4-D	Not applicable	9 - 9.5	30 - 155
	16-D	7.5 - 8	Not applicable	

It can be seen from Table 5.6b that for the Face image, VQRUC provides the best coding performance at low bit rates and VQRUC-1 at medium bit rates. The performance improvement of VQRUC-1 is 40-70% over VQUC and 10-15% over VQRUC at medium bit rates at twice the complexity. This is in contrast to a 10-30% improvement (over VQUC) in using VQIAC, but at the expense of 30-155 times the complexity. The reconstructed Face image in Fig. 5.10 and 5.11 shows that good quality coded images can be obtained at a bit rate of approximately 1.5 bits/pixel. However, the important point is the noticeable blocking effects using 16-D vectors compared to 4-D vectors. The error images in Fig. 5.12 (magnified by a factor of ten) and in Fig. 5.13 & 5.14 (magnified by a factor of five) shows that most of the errors are concentrated near the edges.

The simulation results for VQRUC on the Chest image with 4-D vectors using different codebooks tabulated in Tables 5.5a, 5.5b and 5.3c, and graphed in Fig. 5.6, demonstrate that a 5-33% improvement in coding performance can be obtained using VQRUC-W over VQRUC. This in turn, is a 40-70% improvement compared to VQRUC-WC.

5.5 SUMMARY

In this chapter, two new adaptive techniques for vector quantization of images were presented. In the first technique (VQRUC), a reduced set of codewords from the universal codebook is used in coding an image. This results in savings in the bit rate for label transmission. A usage map is transmitted to the receiver to indicate which of the the codewords are used. The second technique (VQRUC-1) involves one iteration on the reduced codebook as the initial codebook to adapt the codewords to the image to be coded. The error codebook is transmitted to the receiver. Simulation results demonstrate the improvement in coding performance and the gains in computational complexity using the proposed techniques. A technique which minimizes the maximum error and where the codebook generation is done on the fly is detailed in the next chapter.

6. A MINI-MAX ERROR CRITERION BASED ALGORITHM FOR IMAGE ADAPTIVE VECTOR QUANTIZATION

6.1 INTRODUCTION

In this chapter, we present a technique which employs the mini-max error criterion for image compression using adaptive vector quantization. We recall from section 2.3.2 that the image vectors are usually coded using a "universal codebook" generated from a set of training images. The coding performance using this codebook is potentially poor for images outside the training set. A number of inter[42,44,49] and intra-image[6,46] techniques have been proposed to adapt the codewords to the input image(Ref. sec.2.3.2). In all these techniques, the mean square error criterion is used to find the closest codeword for a given input vector. This measure does not guarantee that the closest codeword for a given input vector is within a prespecified bound. This can result in large errors which give rise to artifacts. We propose an intra-image adaptive technique (VQMMC) [81] which employs a criterion that minimizes the maximum error. Here, the codebook is generated on the fly from the input vectors to be coded. A primary codebook of size, 2,4, 8 or 16 is typically used to store the frequently used codewords. A larger secondary codebook is used to store the less frequently used codewords. Both the transmitter and receiver maintain identical codebooks and hence keep track of any changes without any overhead information. The simulation results demonstrate excellent rate-distortion performance and also confirm the absence of large errors when compared to the VQUC technique. As it is a single-pass technique, real-time implementation is possible.

6.2. MINI-MAX ERROR ALGORITHM FOR ADAPTIVE VQ

We now describe our intra-image adaptive technique which is based on the mini-max error criterion:

$$\min \{d(V_i, C_p)\} = \min\{\max (| V_{ij} - C_{pj} |)\} \text{ over all } j \qquad (6.1)$$

In other words, this measure minimizes the maximum error between the original and decoded pixel values. We call this technique VQMMC. In this technique, the codebook is generated on the fly and avoids the computational complexity involved in generating the optimum codebook for the entire image. Given the inherent simplicity of the mini-max measure and the small codebook used real-time implementation is possible.

A primary codebook of size, 2, 4, 8 or 16 is typically used. To start with, the input vectors themselves are used as the codewords and a match within a specified threshold is sought. If no match is obtained, then the input vector is appended to the codebook as a new codeword. If a match is obtained, the label of the codeword is transmitted. Both the transmitter and the receiver maintain identical codebooks as shown in Fig. 6.1 and hence keep track of any changes. When the primary codebook is filled, the least recently used (LRU) codeword in the primary codebook is shifted to a larger secondary codebook freeing room for the new codeword. From this point onwards, the incoming vector first searches for a match in the primary codebook, however, if it fails, the secondary codebook is also searched. A new codeword is appended only if no match is obtained in both the primary and secondary codebooks and in that event, the LRU codeword in the primary codebook is shifted to the secondary codebook. It is to be noted that if the secondary codebook is full, the LRU codeword is deleted from the codebook. If a match is observed in the secondary codebook, the corresponding label is transmitted and the vector gets switched with LRU codeword in the primary codebook. This process of dynamically updating the codewords takes place both at the transmitter and the receiver without any overhead information.

The algorithm (VQMMC) is expressed as follows.

```
BEGIN
    Place first input vector in primary_codebook;
    Transmit it to the receiver.
    REPEAT (for each vector)
        Search primary_codebook for a match*
        IF < match > THEN
            Transmit the corresponding label.
        ELSE
            BEGIN
                IF < primary_codebook is not full > THEN
                    BEGIN
```

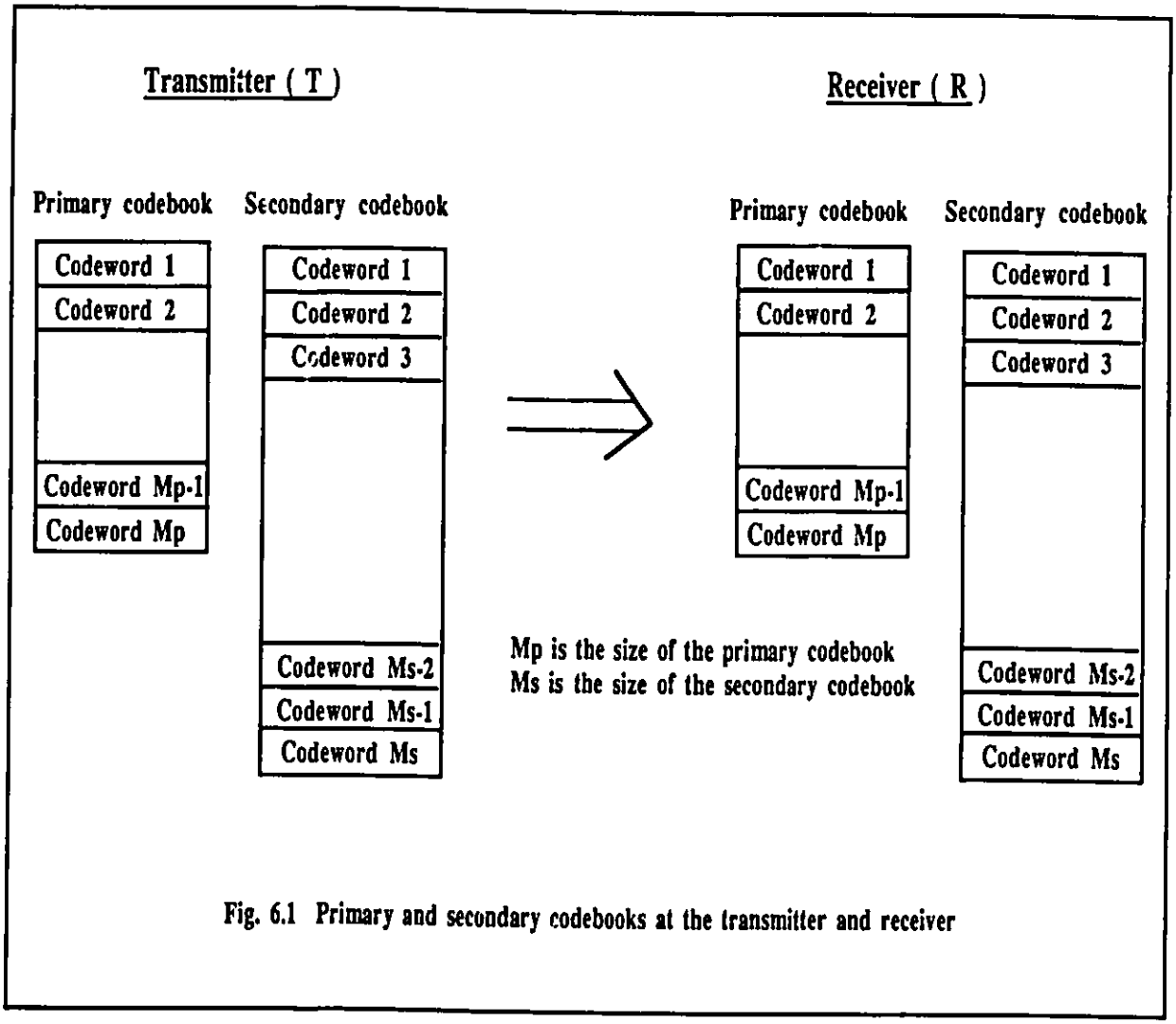

Fig. 6.1 Primary and secondary codebooks at the transmitter and receiver

```
                Add vector as new codeword to primary_codebook;
                Transmit the codeword to the receiver.
            END
        ELSE
            BEGIN
                Search secondary_codebook for a match*
                IF < match > THEN
                    BEGIN
                        Transmit the corresponding label;
                        Swap the matched codeword with the least frequently used
                        (LRU) codeword of the primary_codebook.
                    END
                ELSE
                    BEGIN
                        IF < secondary_codebook is full > THEN
                            BEGIN
                                Delete LRU codeword in secondary_codebook.
                            END
                        Transfer LRU codeword in primary_codebook to
                        secondary_codebook;
                        Add vector as new codeword to primary_codebook;
                        Transmit the codeword to the receiver.
                    END
            END
    END
  UNTIL (all vectors are done)
END
```

*A match is obtained when the maximum error between a vector and the codeword (as defined in equation 6) is within a prespecified threshold value.

To summarize, the advantages of the VQMMC algorithm are as follows:

(a) All vectors are guaranteed to be reproduced within an error threshold.

(b) Since the input image vectors themselves are used as the codewords, the technique is adapted to the local statistics of the source.

(c) Since both transmitter and receiver keep track of the changes and switch between the primary and secondary codebooks simultaneously, this technique is adaptive. It is to be noted that the adaptation does not involve any overhead information.

(d) As it is a single-pass technique with the codebook being generated on the fly, real-time implementation is possible.

6.3. SIMULATION AND DISCUSSION OF RESULTS

Computer simulations are carried out on two test images of size 256 x 256 pixels with 8 bits/pixel. The first image is a Face (Lena) image (Fig.5.8) and the second image is a radiographic (Chest) image (Fig.5.7). Two sets of experiments have been performed. In the first set of experiments, the VQMMC algorithm is applied on the Chest image, for 4-D vectors (formed from blocks of 2 x 2 pixels) with a primary codebook of size 2, for 4-D vectors with a primary codebook of size 4, and for 16-D vectors with a primary codebook of size 16. We call these implementations VQMMC-P2, VQMMC-P4, and VQMMC-P16, respectively. In the second set of experiments, the VQMMC algorithm is applied on the Face (Lena) image, for 4-D vectors with a primary codebook of size 2. The simulations for the two sets of experiments are carried out with different threshold values corresponding to a range of bit rates.

6.3.1 CODING PERFORMANCE

The coding performances are compared using the rate-distortion curves. It is to be noted that even though we have used the mini-max criterion, it is still of interest to use NMSE as comparison. The total bit rate, R_t, for VQMMC consists of two components: one for label transmission R_l, and one for codebook transmission R_c,

$$R_t = R_c + R_l . \qquad (6.2)$$

R_l consists of two components, R_{lp} & R_{ls} corresponding to the primary and secondary codebook labels, respectively. Thus,

$$R_{lp} = (\mathrm{Log}_2 M_p * N_{lp}) / (M \times M), \text{ and} \qquad (6.3)$$

$$R_{ls} = (\mathrm{Log}_2 M_s * N_{ls}) / (M \times M) \qquad (6.4)$$

where,

M x M is the number of pixels in the image,

N_{lp} is the number of labels corresponding to the primary codebook,

N_{ls} is the number of labels corresponding to the secondary codebook,

M_p is the size of the primary codebook, and

M_s is the size of the secondary codebook.

In our experiment, an entropy coder [19-20] is used to transmit the codebook.

The results of the first set of experiments on the Chest image are tabulated in Tables 6.1a, 6.1b, and 6.1c for the VQMMC-P2, VQMMC-P4, and VQMMC-P16, respectively. The corresponding rate distortion curves are shown in Fig. 6.2. The rate distortion curves show the influence of the different vector dimensions, and the different primary codebook sizes on the performance. It can be seen that at low bit rates (0.2 - 0.7 bits/pixel), VQMMC-P16 outperforms VQMMC-P2, which in turn, outperforms VQMMC-P4. However, at medium bit rates (0.7 - 2.0 bits/pixel), VQMMC-P2 and VQMMC-P4 outperform VQMMC-P16. The results of the Chest image coded using the VQRUC-1 and VQIAC techniques (results from chapter 5) are used for comparison. The comparative chart is graphed in Fig. 6.3a. It can be seen that for the Chest image, VQMMC-P2 outperforms the VQRUC-1 and VQIAC techniques over a range of bit rates (0.2 - 1.9 bits/pixel). The reconstructed Chest image at bit rate of 1.36 bits/pixel is shown in Fig. 6.5. The corresponding error image (magnified by a factor of ten) is shown in Fig. 6.7.

The results of the second set of experiments on the Face (Lena) image is tabulated in Table 6.2. The results of the Face image coded using the VQRUC-1 and VQIAC techniques (results from chapter 5) are used for comparison and the comparative chart is graphed in Fig. 6.3b. It can be seen that for the Face image, VQMMC-P2 outperforms VQRUC-1 and VQIAC techniques. The reconstructed Face (Lena) image at bit rate of 1.75 bits/pixel is shown in Fig. 6.6. The corresponding error image (magnified by a factor of five) is shown in Fig. 6.8.

We note that since, the mean square error is used as the distortion criterion, large errors can arise in using VQRUC-1 and VQIAC. This is illustrated in Fig. 6.4a for the Chest image and Fig. 6.4b for the Face (Lena) image, where the distribution of errors greater than 3 and 13, respectively, at a bit rate of 1.8 bits/pixel are graphed.

TABLE 6.1a

Simulation results using VQMMC technique on the Chest image with 4-D vectors and primary codebook of size 2

Bit rate for primary labels Rlp	Bit rate for secondary labels Rls	Bit rate for codewords Rc	Total bit rate Rt	% NMSE
0.22	0.14	0.01	0.37	0.68
0.20	0.28	0.03	0.51	0.32
0.19	0.41	0.04	0.64	0.20
0.17	0.52	0.05	0.74	0.15
0.16	0.63	0.08	0.87	0.11
0.14	0.82	0.09	1.05	0.08
0.12	1.09	0.15	1.36	0.05
0.08	1.49	0.30	1.87	0.03
0.04	1.95	0.79	2.78	0.01

TABLE 6.1b

Simulation results using VQMMC technique on the Chest image with 4-D vectors and primary codebook of size 4

Bit rate for primary labels Rlp	Bit rate for secondary labels Rls	Bit rate for codewords Rc	Total bit rate Rt	% NMSE
0.47	0.06	0.02	0.55	0.59
0.43	0.19	0.03	0.65	0.32
0.40	0.31	0.04	0.75	0.20
0.37	0.43	0.05	0.85	0.15
0.34	0.55	0.07	0.96	0.11
0.30	0.74	0.09	1.13	0.08
0.25	1.01	0.15	1.41	0.05
0.18	1.41	0.30	1.89	0.03
0.09	1.91	0.79	2.79	0.01

TABLE 6.1c

Simulation results using VQMMC technique on the Chest image with 16-D vectors and primary codebook of size 16

Bit rate for primary labels Rlp	Bit rate for secondary labels Rls	Bit rate for codewords Rc	Total bit rate Rt	% NMSE
0.23	0.01	0.10	0.34	0.36
0.23	0.02	0.14	0.40	0.30
0.22	0.05	0.21	0.48	0.25
0.20	0.07	0.26	0.53	0.20
0.17	0.12	0.37	0.66	0.16
0.15	0.16	0.53	0.84	0.12
0.12	0.21	0.77	1.10	0.09
0.09	0.28	1.24	1.61	0.06
0.05	0.31	1.99	2.35	0.03

TABLE 6.2

Simulation results using VQMMC technique on the Face (Lena) image with 4-D vectors and primary codebook of size 2

Bit rate for primary labels Rlp	Bit rate for secondary labels Rls	Bit rate for codewords Rc	Total bit rate Rt	% NMSE
0.17	0.57	0.10	0.84	1.12
0.16	0.67	0.15	0.98	0.86
0.15	0.83	0.21	1.19	0.65
0.14	0.93	0.26	1.33	0.55
0.13	1.02	0.33	1.48	0.45
0.13	1.06	0.37	1.56	0.40
0.11	1.10	0.52	1.75	0.27
0.10	1.20	0.76	2.06	0.20
0.08	1.47	0.98	2.53	0.04

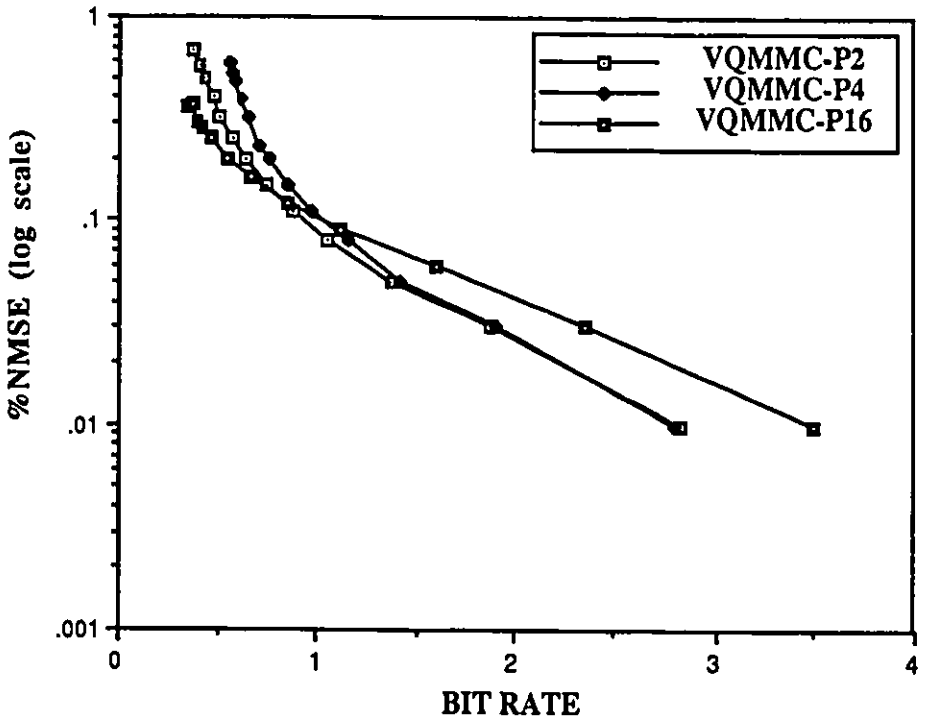

Fig. 6.2 Rate distortion curves for the Chest image using VQMMC technique

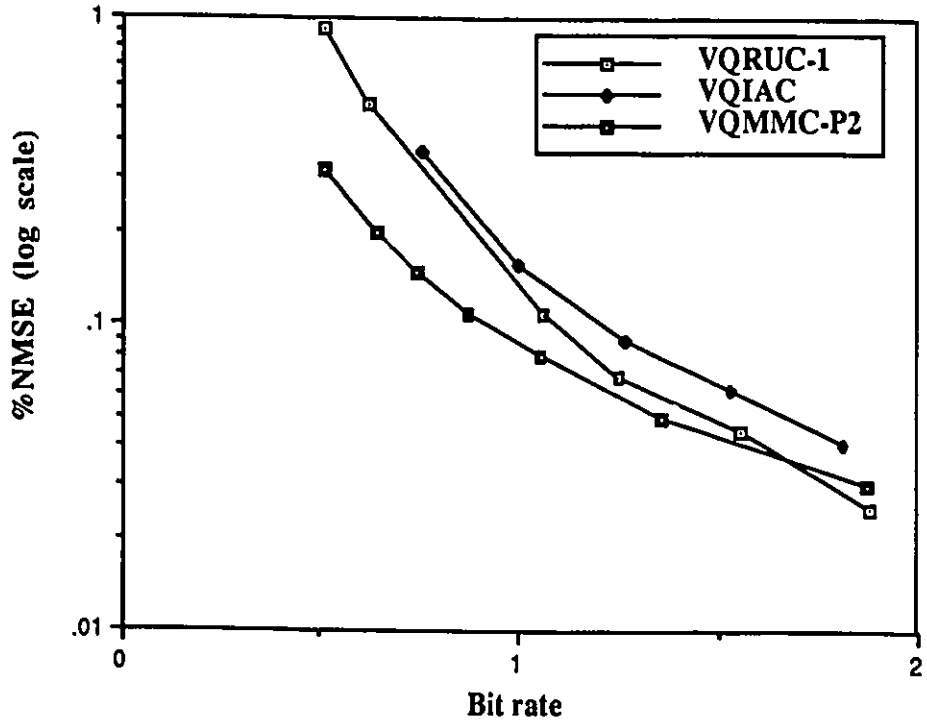

Fig. 6.3a Rate distortion curves for the Chest image

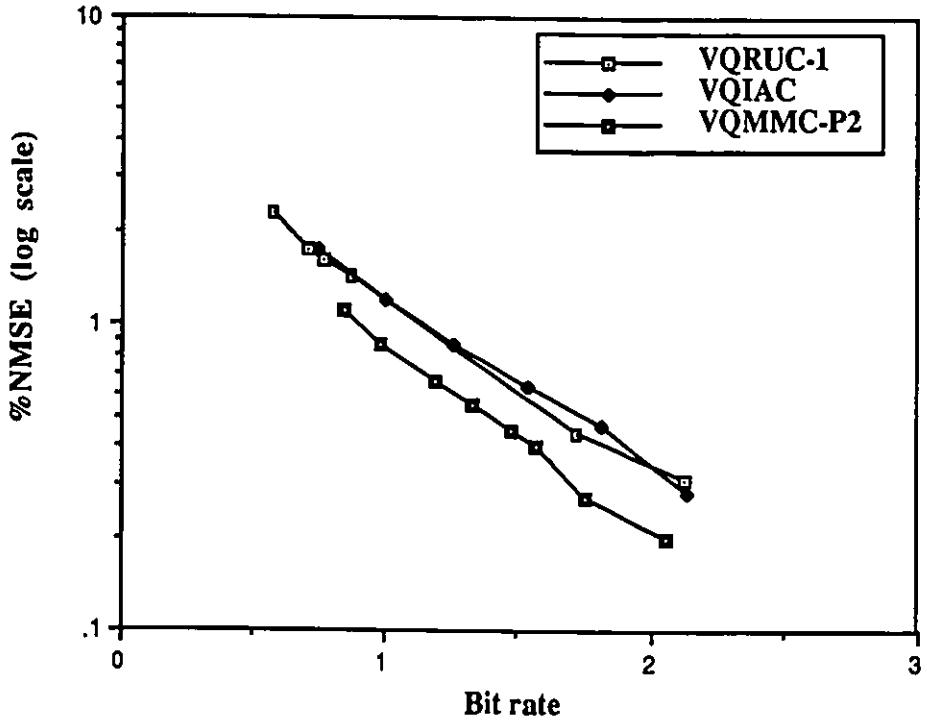

Fig. 6.3b Rate distortion curves for the Face (Lena) image

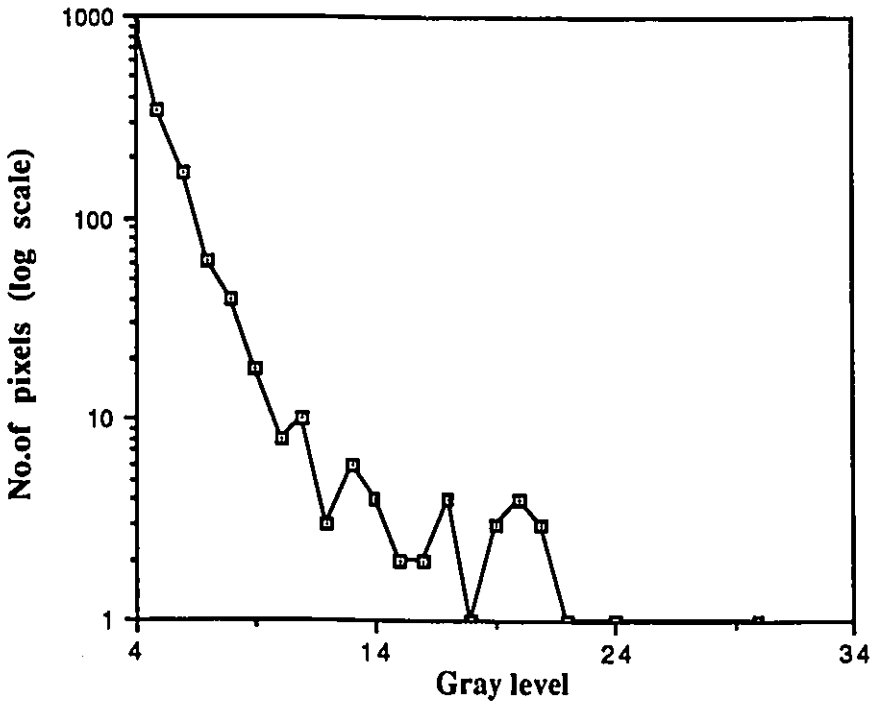

Fig. 6.4a Distribution of errors above threshold for the Chest image using VQIAC at a bit rate of 1.8 bits/pixel

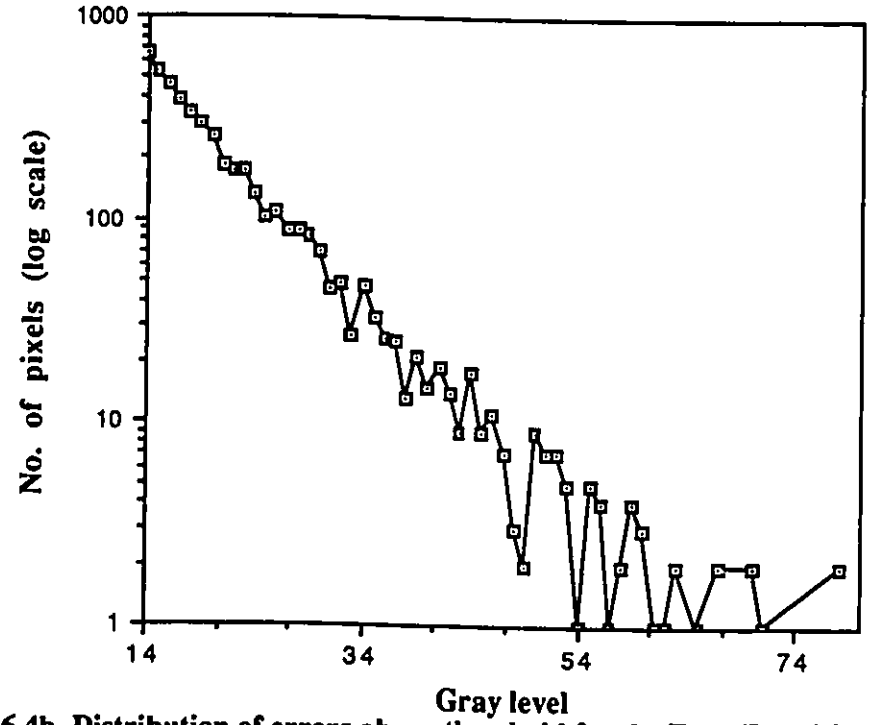

Fig. 6.4b Distribution of errors above threshold for the Face (Lena) image using VQIAC at a bit rate of 1.8 bits/pixel

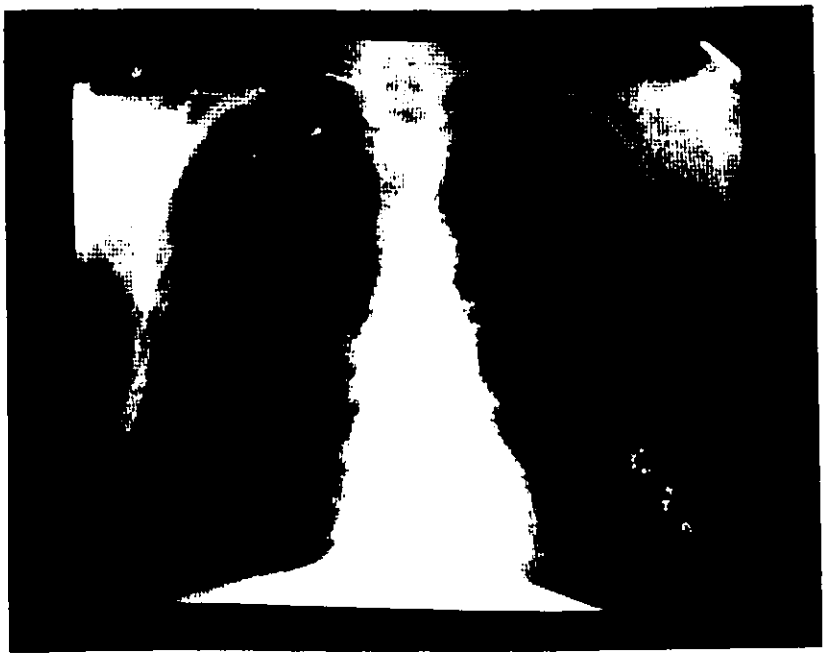

Fig. 6.5 The reconstructed Chest image using VQMMC with 4-D vectors at a bit rate of 1.36 bits/pixel

Fig. 6.6 The reconstructed Face (Lena) image using VQMMC with 4-D vectors at a bit rate of 1.75 bits/pixel

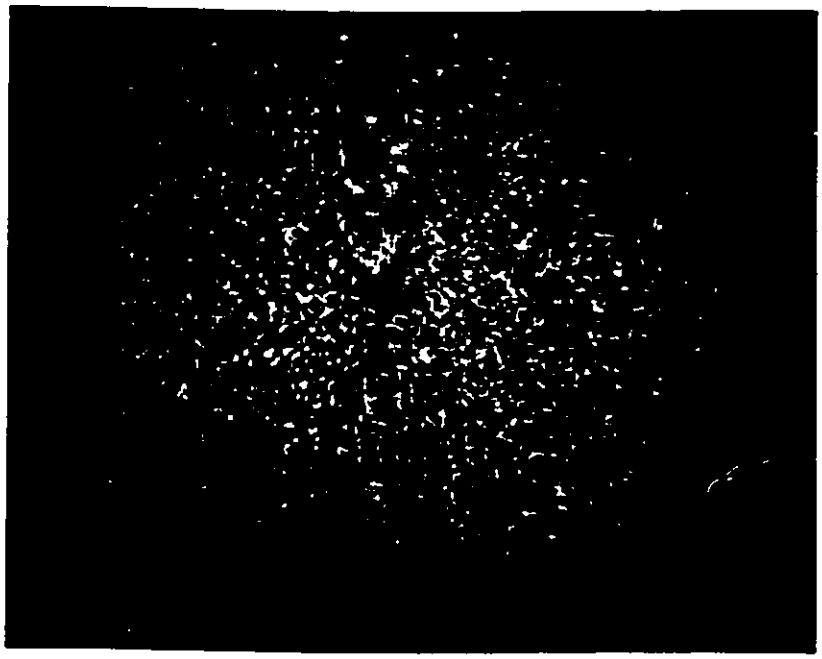

Fig. 6.7 The error Chest image using VQMMC with 4-D vectors at a bit rate of 1.36 bits/pixel

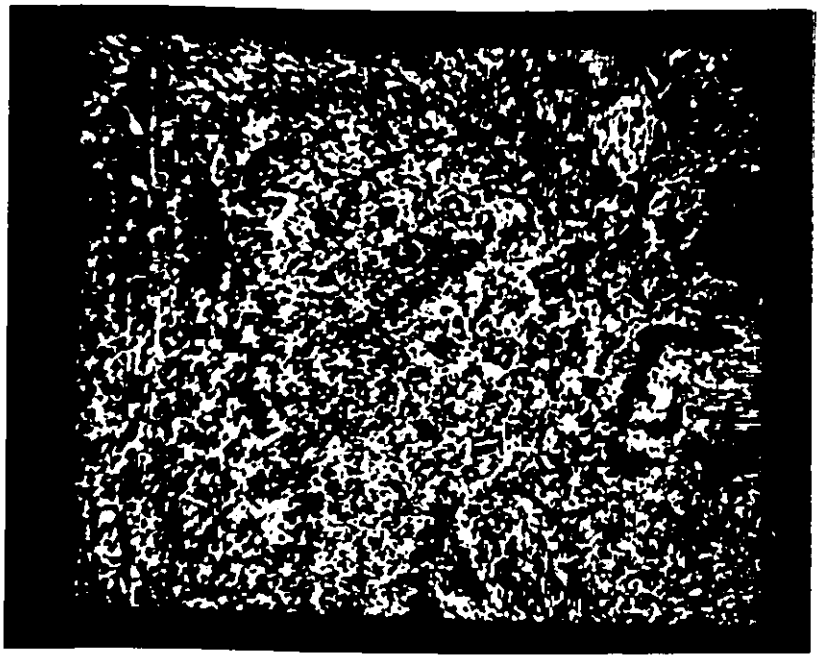

Fig. 6.5 The error Face (Lena) image using VQMMC with 4-D vectors at a bit rate of 1.75 bits/pixel

6.3.2 DISCUSSION OF RESULTS

Table 6.3 qualitatively compares the three VQMMC implementations on the Chest image. Note that a scale of ten is used to express the relative performance, where a value of ten corresponds to the best performance.

TABLE 6.3

Performance comparison chart for the Chest image (a scale of 10 is used to express the relative performance)

Algorithm	Low bit rate 0.2 - 0.7 bpp	Medium bit rate 0.7 - 2.0 bpp
VQMMC-P2	5 - 8	10
VQMMC-P4	3 - 7	8 - 10
VQMMC-P16	10	5 - 8

It can be observed from Tables 6.1a - c & 6.3, that at low bit rates, VQMMC-P16 provides the best coding performance resulting from the smaller bit rate requirement for secondary label transmission. On the other hand, VQMMC-P2 outperforms VQMMC-P4 because of the smaller bit rate for the primary label transmission. It can also be seen at medium bit rates that VQMMC-P2 provides the best coding performance which results from the smaller bit rate for codeword transmission.

It can be seen from the rate distortion curves in Fig. 6.3a that VQMMC provides a performance improvement of 25-60% over VQIAC, and 10-65% (up to 1.5 bits/pixel) over VQRUC-1. The important point is the simplicity of the VQMMC technique, where the codebook is generated on the fly and hence real-time implementation is feasible. In fact, the algorithm can be easily implemented using a stack based architecture[77]. The reconstructed Chest image shown in Fig. 6.5 illustrates that very good quality images, which are subjectively better than that using the VQRUC-1 technique (Fig. 5.9) can be obtained.

It can be seen from the rate distortion curves in Fig. 6.3b that VQMMC provides a

performance improvement of 35-80% over VQIAC and VQRUC-1. The reconstructed Face image shown in Fig. 6.6 illustrates that very good quality images, which are subjectively better than that using the VQRUC-1 technique (Fig. 5.10 & 5.11) can be obtained.

The significant advantage of VQMMC over VQRUC-1 and VQIAC is the absence of large errors. This is illustrated in Fig.6.4a for the Chest image, where the distribution of errors at a bit rate of 1.36 bits/pixel for the VQIAC technique shows that more than 1% of the pixels have errors above 3(threshold value). In fact, there are pixels with errors as high as 30 which is equivalent to 12% of the gray level range. This is also illustrated in Fig. 6.4b for the Face (Lena) image, where the distribution of errors at a bit rate of 1.75 bits/pixel for the VQIAC technique shows more than 6% of pixels have errors more than 13(threshold value). In this case, there are errors as high as 78 which is equivalent to 30% of the gray level range. These large errors can result in poor rendition of the edges. This is illustrated by the error Chest image (magnified by a factor of ten) and error Face image (magnified by a factor of five) shown in Fig. 5.12-5.14. On the other hand, the error image shown in Fig. 6.7 (magnified by a factor of ten) & 6.8 (magnified by a factor of five) for the VQMMC technique, illustrate the uniform distribution of errors (no peaks near edges) within the threshold value.

6.4. SUMMARY

We have presented an intra-image adaptive technique which employs the mini-max error criterion for image compression using vector quantization. The conventional method of coding an image using a universal codebook does not guarantee the closest codewords to be within a prespecified bound of the input vectors. This can result in large errors which may give rise to artifacts. The proposed technique employs a criterion that minimizes the maximum error. Here, the codebook is generated on the fly from the input vectors to be coded. Both the transmitter and receiver maintain identical codebooks and hence keep track of the changes without any overhead information. Simulation results demonstrate the excellent coding performance of the technique and the absence of large errors. As it is a single-pass technique, real-time implementation is possible.

7. SYSTOLIC ARRAY ARCHITECTURE FOR VECTOR QUANTIZATION

7.1. INTRODUCTION

In this chapter, we present a systolic array architecture to implement VQ. We recall from section 2.4 that the high-speed architectures for VQ reported thus far in the literature[7, 13-16] only implement image encoding but not codebook generation. Since the codebook generation algorithm essentially involves several iterations of the encoding algorithm (Eqn. 2.21) followed by centroid computation (Eqn. 2.22), these architectures can also be extended to implement the codebook generation operation [80]. However, the following two remarks are in order. First of all, we require (i) additional hardware (back-end processor) to compute the new centroids at the end of each iteration of the encoding algorithm, and (ii) a high-speed interface to transfer the new centroids into the processors for the next iteration of the encoding algorithm. Secondly, and most importantly, the centroid computation is executed only after the encoding operation is completed, which is then followed by the transfer of the new centroids. This involves a substantial overhead on the execution time of the LBG algorithm for codebook generation. We recall from section 3.4.1, that in the systolic system, the data flow may be at multiple speeds in multiple directions [61]. We propose a systolic array architecture, where the encoding and centroid computation operations are overlapped in the same structure.

A basic systolic cell is designed with two modes (forward and reverse) of operation. In the forward mode, the cell executes the basic operation in a VQ encoder, namely, distortion computation. In the reverse mode, the cell executes the centroid computation operation. An array of L x N cells are connected in parallel and pipeline in the directions of vector dimension, L, and codeword dimension, N, respectively. The input vectors to be encoded are passed through the cells of the array. The distortion values computed in the array in the forward mode, are fed to a comparator unit which determines the index (label) of the codeword with least distortion. This results in a speed-up factor of L*N making possible VQ encoding in real-time. The input vectors

along with its label are routed back into the cell array in the opposite direction (reverse mode) where the cells execute the partial computation of the new centroids. The two modes of operation take place simultaneously with a delay buffer used to synchronize the operations. The regular and iterable structure makes possible the VLSI implementation of the architecture.

In the following section, the systolic array architecture for VQ is presented. In section 7.3, the analysis of execution time is detailed. The summary is then presented in section 7.4.

7.2. SYSTOLIC ARRAY ARCHITECTURE FOR VQ

7.2.1 BASIC SYSTOLIC CELL

The basic systolic cell for VQ is shown in Fig. 7.1. It has two modes of operation, (i) forward mode, and (ii) reverse mode. The cell executes the basic distortion operation, and the partial computation of the new centroids, in the forward and reverse mode, respectively.

Forward mode : We recall the expression for the basic distortion operation in equation 2.27

$$d(V_i , C_p) = \frac{1}{L} \sum_{j=1} [(V_{i,j} - C_{p,j})^2]$$

This can be expressed in a recursive form as follows:

$$d(V_{i,j} , C_{p,j}) = d(V_{i,j-1} , C_{p,j-1}) + 1/L[(V_{i,j} - C_{p,j})^2] \qquad (7.1)$$

where, $d(V_{i,j-1} , C_{p,j-1}) = 0$ for $j=1$

The basic systolic cell for VQ is designed to implement the distortion operation in equation 7.1 (for one value of the index j) in the forward mode. Hence, the basic systolic cell must execute the three basic operations: (i) absolute magnitude difference function , (ii) squaring , and (iii) accumulation. In Fig. 7.1, the input to the cell, $\mathbf{V_{in}}$, is an element of the input vector and the cell has the corresponding element of the codeword stored in the RAM as C. $\mathbf{V_{in}}$ is compared with C and the difference, which is the error, is squared and added to yield the distortion input, $\mathbf{D_{in}}$, to give the distortion output, $\mathbf{D_{out}}$. The input value of the vector element is also sent to the output as $\mathbf{V_{out}}$ which serves as input to the next cell in the array.

Reverse mode : The centroid computation operation is also executed in parallel in the direction of L, in the reverse mode by the basic cell. This operation is executed after the label of the vector is

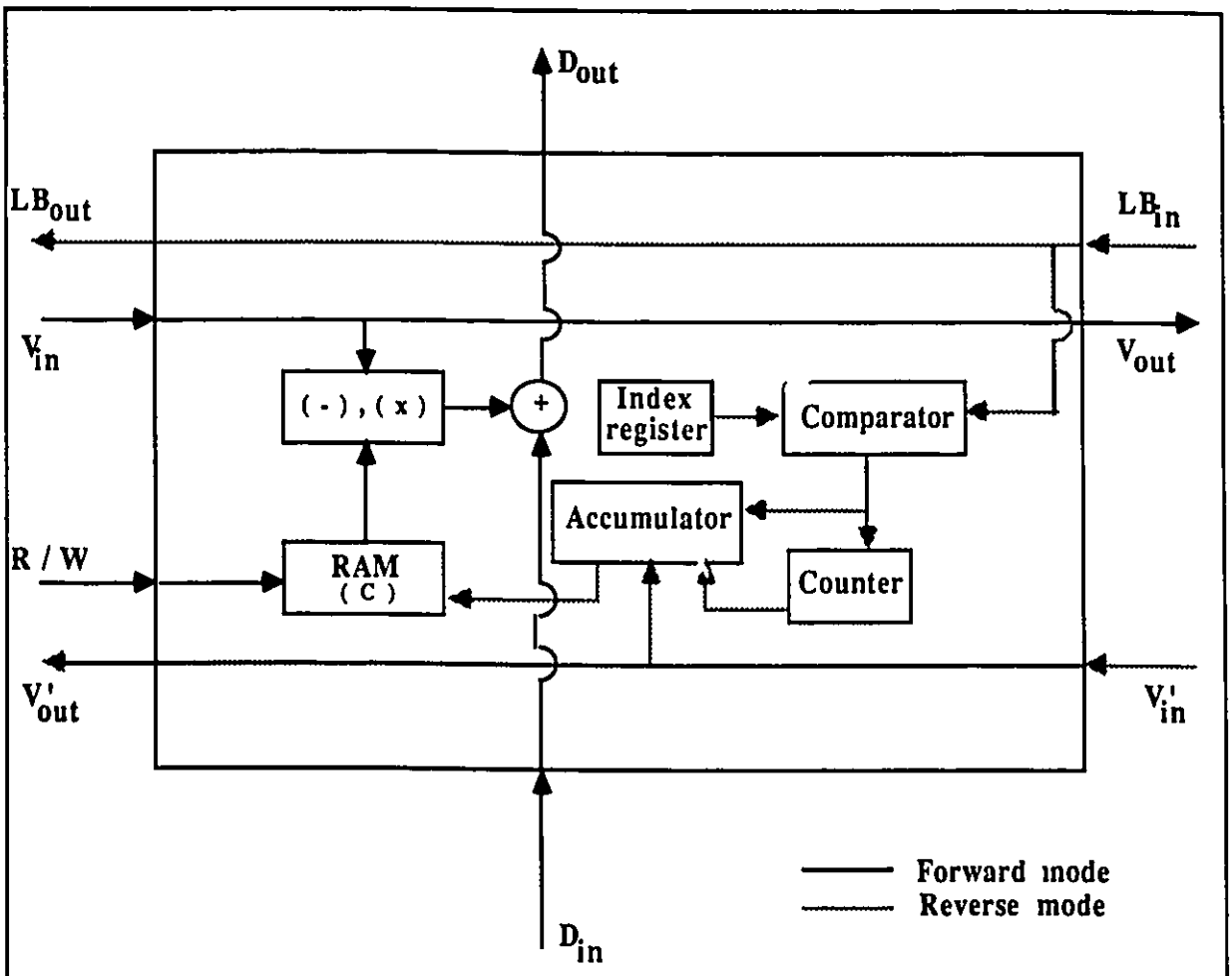

Fig. 7.1 Basic systolic cell for vector quantization

determined. The input value of the vector element is stored in a buffer until the label is determined and is then fed back into the array as, V'_{in} (in the reverse mode). The label of the vector, LB_{in}, is also fed into the input of the cell. This is compared (in the comparator) with the the index of the codeword, available in the index register of the cell. If there is a match, the comparator sends a match signal, which enables the accumulation of the vector element to the contents of the accumulator and the counter to be incremented. The input value of the vector element and the label are sent to the output as V'_{out} and LB_{out}, respectively .

7.2.2 THE SYSTOLIC ARRAY ARCHITECTURE

We now present the systolic array architecture for VQ. The array consists of LN basic cells, connected in parallel and pipeline in the directions of L and N, respectively as shown in Fig. 7.2.

Forward mode : The systolic array executes the encoding algorithm in its forward mode of operation. Here, each input vector element, $V_{i,j}$ (i = 1, 2,....., K; and j = 1, 2, ..., L), is compared with the corresponding element of each codevector, $C_{p,j}$ (p = 1, 2, ..., N) stored in the array (the RAM's are in the read mode). We note that in this mode, the elements of the vector are pumped into the array at intervals of one clock cycle. The distortion, $D_{p,i}$, is the total distortion value of the i^{th} input vector when matched with the p^{th} codeword, C_p. The total distortion values of the i^{th} input vector when matched with all the codewords (p = 1, 2, ...N), are then fed to a comparator which sends out the label of the codeword with least distortion.

A global clock signal serves to synchronize the operations of the cells in the array. The label of the first input vector becomes available after N+L+2 clock cycles which is the latency of the system for the encoding operation. The labels for subsequent input vectors become available at intervals of one clock cycle. The sequence of operations at different clock cycles is best understood by the cell occupancy diagram (Fig. 7.3) which shows the cells occupied by the elements of the vector at different instants of time. The path traced by the vector { $V_{1,j}$ } is shown in Fig. 7.3; subsequent vectors trace the same path with a delay of one clock cycle.

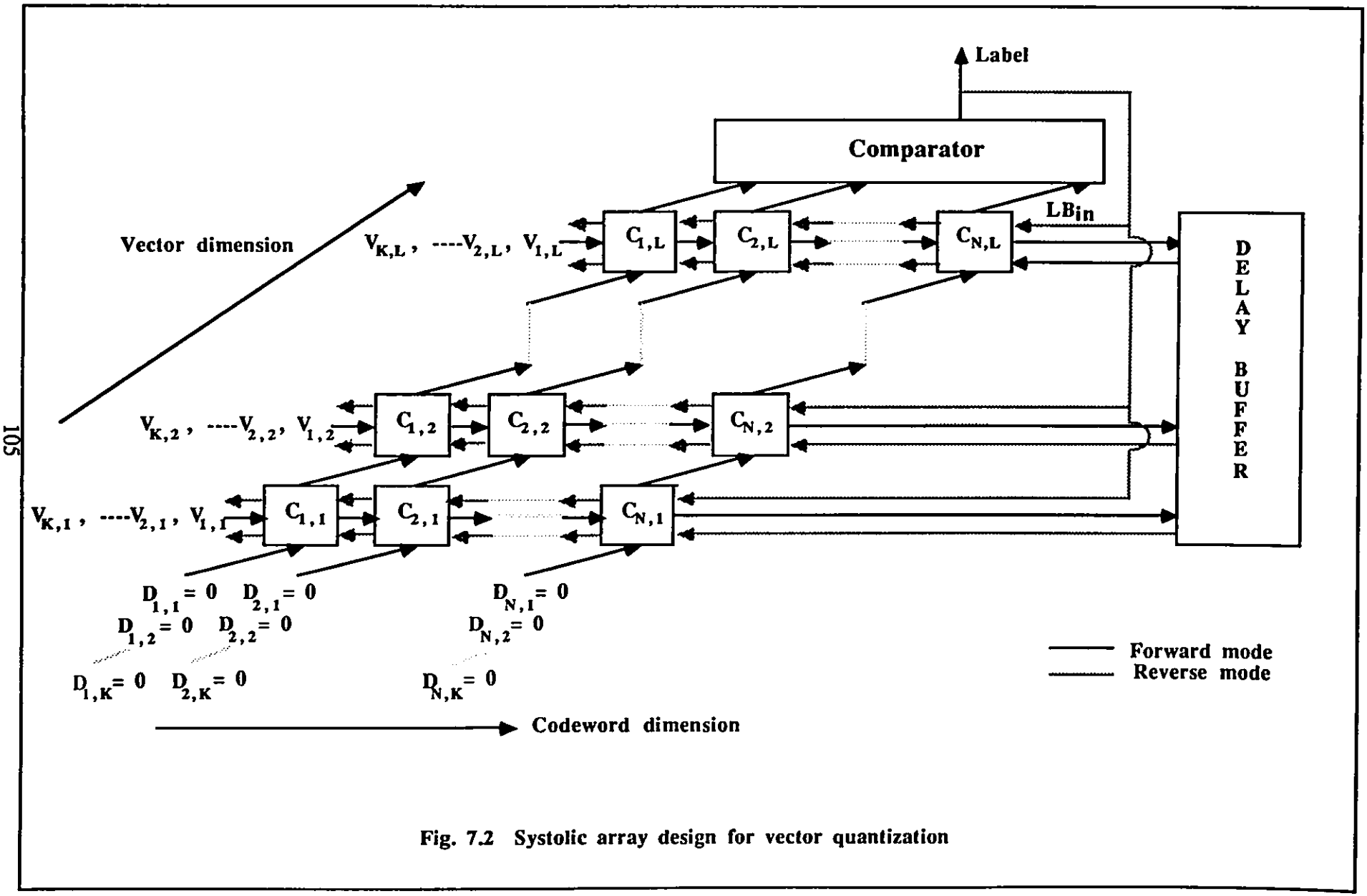

Fig. 7.2 Systolic array design for vector quantization

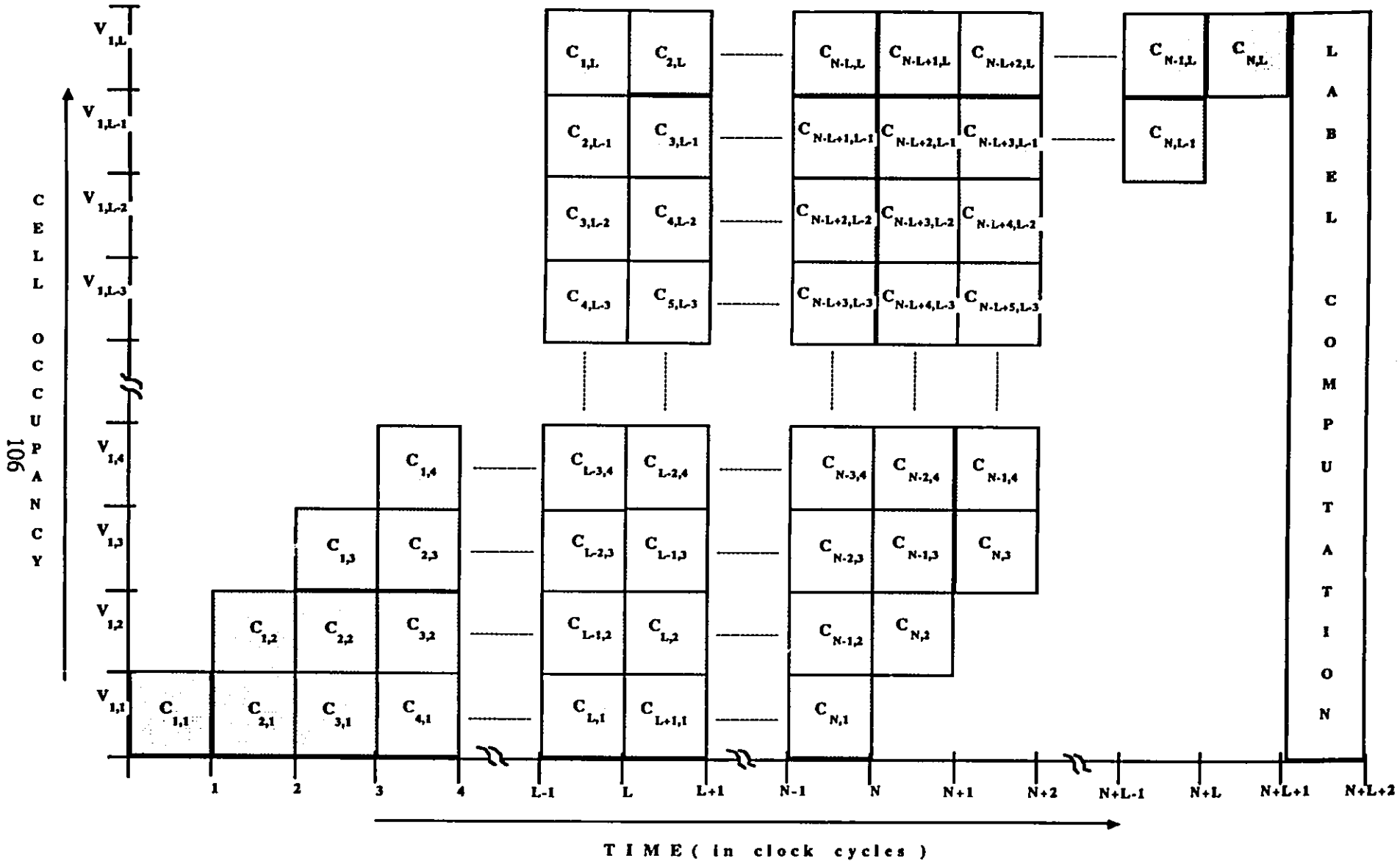

Fig. 7.3 Cell occupancy Vs time in the forward mode

Reverse mode : The reverse mode is used to compute the new centroids (new codewords). In this mode, the vector, $V_{i,j}$, is fed back into the array along with the label, LB_i. Since the label of the vector is determined only after the last element, $V_{i,L}$ of a vector, gets compared with the last element of the last codeword, $C_{N,L}$, the elements of the vector have to be stored (delayed) in order to recirculate them back into the array. It is assumed that the comparator executes the label determination in one clock cycle. Hence, the first element of the vector has to be delayed L+1 clock cycles, the second element L clock cycles, and so on, until the last element which has to be delayed one clock cycle. This implies that the delay buffer should contain (L+1)(L+2)/2 delay elements organized as shown in Fig. 7.4.

The global clock signal synchronizes the operations of the cells in the array during both modes. The duration of the clock pulse is determined by the maximum of (i) the time taken by the basic cell to compute the distortion, and (ii) the time taken to compute the comparison and accumulation operation. The sequence of operations at different clock cycles in both modes of operation is represented by the cell occupancy diagram in Fig. 7.5 which shows the path (both forward and reverse) traced by the vector, $\{ V_{i,j} \}$. The path traced by the vector in the forward mode in the first N-1 cycles is similar to that shown in Fig. 7.3. We note that N+2L+2 cycles after entering the array, the first vector completes its path through one iteration of the LBG algorithm. Subsequent vectors trace the same path but delayed by one clock cycle. Thus, at the end of K+N+2L+2 cycles, one iteration of the LBG algorithm is executed. At this instant, the accumulators of the cells corresponding to a codeword contain the sum of the elements of the vectors assigned to the codeword. The corresponding counters indicate the actual number of vectors assigned to the codeword. The centroid or average can the be easily calculated and is then written into the RAM location as the new codeword. The above procedure (iteration) is repeated until convergence is reached.

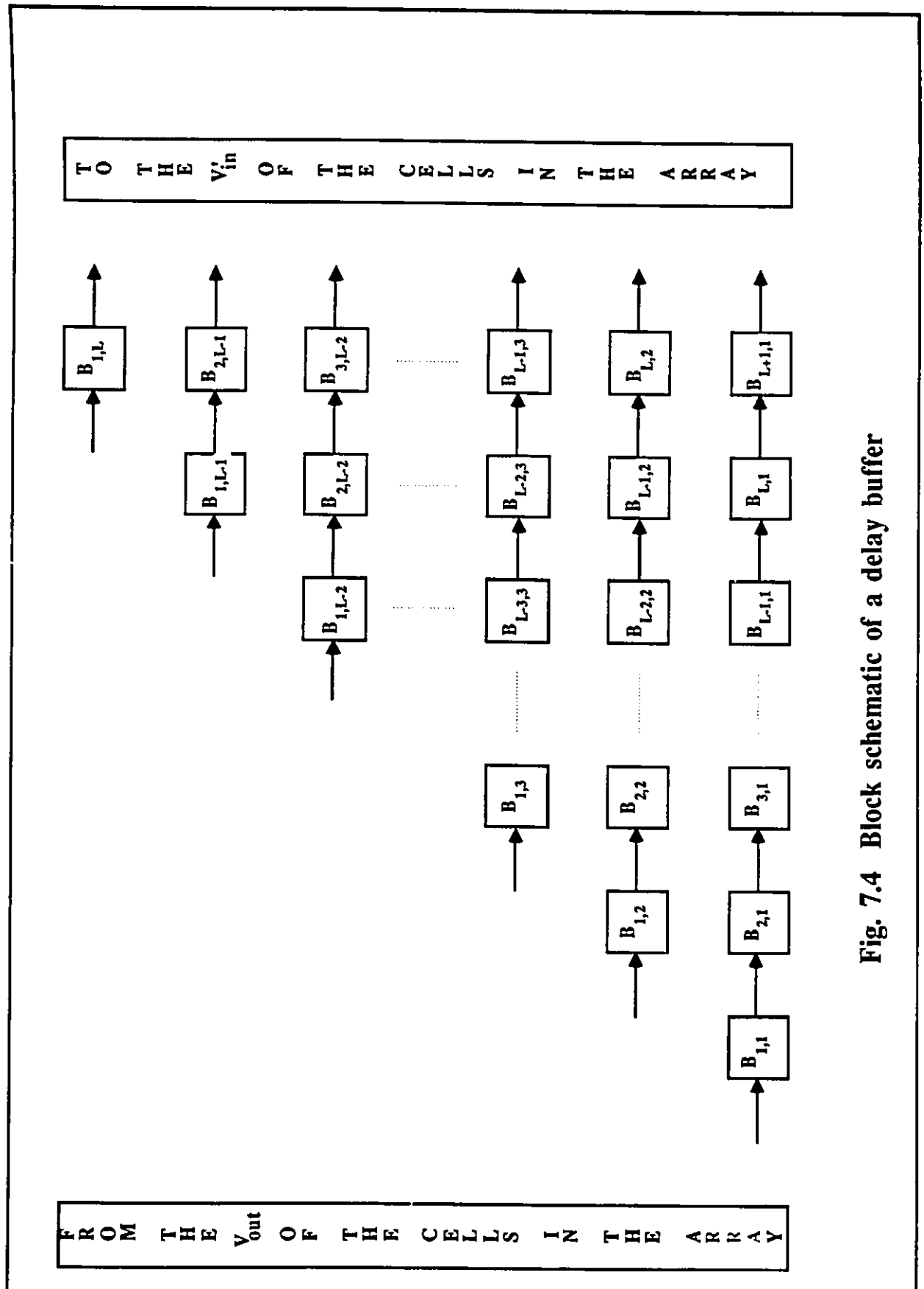

Fig. 7.4 Block schematic of a delay buffer

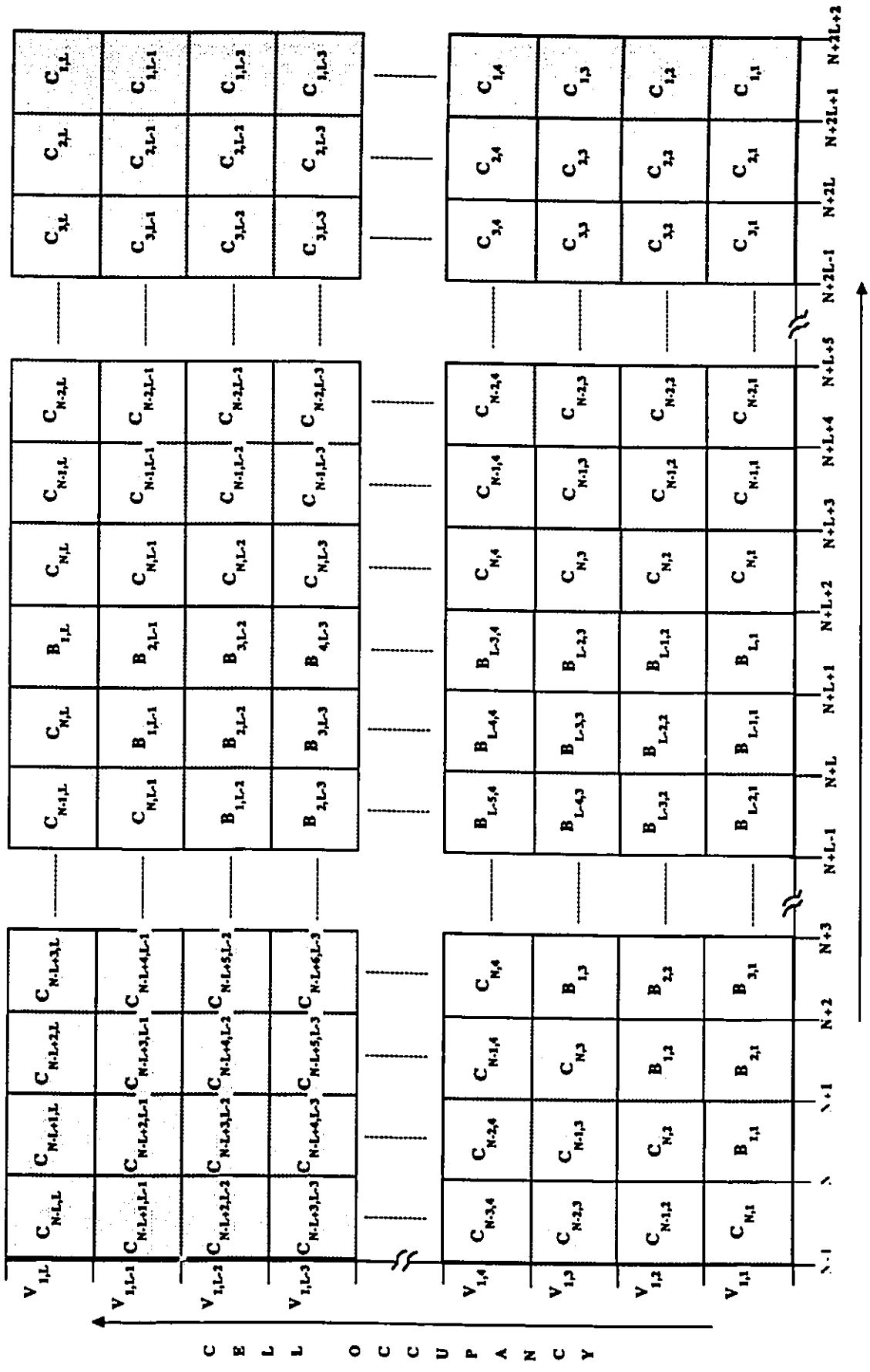

Fig. 7.5 Cell occupancy Vs time chart for both modes

7.3. EXECUTION TIME

We now calculate the execution time, T_{en}, for the encoding algorithm using an SISD implementation (S_p (1)). We recall from section 2.5.2 that the computational complexity of the encoding operation for K vectors of dimension L and a codebook of size N is K*L*N basic operations. Hence, the execution time using an SISD implementation is given by,

$$T_{en(SISD)} = (K*L*N* T_e / S_p(1)) = K*L*N* T_e \text{ time units} \qquad (7.2)$$

where, T_e, is the time of execution of the basic operation. The speed-up using the systolic array in the forward mode for the encoding operation is Sp(LN) = L * N, since the parallelism in the directions of L and N are exploited. Hence, the execution time using this implementation is as follows.

$$T_{en} = (K*L* N*T_e / S_p(LN)) + T_l = (K*L*N*T_e / L*N) + T_l \qquad (7.3)$$

where, T_l is the latency of the systolic array. We recall from section 7.2.2 that T_l = N+L+2 clock cycles. Hence,

$$T_{en} = (K*L*N*T_e / L*N) + (N+L+2)* T_e$$

$$= (K+N+L+2)* T_e \text{ time units.} \qquad (7.4)$$

Typical values for L, N, K, and T_e for a 512 x 512 image with 16-D vectors and a codebook of size 256 (which is equivalent to a bit rate of 0.5 bits/pixel and 0.63 bits/pixel for the VQUC and VQIAC techniques, respectively) are, 16, 256, 16384, and 100 ns [10-13] respectively. Hence, T_{en} is equal to 1.67 ms which satisfies the real-time requirement for encoding (< 33 ms). If the architectures reported in the literature (for encoding) are used for codebook generation with I iterations, then the total execution time is as follows.

$$T_g = (T_{en} + T_{cen} + T_{tr})*I \text{ time units.} \qquad (7.5)$$

where,

T_{cen} = execution time for centroid computation of complexity O(KL) using a back-end processor ≈ $K*L*T_e$ time units, and

T_{tr} = time for transferring the new centroids into the array = N*L / speed of interface.

Assuming that the centroid computation is also executed in parallel in the direction of L (vector dimension), T_{cen} = $K*T_e$ time units = 1.63 ms. The time for transferring the new centroids into the array, T_{tr} , is calculated by assuming that a 1M byte/sec interface is used to transfer the centroids. Hence, T_{tr} = 4 ms. The generation of an optimal codebook involves several iterations of the LBG algorithm (typically 100-200 iterations). Assuming 100 iterations of the LBG algorithm, the execution time , T_g = 730 ms.

However, in the case of the proposed architecture, the centroid computation is overlapped with the encoding operation. Moreover, there is no need for transferring the new centroids into the array. This makes possible fast execution of the codebook generation algorithm. The execution time is given by,

$$T_{gp} = T_{en} * I \qquad (7.6)$$

Hence, T_{gp} = 167 ms.

The values of T_{en} , T_{cen} , T_{tr} , T_g , and T_{gp} corresponding to 256 x 256, 512 x 512 and 1024 x 1024 images for a codebook of size 256, are tabulated in Table 7.1. It can be clearly seen that real-time codebook generation (< 33 ms) is not feasible.

However, the proposed architecture makes possible the generation of sub-optimal codebooks (assuming 10-20 iterations of the LBG algorithm) in real-time, which is not attainable using the architectures reported in the literature.

7.4. SUMMARY

In this chapter, we have presented a systolic array architecture to implement VQ. A basic systolic cell was designed with two modes of operation. In the forward mode, the cell executes the basic distortion operation while, in the reverse mode, the cell executes the centroid computation operation. Both modes co-exist in perfect synchronism. The cells are interconnected in parallel and pipeline in the directions of L and N, respectively. This architecture has the following advantages: (i) there is no need for separate hardware to compute the new centroids; (ii) there is no need for a

TABLE 7.1

Execution time for codebook (size = 256) generation for different image sizes

IMAGE SIZE	T_{en}	T_{cen}	T_{tr}	T_g^{10}	T_{gp}^{10}	T_g^{100}	T_{gp}^{100}
256 x 256	0.44 ms	0.41 ms	4 ms	48.5 ms	4.4 ms	485 ms	44 ms
512 x 512	1.67 ms	1.63 ms	4 ms	73.0 ms	16.7 ms	730 ms	167 ms
1024 x 1024	6.58 ms	6.55 ms	4 ms	131.3 ms	65.8 ms	1313 ms	658 ms

T_{en} = Execution time for the encoding.

T_{cen} = Execution time for centroid computation.

T_{tr} = Time for transfer of new centroids.

T_g^I = Execution time for codebook generation using the architectures reported in the literature = $\{ T_{en}+T_{cen}+T_{tr} \}*I$

T_{gp}^I = Execution time for codebook generation using the proposed architecture = T_{en} *I

high-speed interface to transfer the new centroids into the systolic array; and (iii) there are no delays involved in the computation and transfer of new centroids. We note that the primitive cell design can be simplified, avoiding the repetition of the comparators, index registers and counters in the L dimension by designing a separate module with the three components. The label comparison takes place in this module and if a match is obtained, the signal is then sent to the accumulators of all the cells corresponding to that codeword. This architecture can be used to implement the VQUC, VQRUC, VQIAC and VQRUC-1 algorithms presented in chapter 5. Although the proposed design can implement the encoding and codebook generation algorithms, only the coding algorithm and sub-optimal codebooks can be executed in real-time. Real-time codebook generation (optimal codebooks), therefore, requires us to exploit parallelism in other dimensions and this concept is the basis of the design to be discussed in the following chapter.

8. CONTENT-ADDRESSABLE MEMORY ARCHITECTURE FOR VECTOR QUANTIZATION

8.1 INTRODUCTION

In this chapter, an architecture suitable for real-time image coding using adaptive vector quantization is presented. This architecture is based on the concept of content-addressable memory (CAM). We recall from section 3.4.3 that a promising architecture with a high degree of parallelism is a content-addressable memory CAM [63,64], which consists of a collection of cells that can be searched simultaneously and in parallel [64,65] on the basis of their contents. The use of a CAM for both exact and inexact matching of patterns [66-69] has also been discussed in section 3.4.3. VQ can be viewed as a pattern matching process where each input pattern (vector) is compared with a finite set of templates (codewords)[2]. Hence, a CAM is very efficient to implement VQ and the resulting parallelism is essentially proportional to the number of elements stored in the array.

A CAM architecture to implement VQ based upon inexact pattern matching combined with a varying threshold is presented in this chapter. The most straightforward approach is to set the codewords as templates stored in the CAM array and the input vectors as input patterns (search argument); in other words, matching is performed from the perspective of input vectors. The speed-up obtained with this approach is $S_p(LN)$, which is similar to that reported in the literature for parallel and pipelined architectures [10,11] and is not sufficient to implement codebook generation in real-time. However, as K>>N for image coding, a greater degree of parallelism can be obtained by employing parallelism in the directions of L and K. A novel approach is introduced based on the concept of inverting the order of comparison in VQ: the input vectors are set as templates stored in the CAM array and the codewords are set as input patterns (search argument); in other words, matching is performed from the perspective of the codewords. This approach results in a large speed-up, $S_p(KL)$. We note that in using the CAM, the conventional MSE measure is substituted by the absolute difference measure. This measure results in little

degradation and, in fact, we show that this limits large errors. The reductions in computational complexity (the large speed-up), coupled with the gains in the execution time for the basic distortion computation, results not only in real-time image encoding but also real-time codebook generation (< 33 milliseconds). Hence, image adaptive VQ techniques become feasible. The regular and iterable nature of the CAM architecture is particularly well suited for VLSI implementation.

In the following section, the content-addressable memory based architecture for implementing vector quantization in real-time is presented. In section 8.3 we provide some simulation results, and a detailed analysis of complexity issues. The summary is then presented in section 8.4.

8.2. CAM DESIGN FOR VECTOR QUANTIZATION

We recall from chapter 2, that in VQ, encoding an input vector essentially involves comparing the vector with a set of stored codewords. For each input vector, the closest codeword is found and the corresponding label is used to represent the vector. The coding (quantization) operation is conventionally done in a sequential manner (SISD machine implementation) where, each input vector of the image to be coded is compared with the codewords, one at a time. We recall from section 2.5.2 that for K input vectors of dimension L, and N codewords, the complexity is O(KLN). Let us consider some values reported in the literature for vector quantization of images. A vector dimension L=16 is often used, as smaller values result in an increased bit rate, while larger values result in an increased encoding complexity [45]. Image sizes range from 256 x 256 for lower quality images to 2000 x 2000 for high quality medical images: the corresponding values for the number of input vectors, K, range from 4096 to 250,000. As for the codebook size, N, three values frequently quoted are, 256, 512, and 1024 corresponding to bit rates (for VQUC) of 0.5 bits/pixel, 0.63 bits/pixel and 0.69 bits/pixel, respectively.

In this section, we show how VQ can be implemented using a CAM, where the resulting parallelism is essentially proportional to the number of elements stored in the array. The most straightforward approach is to set the codewords as templates stored in the CAM array and the input vectors as input patterns (search argument). We call this approach VQ_CAM. If the CAM stores the entire codeword, then the speed-up obtained with this approach is $S_p(LN)$, which is

similar to that reported in the literature for parallel and pipelined architectures [10,11].

A second approach presented to using a CAM based architecture builds upon the novel concept of inverting the order of comparison in VQ: the input vectors are set as templates stored in the CAM array and the codewords are set as input patterns (search argument). We call this approach VQ_CAM_INVERT. If the entire vector is stored in the CAM array, then the speed-up is now $S_p(LK)$. Since K>>N, a much greater speed-up is obtained using the VQ_CAM_INVERT as compared to VQ_CAM.

Before detailing the CAM based architecture, we introduce the concept of *Gating*, which is the basis for implementing the basic operation in VQ, namely finding the closest codeword. To facilitate the explanation, the basic concept is introduced for the one dimensional case. In extending the results to L dimensions, we adopt a distortion measure based upon the absolute differences, since the usual MSE measure (Eqn. 2.18) cannot be readily implemented in a CAM architecture. We show by simulation studies in section 8.3 that there is little degradation in performance.

8.2.1 GATING

The output of the Inexact_Match procedure presented in section 3.4.3 gives all the templates which are within the threshold (Delta) of the input pattern. If the correct value of the threshold is set, then the procedure will give a unique response, i.e., the input pattern is assigned to the closest template. If too large a value is set, then the procedure will give more than one response, i.e., the input pattern is assigned to more than one template. On the otherhand, if too small a value is set, the procedure will give no response at all, i.e., the input pattern is not assigned to any template. We now introduce a multiple pass technique called *Gating* in which the Inexact_Match procedure is applied with monotonically increasing threshold values, starting with a threshold value of 0. This technique guarantees that each input pattern is assigned to the closest template; the corresponding indices are stored in an index register, IR_i (in the result store module). In the event that there is not a unique closest template, Gating results in a multiple response, one of which can be arbitrarily chosen. In procedural form, the Gating algorithm for a set of input patterns is as follows.

```
PROCEDURE Gating ( In: Set of Input_patterns,Templates; Out: Index_register )
  BEGIN
    DO ( For each Input_pattern 1, ..., K )
    Initialize Response_register, R_i = 0, for i = 1, ..., N
    Initialize Delta (threshold) to 0.
      WHILE ( Closest template not assigned to the Input_pattern {R_i = 0, for all i } )
        BEGIN
          Inexact_Match ( In: Input_pattern,Templates, Delta ; Out: Response_
                                                                    register )
          IF < multiple response > THEN
            Closest template := First response
          END IF
        END
        Increment Delta
      END WHILE
    Index_register := Index of the closest template
    END DO
  END
END PROCEDURE
```

The index register now contains for each input pattern, the index of the closest template.

8.2.2 VQ_CAM

We now show how a CAM can be used to execute the encoding algorithm, i.e., finding for each input vector, the closest codeword. The codewords are set as templates and the input vectors to be coded are set as the input patterns (search argument). For each input vector, the codewords are searched in parallel to determine the closest codeword. To illustrate the concept, we present the the VQ_CAM procedure for 1-D vectors.

PROCEDURE VQ_CAM (In: Input_vectors, Codewords; **Out:** Labels)

Initialize Templates := Codewords

Initialize Input_Patterns := Input_vectors

BEGIN

Gating (**In:** Input_patterns,Templates; **Out:** Index_register)

Label of the Input_pattern = Index of the closest template

END

END PROCEDURE

For the case of L-D vectors , L separate CAM modules are required; which results in a speed-up $S_p(LN)$. We note that in order to compute the effective speed-up, we have to factor in the extra passes for the (i) *"within Δ"* search, (b passes for a b-bit word) and (ii) *Gating* operation (a maximum of 2^b passes for a 2^b gray level resolution). Hence, the effective speed-up is $S_p(LN/2^b b)$.

8.2.3 VQ_CAM_INVERT

We now present an alternate CAM based architecture for implementing VQ in which the input vectors to be coded are set as templates stored in the CAM array and the codewords are set as the input patterns (search argument). As the role of the two sets of parameters is inverted with respect to VQ_CAM, we call this approach VQ_CAM_INVERT. Since, K>>N in our example for image coding, this approach yields a greater speed-up, $S_p(LK)$, compared to, $S_p(LN)$, using the VQ_CAM approach.

The codewords are now considered one at a time and are compared in parallel with the set of input vectors. Consider the possible outcomes when the Inexact_Match procedure is applied to the set of input vectors for some fixed codeword (Fig. 8.1). All the vectors which are within the threshold (Delta) are assigned to this codeword: eg., in Fig. 8.1 vectors A, B are assigned to codeword X. If we repeat this procedure for all codewords, there are three possible scenarios as observed from the perspective of the input vectors.

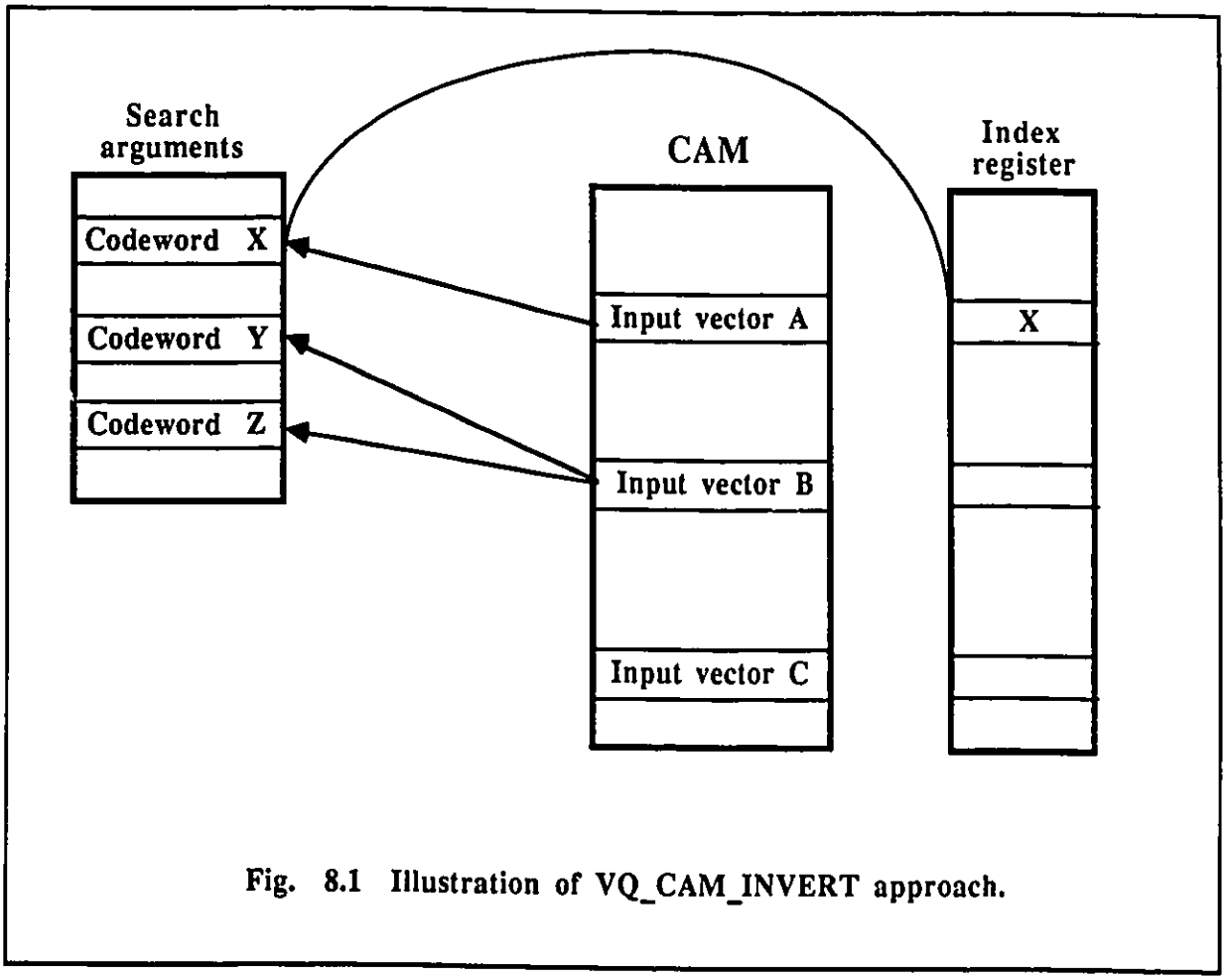

Fig. 8.1 Illustration of VQ_CAM_INVERT approach.

(i) The input vector (A) is assigned to a unique codeword (X). In other words the closest codeword to vector A is X. The index of the closest codeword is now stored in the index register, IR_i (in the result store module).

(ii) The input vector (B) is assigned to more than one codeword (Y, Z). In this case the threshold must be decreased and the Inexact_Match procedure repeated.

(iii) The input vector (C) is not assigned to any codeword. In this case the threshold must be increased and the Inexact_Match procedure repeated.

As in the case of the Gating procedure for VQ_CAM, we start off with a threshold value of 0 and monotonically increase the threshold until all input vectors are assigned to their closest codeword. The corresponding indices are then recorded in the index register. We call this procedure Gating_Mod and in procedural form it is expressed as follows.

PROCEDURE ***Gating_Mod*** (**In:** Set of Input_patterns,Templates; **Out:** Index_register)

Initialize Index_register, $IR_i = 0$ for i = 1, ..., K

Initialize Delta (threshold) to 0

BEGIN

WHILE (Some Template not assigned to the closest Input_pattern{IR_i=0, for any i })

DO (For each Input_pattern 1, ..., N)

BEGIN

Inexact_Match (**In:** Input_pattern, Templates, Delta ;

Out: Response_register)

Update contents of Index_register (corresponding to Templates which have been assigned the closest Input_pattern)

END

END DO

Increment Delta

END WHILE

END

END PROCEDURE

The index register records for each template, the index of the closest input pattern.

We now show how a CAM can be employed to execute the encoding algorithm using the VQ_CAM_INVERT approach. Once again, to illustrate the concept, we first present the implementation for 1-D vectors. In procedural form, the VQ_CAM_INVERT expressed in terms of the Gating_Mod procedure is as follows.

PROCEDURE VQ_CAM_INVERT (In: Input_vectors, Codewords; **Out:** Labels)

Initialize Templates := Input_vectors

Initialize Input_Patterns := Codewords

BEGIN

Gating_Mod (**In:** Input_patterns,Templates; **Out:** Index_register)

Label of the Input_pattern = Index of the closest template

END

END PROCEDURE

The index of the closest codeword (available in the index register) is then designated as the label. If we store all input vectors of dimension L = 1 in the CAM array, the speed-up is $S_p(K)$. The corresponding effective speed-up is $S_p(K/2^b b)$. The extension of this concept to an L-D vector is discussed in the following section.

8.2.4 MULTI-DIMENSIONAL VECTORS

In VQ, the closest codeword for a given input vector is usually determined by employing the MSE distortion measure (Eqn. 2.18). However, this measure cannot be readily implemented in a CAM based architecture and in its place is substituted the absolute difference measure. For the case of 1-D vectors, the two measures are equivalent and hence, the closest codeword is determined by minimizing the absolute difference between the input vector and the codewords: $\min_p (| V_i - C_p |)$, where the index i refers to the input vector and the index p refers to the codeword being compared.

For multi-dimensional vectors, we propose to use a metric based upon the maximum difference, namely

$$d^*(V_i, C_p) = \max_j (| V_{ij} - C_{pj} |) \qquad (8.1)$$

The optimal quantizer (equation 4) then becomes,

$$q(V_i) = C_p, \quad \text{iff } d^*(V_i, C_p) \leq d^*(V_i, C_n) \text{ for all } p \neq n \qquad (8.2)$$

The results reported by Murakami *et al* [78] and our simulations in section 8.3 show that this measure results in little degradation in performance compared to using the conventional MSE measure. In fact we show that this measure has the advantage of limiting large errors (Fig. 8.5).

Gating for Vectors :

A CAM-based procedure, Gating_Mod_Vector, which assigns a multi-element template to the closest multi-element input pattern, is now presented. A separate CAM array module, called an elemental CAM, is used for each vector dimension.

PROCEDURE ***Gating_Mod_Vector*** (**In:** Input_patterns,Templates; **Out:** Index_register)

Initialize Index_register, $IR_i = 0$ for i = 1, ..., K

Initialize Delta (threshold) to 0.

BEGIN

WHILE (Some Template not assigned to the closest Input_pattern{IR_i=0, for any i })

DO (For each Input_pattern 1, ..., N)

BEGIN

DO (For all elemental CAM's in parallel j = 1, ..., L)

Inexact_Match (**In:** Input_pattern (j), Templates (j), Delta;

Out: Response_register (j))

END DO

IF (Response_register (j) = 1, for j =1, ..., L) **THEN**

Input_pattern is closest for the template

END IF

Update contents of Index_register

END

Increment Delta

END WHILE

END

END PROCEDURE

The index register records for each template, the index of the closest input pattern. We note that the index register serves to mask out those templates which have been already assigned the closest input pattern ($IR_i \neq 0$).

Encoding algorithm :

The encoding algorithm can be implemented using the Gating_mod_vector procedure where the input vectors to be coded are set as the templates stored in the CAM array and the codewords are set as input patterns (search argument). The index of the closest codeword (available in the index register) is then designated as the label. The algorithm is expressed as follows.

PROCEDURE ***Encoding*** (**In:** Input_vectors, Codewords; **Out:** Labels)

Initialize Templates := Input_vectors

Initialize Input_patterns := Codewords

BEGIN

Gating_mod_vector (**In:** Input_pattern,Templates; **Out:** Index_register)

Label of the Input_pattern = Index of the closest template

END

END PROCEDURE

The labels are then transmitted.

8.2.5 CAM DESIGN

The block schematic of the general CAM design for VQ is shown in Fig. 8.2. For an L-dimensional vector, L modules of the basic CAM design (Fig. 3.3) are arranged in parallel, one per vector dimension. The design of the individual modules follows.

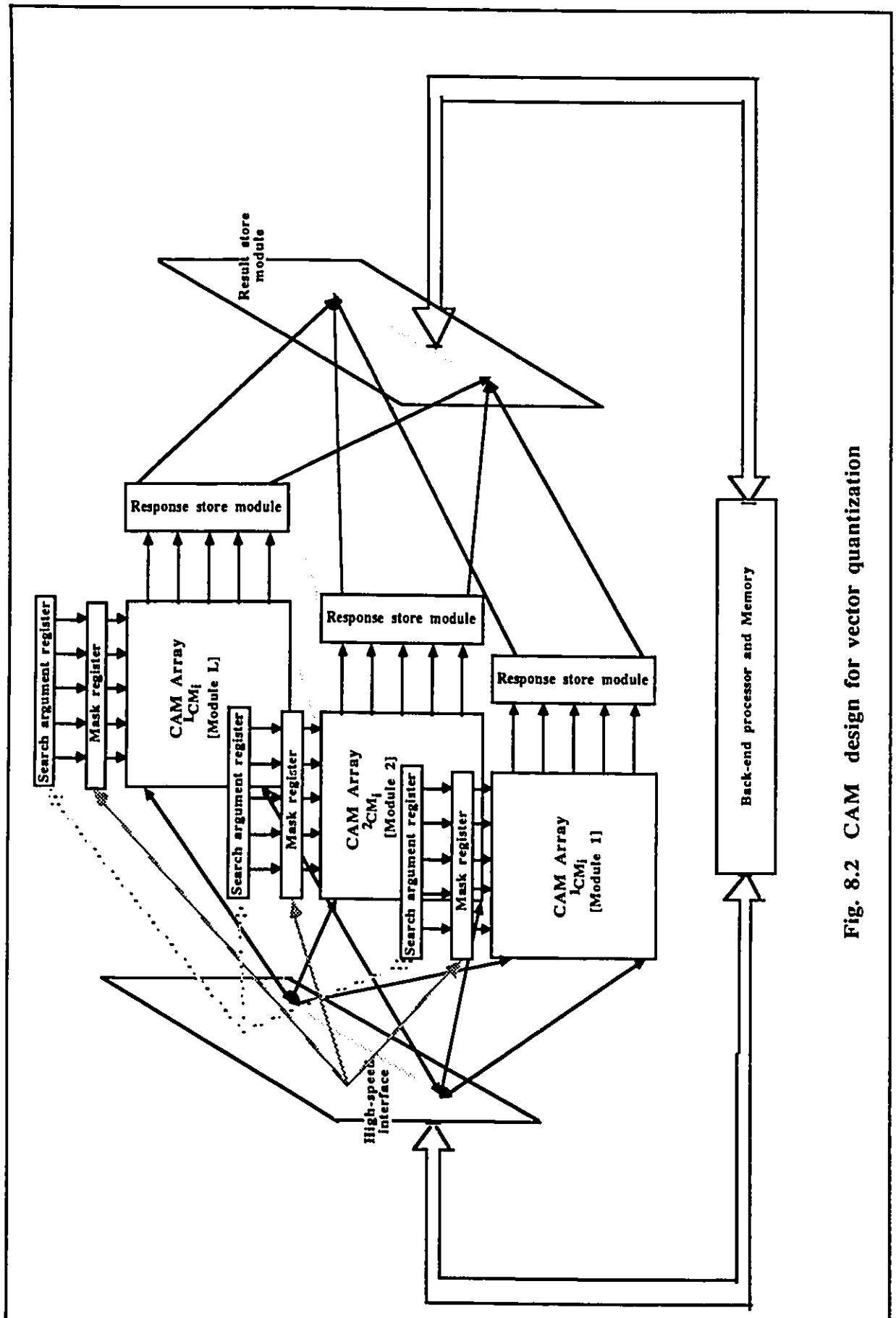

Fig. 8.2 CAM design for vector quantization

CAM array module :

Each CAM array module is organized into K x b cells, where K is the number of templates (input vectors in the image to be coded) and b is the number of bits per word (template). The input vectors are loaded in parallel into the cell array from an external data store through a high-speed interface.

Response store module :

Each response store module consists of an intermediate response register, I_i, and a response register, R_i. Two different response registers are required for computing the "within Δ" search (section 3.4.3).

Result store module :

The result store module consists of the intermediate result register, IS_j, which collects the results available from the individual final response registers. It also contains the index register, IR_j, to store the index of the closest input pattern (codewords) for the templates (input vectors).

8.2.6 IMPLEMENTATION OF VQIAC

We recall from section 2.3.2, that in VQIAC a new codebook is generated for each image, using the current input vectors as the training set. The algorithm is implemented using the approach adopted by VQ_CAM_INVERT; in other words, by storing the input vectors, V_{ij}, as templates in the elemental CAM array modules, jCM_i, where the index i refers to the individual vectors (i = 1, 2, ..., K) and j refers to the elements of the vector (j = 1, 2, ..., L). We note that each element of the vector is a word in the CAM array. The corresponding elements of the codewords, C_{pj}, where the index p refers to the codewords (p = 1, 2, ..., N) are set as the input patterns (search argument) in the L modules. The encoding procedure in section 8.2.4 is then applied and the closest codewords for each input vector is determined. We recall from section 2.3 that codebook generation involves a number of iterations of the encoding algorithm and hence, the encoding procedure is applied iteratively until the optimum codewords are obtained. However, at the end of

each iteration, the centroids of the cluster of vectors assigned to each codeword (search argument) must be calculated. This step is executed using the back-end processor (refer Fig. 8.2) or in the CAM array itself, if it has the computation capability. The new cluster centers are used as the codewords (search arguments) for the next iteration.

To facilitate fast computation of the cluster centers, we can balance the load and increase the utilization of the back-end processor by performing the sum of the vectors belonging to a cluster as and when the closest codeword is determined and storing the intermediate sum values in the Sum_register. In procedural form, the implementation of the VQIAC using the CAM is as follows.

PROCEDURE ***VQIAC-CAM*** (**In:** Input_vectors, Initial_codewords; **Out** : Labels, Final_codewords)

Initialize New_codewords := Initial_codewords

Initialize Sum_register contents to 0;

BEGIN

REPEAT

Encoding (**In:** Input_vectors, New_codewords; **Out:** Labels)

Update Codewords {New_Codewords := Centroids of the Sum_register contents}

UNTIL (Distortion using the current New_codebook is within acceptable limits)

Final_codewords := New_codewords

Encoding (**In:** Input_vectors,Final_codewords; **Out:** Labels)

END

END PROCEDURE

The labels and the final codewords are then transmitted.

8.3. PERFORMANCE

Computer simulations are carried out on two test images of size 256 x 256 pixels with 8 bits/pixel. The first image is a Face (Frank) image and the second image is a radiographic (Chest) image. The images are preprocessed to yield blocks of 4 x 4 pixels which are rearranged as 16-dimensional vectors as shown in Fig. 2.3. A new (adaptive) codebook is generated using the

input vectors of the image to be coded as the training set, and starting with a universal codebook pre-designed from a separate set of ten images, as the initial codebook (seeds). The image vectors are then quantized using the new (adaptive) codebook. The simulation results using the VQIAC-CAM technique for the Face and Chest images, over a range of bit rates, are tabulated in Tables 8.1a & 8.1b, respectively. The corresponding results for the two images using VQIAC (starting with the same initial codebook) are also tabulated in Tables 8.1a & 8.1b, respectively. These results are used as a baseline for comparison.

8.3.1 COMPUTATIONAL COMPLEXITY

We recall from section 8.2.2 and 8.2.3, that the speed-up using the VQ_CAM and VQ_CAM_INVERT implementations are $S_p(LN/b2^b)$ and $S_p(KL/b2^b)$, respectively. A CAM based implementation results in an additional speed-up, $S_p(P)$, from the faster execution of the basic operation. For the CAM implementation, the basic operation is a logic operation (at the bit level), whereas in the SISD implementation it requires an arithmetic operation. This additional speed-up is given by,

$$S_p(P) = S_p(T_e / T_{e(CAM)}) \qquad (8.3)$$

Hence, the total speed-up using the VQ_CAM and VQ_CAM_INVERT implementations are $S_p(NLP/b2^b)$, and $S_p(KLP/b2^b)$, respectively.

We now present in graphical form (Fig. 8.3a) the relative importance of the various speed-up factors, where a logarithmic scale has been used. The SISD implementation is used as a baseline for comparison with $\log(S_p(1)) = 0$. In the graph, the speed-up values for the two cases of VQ_CAM and VQ_CAM_INVERT are shown. The numbers presented in Fig. 8.3a correspond to a 512 x 512 image with 16-D vectors (L = 16, and K = 16384) and 8-bit words (b = 8) using a codebook of size, N = 256 (which is equivalent to a bit rate of 0.5 bits/pixel and 0.63 bits/pixel for the VQUC and VQIAC techniques, respectively). We note that there is a decrease in execution time due to a positive speed-up and an increase in execution time which we call the negative speed-up.

The positive speed-up is due to the following three factors.

Simulation results using VQIAC and VQIAC-CAM techniques

TABLE 8.1a (Face image)

NO. OF CODEWORDS	BIT RATE FOR LABEL	BIT RATE FOR CODE WORDS	TOTAL BIT RATE	% NMSE	% NMSE (CAM)	NUMBER OF ITERATIONS	EXECUTION TIME (in milliseconds)
8	0.18	0.01	0.19	0.80	0.95	9	1.80
16	0.22	0.02	0.24	0.57	0.68	7	2.62
31	0.26	0.05	0.31	0.41	0.46	6	5.71
59	0.31	0.11	0.42	0.30	0.35	8	8.46
95	0.34	0.17	0.51	0.24	0.26	7	15.57
164	0.40	0.33	0.73	0.17	0.20	10	33.59

TABLE 8.1b (Chest image)

NO. OF CODEWORDS	BIT RATE FOR LABEL	BIT RATE FOR CODE WORDS	TOTAL BIT RATE	% NMSE	% NMSE (CAM)	NUMBER OF ITERATIONS	EXECUTION TIME (in milliseconds)
9	0.17	0.01	0.18	0.58	0.69	11	2.21
16	0.20	0.02	0.22	0.37	0.43	12	4.26
29	0.24	0.04	0.28	0.23	0.28	7	4.75
40	0.29	0.06	0.35	0.16	0.19	8	7.37
59	0.32	0.10	0.42	0.13	0.15	7	9.67
84	0.36	0.13	0.49	0.11	0.12	9	17.20
127	0.39	0.21	0.60	0.09	0.10	7	20.81

Simulation results using different passes of the Gating operation

TABLE 8.2a (Face image)

NO. OF PASSES	%NMSE
128	0.259
64	0.261
20	0.268
10	0.278

TABLE 8.2b (Chest image)

NO. OF PASSES	%NMSE
128	0.119
64	0.119
20	0.123
10	0.136

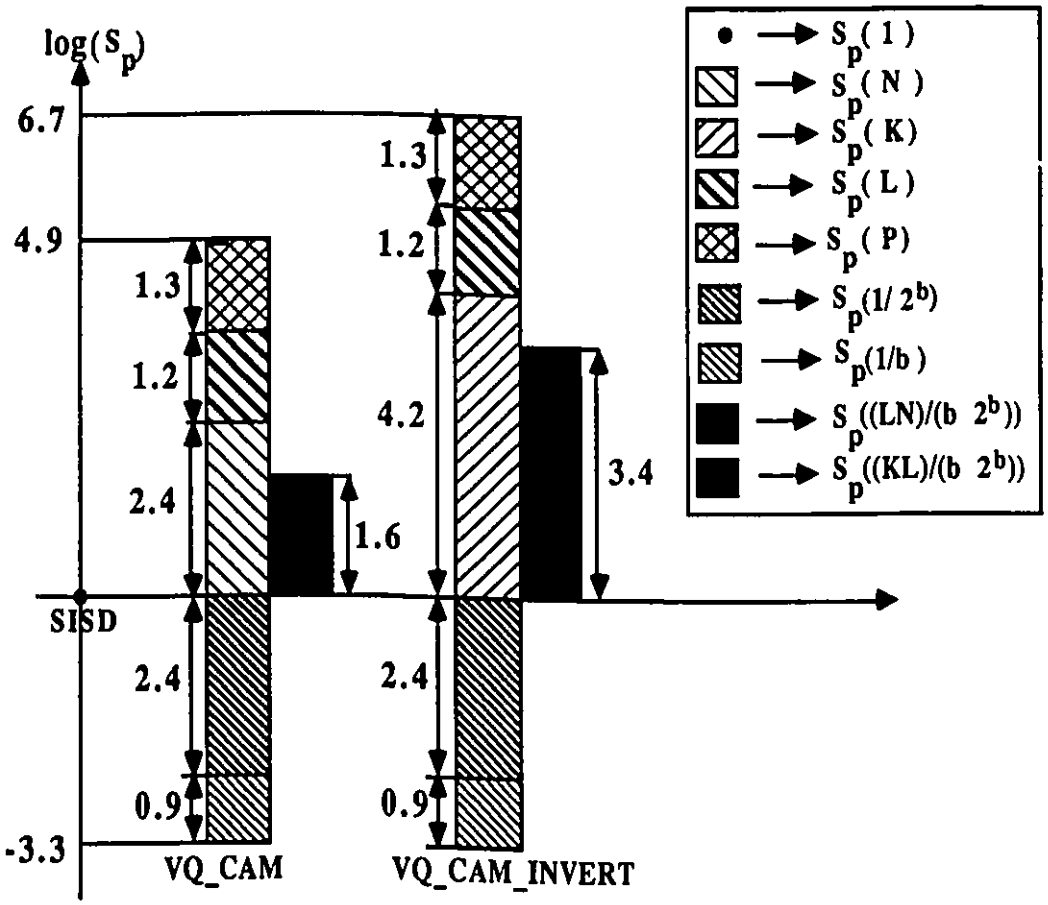

Fig. 8.3a Speed-up chart for a 512 x 512 image using a codebook of size 256

1. The first factor is the speed-up in the execution of the basic operation, $S_p(P)$. Typical values in the literature for, T_e and $T_e(CAM)$, are respectively, 100 ns - 200 ns [10-13] and 5 ns - 25 ns [79]. We have used for T_e and $T_e(CAM)$ the values of 200 ns and 10 ns, respectively. Hence, $S_p(P) = 20$. This speed-up is the same for both cases.
2. The next factor is the speed-up due to parallelism in the direction of vector dimension, $S_p(L) = 16$. This speed-up is also same for both cases.
3. The third factor for the case of VQ_CAM is the speed-up due to the parallelism in the direction of codewords, $S_p(N) = 256$; whereas for the case of VQ_CAM_INVERT the speed-up is due to the parallelism in the direction of input vectors, $S_p(K) = 16384$.

The negative speed-up (which slows down the system) is the same for both cases and is due to the following two factors.

1. The extra 2^b passes required for the *Gating* operation, $S_p(1/2^b) = 0.0039$.
2. The b iterations required for the "within Δ" search, $S_p(1/b) = 0.125$.

If we add up all the factors, this gives the effective speed-up which is shown by the filled bars in Fig. 8.3a. Hence, the effective speed-up using the VQ_CAM and VQ_CAM_INVERT implementations are $S_p(40)$ and $S_p(2560)$, respectively. Thus, VQ_CAM_INVERT implementation results in a speed-up which is two orders of magnitude greater than that of VQ_CAM implementation.

The speed-up chart for a 1024 x 1024 image with 16-D vectors (L = 16, and K = 16384) and 8-bit words using a codebook of size, N =256 (which is equivalent to a bit rate of 0.5 bits/pixel and 0.53 bits/pixel for the VQUC and VQIAC techniques, respectively) is shown in Fig. 8.3b. The net speed-up (log scale) using VQ_CAM and VQ_CAM_INVERT for 256 x 256, 512 x 512, and 1024 x 1024 images with a codebook of size, N = 16 is tabulated in Table 8.4a. The corresponding values with codebooks of size, N = 256 and N = 4096 are tabulated in Tables 8.4b & 8.4c, respectively. It can be observed from Figures 8.3a & 8.3b and Tables 8.4a - 8.4b, that for a fixed codebook size, the speed-up using VQ_CAM_INVERT increases with image size as the speed-up is proportional to the number of input vectors. On the other hand the speed-up using

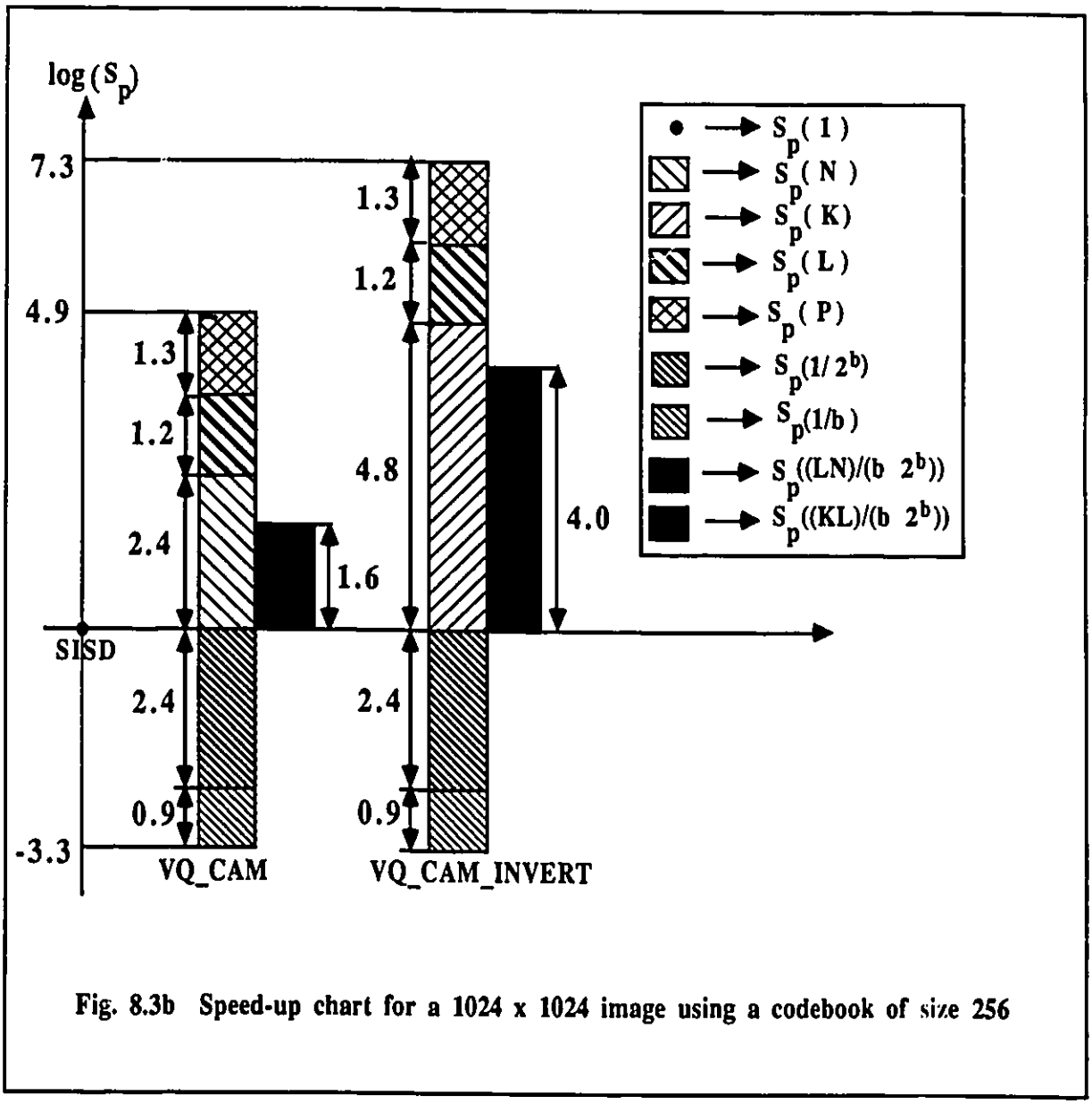

Fig. 8.3b Speed-up chart for a 1024 x 1024 image using a codebook of size 256

TABLE 8.3a

Speed-up factors (in log scale) using VQ_CAM and VQ_CAM_INVERT for different image sizes with a codebook of size 16

IMAGE SIZE	VQ_CAM	VQ_CAM_INVERT
256 x 256	3.7	6.1
512 x 512	3.7	6.7
1024 x 1024	3.7	7.3

TABLE 8.3b

Speed-up factors (in log scale) using VQ_CAM and VQ_CAM_INVERT for different image sizes with codebook of size 256

IMAGE SIZE	VQ_CAM	VQ_CAM_INVERT
256 x 256	4.9	6.1
512 x 512	4.9	6.7
1024 x 1024	4.9	7.3

TABLE 8.3c

Speed-up factors (in log scale) using VQ_CAM and VQ_CAM_INVERT for different image sizes with a codebook of size 4096

IMAGE SIZE	VQ_CAM	VQ_CAM_INVERT
256 x 256	6.1	6.1
512 x 512	6.1	6.7
1024 x 1024	6.1	7.3

VQ_CAM remains a constant over the range of image sizes. It can also be observed from Tables 8.4a - 8.4c that the speed-up using VQ_CAM_INVERT is a constant over the range of codebook size. On the other hand, the speed up using VQ_CAM increases with codebook size as in this case, the speed-up is proportional to the number of codewords.

The total speed-up, $S_p(KLP/b2^b)$, using the VQ_CAM_INVERT implementation results in an encoding complexity of order $O(NLK)/S_p(KLP/b2^b) = O(N2^b b/P)$. Since, the centroid computation is overlapped with the search operation, it follows that the computational complexity for codebook generation involving I iterations is of order $O(N2^b bI/P)$.

Execution time :

We now calculate the execution time, T_g, for codebook generation using an SISD implementation (ignoring the centroid computation complexity). For the numbers presented and assuming 10 iterations of the codebook generation algorithm, the execution time of the codebook generation is given by,

$T_{g(SISD)}$ = 256*16384*16*10*200 ns = 134.22 seconds = 2.24 minutes.

The corresponding execution time using the VQ_CAM_INVERT implementation is given by,

$T_g = T_{g(SISD)} / S_p(KLP/2^b b)$ = 52.43 milliseconds.

8.3.2 SIMULATION RESULTS

The coding performances are compared using the rate-distortion curves and are shown in Fig. 8.4a & 8.4b, for the Face (Frank) image and Chest image, respectively using the VQIAC-CAM technique. The bit rates for VQIAC-CAM and VQIAC, for the two images, are computed by using equation 2.25, where the codewords and the labels are both coded using a variable rate coder. For both techniques, the normalized mean square error, NMSE, (equation 2.14) is used as the measure of distortion. We note that VQIAC-CAM uses a mini-max criterion and hence, limits large errors. On the other hand with the VQIAC technique large errors may occur. The maximum error using the mini-max criterion for the chest image at a bit rate of 0.5 bits/pixel is 18 in the gray scale. However, for the MSE measure, there are errors as high as 37 (more than twice the maximum

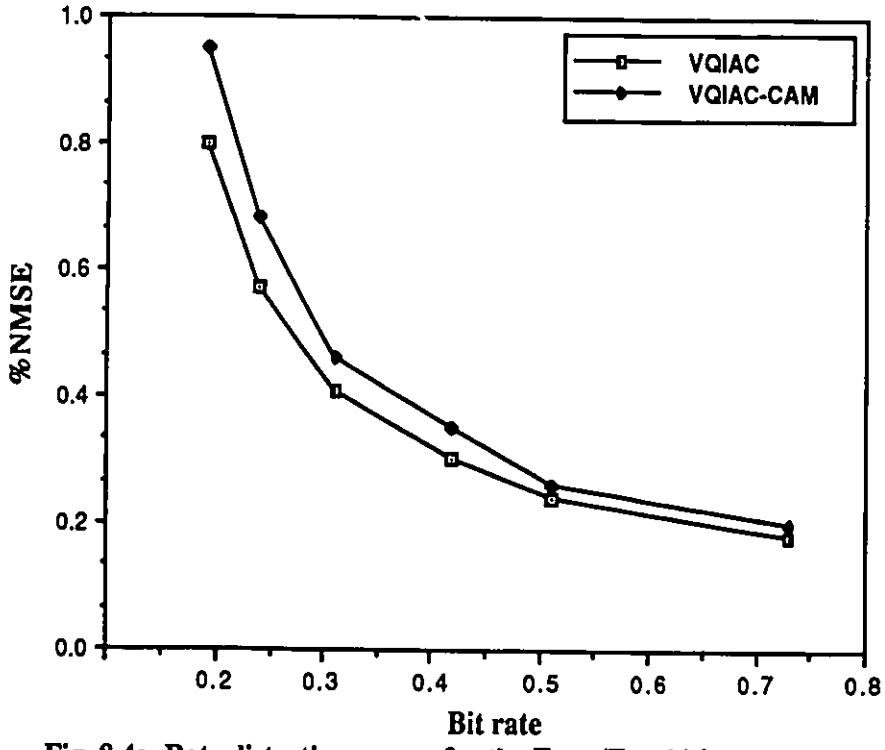

Fig. 8.4a Rate distortion curves for the Face (Frank) image

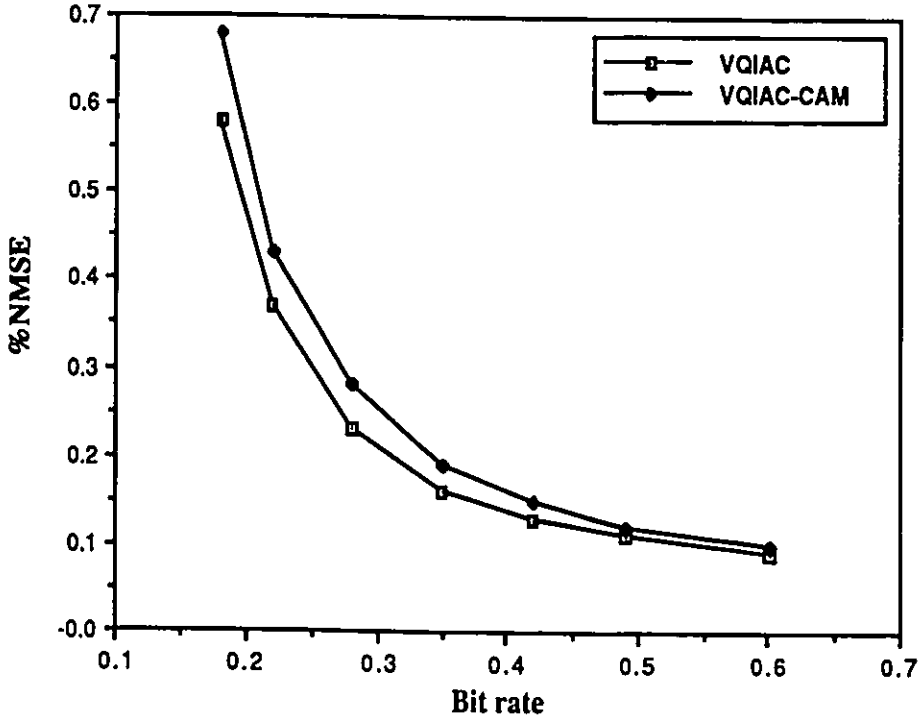

Fig. 8.4b Rate distortion curves for the Chest image

using the mini-max) in the gray scale. The distribution of errors for the Chest image for MSE measure above 18 at a bit rate of 0.5 bits/pixel (VQUC) is shown in Fig. 8.5. In other words, although the mini-max measure has a slightly higher NMSE, it has the advantage of limiting large errors.

The execution time of VQIAC-CAM for the Face and Chest images over a range of bit rates are tabulated in Tables 8.1a & 8.1b, respectively. From the results, we observe that real-time (30 frames per second) codebook generation and encoding is possible. We note that further gains in speed-up is possible by reducing the number of passes for the *Gating* operation. Our simulation results for different number of passes of the *Gating* operation (at a bit rate of 0.5 bits/pixel) tabulated in Tables 8.2a & 8.2b for the Face and Chest image, respectively, indicate that there is very little degradation (< 3.5%) in performance in using 20 iterations instead of 256 iterations (savings of 92%). This implies an additional speed-up $S_p(13)$ and a reduced execution time of 2.63 ms and 1.66 ms, for the Face (at a bit rate of 0.73 bits/pixel) and Chest (at a bit rate of 0.6 bits/pixel) images, respectively. This makes possible real-time codebook generation of optimum codebooks.

8.4. SUMMARY

We have presented a CAM based architecture for real-time image coding using adaptive vector quantization. VQ essentially involves a search operation on the codebook to obtain the best match. The search mechanism when implemented sequentially results in a complexity O(KLN) which is heavily compute intensive making real-time implementation difficult. We have presented a novel concept of inverting the order of comparison in VQ: the input vectors are stored as templates stored in the CAM array and the codewords are set as the search arguments, which exploits parallelism in the L and K directions resulting in a speed-up $S_p(KL)$ (typically of order 64000). In addition, there are gains in the execution time for the basic distortion operation (typically of order 5-40). This makes possible codebook generation and encoding in real-time (33 frames per second). We note that although the mini-max measure results in a higher NMSE, it has the advantage of limiting large errors. The regular and iterable architecture is suitable for VLSI implementation.

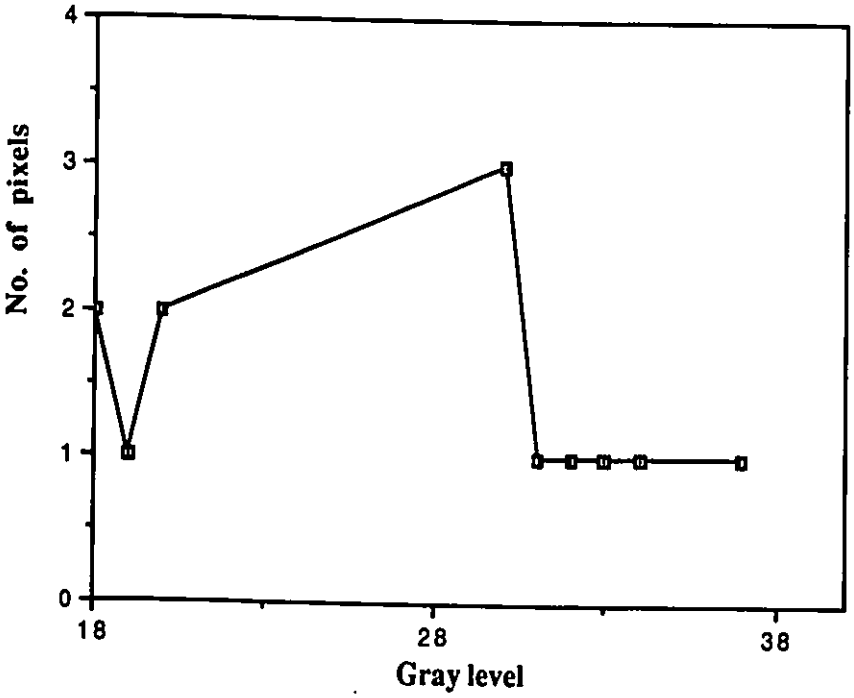

Fig.8.5 Errors above the threshold (18) using the MSE criterion at a bit rate of 0.5 bits/pixel

9. SUMMARY, CURRENT, AND FUTURE WORK

9.1 SUMMARY

In this thesis, two new adaptive algorithms for image vector quantization and two new architectures to implement vector quantization in real-time have been presented. The two algorithms provide a good compromise between coding performance and computational complexity thus, making possible real-time implementation. Simulation results demonstrate the improvement in coding performance and the gains in computational complexity. The two architectures map vector quantization algorithms efficiently, and provide substantial speed-up by exploiting parallelism in a number of directions. The architectures are regular and iterable in structure which makes possible VLSI implementation.

VQRUC and VQRUC-1

In VQRUC, a subset of codewords in the universal codebook which are used by the input vectors of a given image are identified which form the reduced codebook. This results in considerable savings in bit rate for label transmission. An extension of this algorithm is the VQRUC-1 technique, which involves one iteration on the reduced codebook to adapt the codewords to the image to be coded. In this case, there is an overhead involved to transmit the difference codebook. An entropy coder is used to code the labels and the error codebook. Simulation results demonstrate good coding performance at a substantially reduced complexity.

VQMMC

This is a super-adaptive algorithm based on mini-max error criterion. Here, the codebook is generated on the fly from the input vectors to be coded. A small primary codebook is used to store the frequently used codewords while, a larger secondary codebook is used to store the less frequently used codewords. Both the transmitter and receiver maintain identical codebooks and hence keep track of the changes without any overhead information. The labels and codewords are encoded using an entropy coder. Simulation results demonstrate the excellent rate distortion performance. The important point is the absence of large errors and the simplicity of the technique.

Systolic Array Architecture for VQ

In this architecture, the encoding and codebook generation operations are overlapped in the same structure. A basic systolic cell is designed with two modes of operation (forward and reverse). In the forward mode, the cell executes the distortion computation operation. In the reverse mode, the cell executes the new codeword computation. The two operations takes place simultaneously with a delay buffer to synchronize the operations. The speed-up obtained using this architecture, as the result of exploiting parallelism in the directions of vector dimension, L, and codeword dimension, N, and the gains in the computation and transfer times for the new codewords makes possible sub-optimal codebook generation and encoding in real-time.

CAM Architecture for VQ

In the content-addressable memory (CAM) architecture, the input vectors are stored in the CAM array and the codewords are used as search arguments and a match is sought by accessing all the input vectors in parallel. This implies that matching must be performed from the perspective of codewords; namely for a given codeword, all input vectors are evaluated in parallel. This architecture results in a substantial speed-up by exploiting parallelism in the directions of input vectors, K, and codewords, N. This speed-up coupled with the gains in execution time of the basic distortion operation makes possible even optimal codebook generation in real-time (< 33 ms).

9.2 CURRENT RESEARCH WORK

It can be observed that among the classes of architectures discussed, CAM holds the promise for real-time codebook generation. The design discussed in chapter 8 is a preliminary one and several extensions are possible to make it more efficient and cost-effective

Efficient implementation of the gating operation using CAM by optimizing threshold selection.

We recall from section 8.2.1 that the threshold values used in the *Gating* procedure are not optimum. The threshold values are dependent on the image vectors to be coded and the choice of the initial codebook and hence, some gains in the coding performance can be achieved if the appropriate thresholds are used. We are currently investigating self-optimizing strategies for

threshold selection. One example is a tree-based threshold incrementing policy.

Effective utilization of the CAM by modifying the design of the basic cell.

The CAM architecture proposed in chapter 8 can be modified so as to be utilized more effectively. For example, we recall from section 8.1 that the CAM array is essentially pixel centered, where each cell is used to store/search one pixel value. We are investigating a modified design in which the CAM array is vector centered, where each cell stores a vector. The individual pixels in the vector can then be circulated into the search portion of the cell. The response and result store modules have to be accordingly modified.

9.3 FUTURE RESEARCH WORK

The two algorithms and the two architectures presented in this thesis have some promise for future investigation, particularly, the VQMMC algorithm and CAM architecture. Some of possible directions for future work are discussed below.

1) Comparative evaluation of universal vs image adaptive techniques on larger size images.
2) Systolic array designs with communication efficient inter-connectivities, such as star network and hypercube inter-connectivities.
3) CAM architecture for transform domain coding.
4) VLSI implementation of the CAM architecture
5) Improvements in the VQMMC algorithm performance by employing,
   (i) modified scanning methods such as peano scanning to exploit adjacent vector correlation,
   (ii) adapting the primary and secondary codebook sizes at inter and intra image levels, and
   (iii) adapting the threshold values to the local activity of the image.
6) Hardware implementation strategies for the VQMMC algorithm using stack based and CAM based architectures.

10. REFERENCES

[1] R.M. Gray, "Vector Quantization", *IEEE ASSP Magazine*, pp. 4-29, April 1984.

[2] A. Gersho and V. Cuperman, "Vector Quantization: A Pattern Matching Technique for Speech Coding", *IEEE Communications Magazine*, pp. 15-21, December 1983.

[3] J. Makhoul, S. Roucos and H. Gish, "Vector Quantization in Speech Coding", *Proc. of the IEEE*, Vol. 73, No. 11, pp. 1551-1588, November 1985.

[4] Nasser M. Nasrabadi and Robert A. King, "Image Coding Using Vector Quantization: A Review", *IEEE Trans. on Communications.*, Vol. COM-36, No. 8, pp. 957-971, August 1988.

[5] Y. Linde, A. Buzo, and R.M. Gray, "An Algorithm for Vector Quantizer Design", *IEEE Trans. on Communications*, Vol. COM-28, pp. 84-95, January 1980.

[6] M. Goldberg, P.R. Boucher, and S. Shlien, "Image Compression Using Adaptive Vector Quantization", *IEEE Trans. on Communications*, Vol. COM-34, No. 2, pp. 180-187, February 1986.

[7] G.A. Davidson, P.R. Capello, and A. Gersho, "Systolic Architectures for Vector Quantization", *IEEE Trans. on ASSP* , Vol. 36, No. 10, pp. 1651-1664, October 1988.

[8] B.P.M. Tao, H. Abut, and R.M. Gray, "Hardware Realization of Waveform Vector Quantizers", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-2, pp. 343-352, March 1984.

[9] G. Davidson and A. Gersho, "Application of a VLSI Vector Quantization Processor to Real-Time Speech Coding", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 1, pp. 112-124, January 1986.

[10] P. Capello, G. Davidson, A. Gersho, C. Koc, and V. Somayazulu, "A Systolic Vector Quantization Processor for Real-Time Speech Coding", *ICASSP'86*, pp. 2143-2146.

[11] G. Davidson, T. Stanhope, R. Aravind, and A. Gersho, "Real-Time Speech Compression with a VLSI Vector Quantization Processor", *Proc. IEEE ICASSP'85*, Vol. 4, pp. 1437-1440, Tampa, Fl., March 1985.

[12] D. Chase and A. Gersho, "Real-Time VQ Codebook Generation Hardware for Speech

Processing", *Proc. IEEE ICASSP'88*, pp. 1730-1733, New York, April 1988.

[13] P.A. Ramamoorthy and B. Potu, "Bit-Serial Systolic Chip Set for Real-Time Image Coding", *Proc. IEEE ICASSP'87*, Vol. 2, pp. 756-759, Dallas, TX, April 1987.

[14] R. Dianysian and R.L. Baker, "A VLSI Chip Set for Real-Time Vector Quantization of Image Sequences", *Proc. Intl. Symposium on Circuits and Systems*, May 1987.

[15] H. Sun and H. Hsu, "A VLSI Wavefront Array for Image Vector Quantization", *Proc. 30th Midwest Symposium on Circuits and Systems*, pp. 1230-1233, August 1987.

[16] H. Abut, B.P.M. Tao, and J.L. Smith, "Vector Quantizer Architectures for Speech and Image Coding", *Proc. IEEE ICASSP'87*, Vol. 2, pp. 756-759, Dallas, TX, April 1987.

[17] T.A. Nodes, J.L. Smith, and R. Hecht-nielsen, "A Fuzzy Associative Memory Module and its Application to Signal Processing", *Proc. IEEE ICASSP'85*, Vol. 4, pp. 1511-1514, Tampa, Fl., March 1985.

[18] Nasser M. Nasrabadi and Y. Feng, "Vector Quantization of Images Based upon the Kohonen Self-Organizing Feature Maps", *Proc. International Conference on Neural Networks*, pp. I-101-108, 1988.

[19] N. Abramson, *Information Theory and Coding* , McGraw Hill, 1963.

[20] R.W. Hamming, *Coding and Information Theory* , Prentice-Hall Inc., 1980.

[21] R.C. Gonzalez and P. Wintz, *Digital Image Processing* , Addision-Wesley Publishing Company Inc., 1977.

[22] W. K. Pratt, *Digital Image Processing* , Wiley, New York, 1978.

[23] W.K. Pratt, *Image Transmission Techniques* , Academic Press, New York, 1979.

[24] E.L. Hall, *Computer Image Processing and Recognition* , Academic Press, New York, 1979.

[25] A. Rosenfeld and A.C. Kak, *Digital Picture Processing* , Academic Press, New York, 1982.

[26] N.S. Jayant and P.C. Noll, *Digital Coding of Waveform* , Prentice-Hall, 1984.

[27] A.N. Netravali and J.O. Limb, "Picture Coding: A Review", *Proc. IEEE* , Vol. 68, pp. 366-406, March 1980.

[28] A.K. Jain, "Image Data Compression: A Review", *Proc. IEEE* , Vol. 69, pp. 349-389, March 1981.

[29] A. Gersho, "On the Structure of Vector Quantizers", *IEEE Trans. Inform. Theory*, Vol. IT-28, pp. 157-166, March 1982.

[30] J.P. Adoul, C. Collin, and D. Dalle, "Block Encoding and its Application to Data Compression of PCM Speech", *Proc. Canadian Communications. EHV Conf.*, Montreal, P.Q., Canada, pp. 145-148, 1978.

[31] D.T.S. Chen, "On Two-or More Dimensional Optimum Quantizers", *Proc. IEEE Intl. Conf. Acoustics., Speech, and Signal Processing*, pp. 640-643, 1977.

[32] A. Gersho, "Asymptotically Optimum Block Quantization", *IEEE Trans. Inform. Theory*, Vol. IT-25, pp. 373-380, July 1979.

[33] S.P. Lloyd, "Least Squares Optimization in PCM", Bell Lab. Tech. Note, 1957; also in *IEEE Trans. Inform. Theory*, Vol. IT-28, pp. 129-137, March 1982.

[34] L.D. Davisson, "Rate Distortion Theory and Application", *Proc. IEEE* , Vol. 60, No. 7, pp. 800-808, July 1972.

[35] T. Berger, *Rate Distortion Theory* , Prentice-Hall Inc., 1971.

[36] J.L. Mannos and D.J. Sakrison, "The Effects of Visual Fidelity Criterion on the Encoding of Images", *IEEE Trans. on Inform. Theory* , Vol. IT-20, pp. 525-536, July 1974.

[37] A. Gersho and B. Ramamurthi, "Image Coding using Vector Quantization", *Proc. IEEE ICASSP'82*, pp. 428-431, May 1982.

[38] R.L. Baker, and R.M. Gray, "Image Compression using Non-adaptive Spatial Vector Quantization", *IEEE Conference on Circuits and System-Computer*, pp. 55-61, 1983.

[39] T. Murakami, K. Aasi, E. Yamasaki, "Vector Quantizer of Video Signals", *Electronic Letters* 7, pp. 1005-1006, Nov. 1982.

[40] E.E. Hilbert, "Joint Pattern Recognition/Data Compression Concept for ERTS Multispectral Data", *SPIE*, Vol. 66, "Efficient Transmission of Pictorial Information," August 1975.

[41] R.A. King and N.M. Nasrabadi, "Image Coding using Vector Quantization in the Transform Domain", *Pattern Recognition Lett.*, Vol. 1, pp. 323-329, 1983.

[42] M. Goldberg, and H.F. Sun, "Image Sequence Coding using Vector Quantization", *IEEE Trans. on Communications.*, Vol. COM-34, pp. 703-710, July 1986.

[43] H.F. Sun, and M. Goldberg, "Image Coding using LPC with Vector Quantization", *Proc.*

IEEE International Conf. on Digital Signal Processing, pp. 508-512, Florence, Italy, Sept. 1984.

[44] Y.Yamada, K. Fujita, and S. Tazaki, "Vector Quantization of Video Signals", *Proc. Annu. Conf. IECE*, pp. 1031, 1980.

[45] B. Ramamurthi and A. Gersho, "Classified Vector Quantization of Images",IEEE Trans. on Communications., Vol. COM-34, pp. 1105-1116, November 1986.

[46] R. Aravind and A.Gersho, "Image Compression Based on Vector Quantization with Finite Memory", *Optical Engineering Journal*, July 1987.

[47] V. Cuperman and A. Gersho, "Adaptive Differential Vector Coding of Speech", *Proc. Conf. Rec., GLOBECOM'82*, pp. 1092-1096, December 1982.

[48] A. Gersho and M. Yano, "Adaptive Vector Quantization by Progressive Codevector Replacement", *Proc. IEEE ICASSP'85*, Vol. 1, pp. 133-136, Tampa, Fl., March 1985.

[49] C.L. Yeh, "Color-image Compression using Adaptive Binary-tree Vector Quantization with Codebook Replenishment", *Proc. IEEE ICASSP'87*, Dallas, Texas, April 1987.

[50] C.L. Yeh, "Image Compression Using Nonadaptive Vector Quantization with Adaptive Overhead Transmission for Codevector Replacement", *Proc. GLOBECOM 87.*

[51] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, "Equation of state calculations by fast computing machines", *J. Chem. Phys.*, Vol. 21, No. 6, pp. 1087-1092, June 1953.

[52] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, pp. 671-680, May 1983.

[53] Jacques Vaisey and Allen Gersho, "Simulated Annealing and Codebook Design", *Proc. IEEE ICASSP'88*, Vol. 2, pp. 1176-1179, New York, April 1988.

[54] A. Buzo, A.H. Gray Jr., R.M. Gray and J.D. Markel,"Speech Coding Based upon Vector Quantization", *IEEE Trans. Acoustics, Speech and Signal Processing*, Vol. ASSP-28, pp. 562-574, October 1980.

[55] B.H. Juang and A.H. Gray, "Multiple Stage Vector Quantization for Speech Coding", *Proc. IEEE Intl. Conf. on ASSP*, pp. 597-600, April 1982.

[56] M.J. Sabin and R.M. Gray, "Product Code Quantizers for Waveform and Voice Coding",

IEEE Trans. Acoustics, Speech and Signal Processing, Vol. ASSP-32, pp. 474-488, June 1984.

[57] R.L. Baker and R.M. Gray, "Differential Vector Quantization of Achromatic Imagery", *Proc. Intl. Picture Coding Symp.* March 1983.

[58] Josef Kittler and M. J. B. Duff, *Image Processing System Architectures* , Research Studies Press Ltd., England, 1985.

[59] W.K. Pratt and P.F. Leonard, "Review of Machine Vision Architectures", *Proc. SPIE Vol. 755, Image Pattern Recognition : Algorithm Implementations,Techniques and Technology* . pp 2-12, 1987.

[60] Duff, M.J.B, "*Computing Structures For Image Processing* ", Academic Press, New York, 1983.

[61] H.T. Kung, "Why Systolic Architectures ?", *IEEE Comput.*, pp. 37-46, January 1984.

[62] K. S. Fu, *VLSI for Pattern Recognition and Image Processing* , Springer-Verlag, New York, 1984.

[63] A.G. Hanlon, "Content-Addressable and Associative Memory Systems - A SURVEY", IEEE Trans. on Electronic Computers, Vol. 15, pp. 509-521, 1966.

[64] T. Kohonen, "Content-addressable Memories", Series in Information Sciences, Vol. 1, Springer-Verlag, Brelin-Heidelberg-New York-Tokyo, 1980.

[65] C.C. Foster, "Content Addressable Parallel Processors", Computer Science Series, Van Nostrand Reinhold Company, New York, 1976.

[66] J.E. McAteer, J.A. Capobianco, and R.L. Koppel, "Associative Memory System Implementation and Characteristics", *Proc. 1964 E.J.C.C.*, pp. 81-92.

[67] G.J. Simmons, "Application of an associatively addressed, distributed memory", *Proc. 1964 W.J.C.C.*, pp. 493-513.

[68] S.S. Yau and C.C. Yang "Pattern Recognition by using an Associative Memory", *IEEE Trans. on Electronic Computers*, Vol. 15, pp. 944-947, December 1966.

[69] Y. Chu, " A Destructive Read-out Associative Memory ", *IEEE Trans. on Electronic Computers*, Vol. 14, pp. 600-, 1965.

[70] L.O. Chua and L. Yang, "Cellular Neural Networks : Theory", *IEEE Trans. on Circuits and*

Systems , Vol. 35, No. 10, pp. 1257-1272, October 1988.

[71] L.O. Chua and L. Yang, "Cellular Neural Networks : Applications", *IEEE Trans. on Circuits and Systems* , Vol. 35, No. 10, pp. 1273-1290, October 1988.

[72] J. Hopfield, "Neurons with Graded Response have Collective Computational Properties like those of Two-state Neurons",*Proc. Nat. Acad. Sci. U.S.A.* ,Vol. 81, pp. 3088-3092, 1984.

[73] L.O. Chua and T. Lin, "A Neural Network Approach to Transform Image Coding", *International Journal of Circuit Theory and Applications* , Vol. 16, pp. 317-324, 1988.

[74] R.P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine* , Vol. 4, pp. 3-22, 1987.

[75] T. Kohonen, "Self-Organization and Associative Memory", Series in Information Sciences, Vol. 8, Springer-Verlag, Brelin-Heidelberg-New York-Tokyo, !984; 2nd ed. 1988.

[76] T. Kohonen, "The Neural Phonetic Typewriter", *IEEE Comput.*, Vol. 21, pp. 11-21, March 1988.

[77] H.F. Li, D.K. Probst, and R.N. Prasad, "Traversing the VLSI design hierarchy for a new, fast systolic stack", *IEE Proceedings* , Vol. 135, Pt. E, No. 1, January 1988.

[78] T. Murakami, K. Asai, and E. Yamazaki, "Vector Quantiser of Video Signals", *Electronics letters* , 1982.

[79] S. Jones, "Design, selection and implementation of a content-addressable memory for a VLSI CMOS chip architecture", *IEE Proceedings* , Vol.135, pp. 165-172, May 1988.

[80] S. Panchanathan and M. Goldberg, " Algorithms and Architecture for Image Adaptive Vector Quantization", *Proc. SPIE Visual Communications and Image Processing III*, Cambridge, Massachusetts, Vol. 1001, pp. 336-344, November 1988.

[81] S. Panchanathan and M. Goldberg, "A Mini-Max Error Criterion based Algorithm for Image Adaptive Vector Quantization ", *Proc. SPIE Medical Imaging '89*, New Port Beach, California, February 1989.

[82] S. Panchanathan and M. Goldberg, " A Systolic Array Architecture for Image Coding using Vector Quantization ", *Proc. 1989 International Symposium on VLSI Systems, Technology and Applications*, Taipei, Taiwan, May 1989.

[83] S. Panchanathan and M. Goldberg, "A Systolic Array Architecture for Image Coding using

Vector Quantization ", submitted to *IEE Proceedings - Computers and Digital Techniques*. April 1989.

[84] S. Panchanathan and M. Goldberg, "A Content-addressable Memory Architecture for Image Coding using Adaptive Vector Quantization ", *Proc. SPIE 1989 Technical Symposia on Aerospace Sensing - Advances in Image Compression and Automatic Target Recognition*, Orlando, Florida, March 1989.

[85] S. Panchanathan and M. Goldberg, "A Content-addressable Memory Architecture for Image Coding using Vector Quantization ", submitted to *IEEE Trans. on ASSP*, April 1989.