An Architecture for Tree Search Based Vector Quantization for Single Chip Implementation

Heonchul Park, Viktor K. Prasanna and Cho-Li Wang
Department of Electrical Engineering-Systems, EEB-244
University of Southern California
Los Angeles, CA 90089-2562
(213) 740 - 4483
prasanna@halcyon.usc.edu

Abstract

Vector Quantization (VQ) has become feasible to be used in real-time applications by employing VLSI technology. In this paper, we propose a new search algorithm and an architecture for implementing it, which can be used in real-time image processing. This search algorithm takes $O(k)$ time units on a sequential machine, where $k$ is the dimension of the codewords, assuming unit time corresponds to one comparison operation. The proposed architecture employs a single Processing Element (PE) and $O(N)$ external memory for storing $N$ hyperplanes used in the search, where $N$ is the number of codewords.

Compared with known architectures for VQ in the literature, the proposed design does not perform any multiplication operation, since the search method is independent of any $l_q$ metric, $1 \leq q \leq \infty$. It leads to an area efficient design with the PE consisting of a comparator and $O(k)$ registers. Also, the memory used by the design is significantly less than those employed in the known architectures.

1 Introduction

Low bit rate coding is essential for image applications such as TV transmission, video conferencing, remote sensing via satellite, computer communication, facsimile transmission, etc. [19]. Data compression techniques reduce the storage and communication channel bandwidth needed to process such signals while maintaining acceptable fidelity. One deficiency of conventional compression techniques is that scalar quantization on individual pixels of images is required [15]. According to Shannon's work [21], Vector Quantization (VQ) provides better performance by coding vectors instead of scalars. In addition, VQ technique can be used to solve pattern recognition and data classification problems [13] [23].

*This research was supported in part by NSF under grant IRI-9115810.
In VQ, a set of input pixels is compared with codevectors or hyperplanes associated with the codevectors, and the index of the nearest codevector is transmitted. This index is used to recover the input pixels at the decoder. Two main search techniques have been proposed to map an input vector to the index of a codevector in the codebook; full search and tree search. Full search compares the input vector with each codevector in a given set of codevectors and outputs the index of the codevector for which the distance is minimum [16]. Tree search performs limited amount of search by employing a (binary) tree data structure. On a sequential machine, the time complexity of the full search is $O(kN)$ and that of the tree search is $O(k \log N)$, where $N$ is the number of codevectors and $k$ is the number of dimensions in the codevectors. This assumes unit time corresponds to a multiplication. The recovered image encoded by the tree search have been shown to be competitive with the ones encoded by the full search [11]. The codebook employed in VQ has approximate codevectors, since generating an optimal set of codevectors is NP-Complete [10].

The complexity of the encoding system becomes a major factor in coding data at a low bit rate with an acceptable level of distortion. With advances in VLSI, VQ has become a feasible approach for coding speech and image data during the recent years [11] [15] [19]. However, most implementations of VQ have been limited to speech coding [6], since image coding requires much higher throughput rate. Several architectures have been reported for image coding using 1-D and 2-D systolic arrays [5] [9] [12] [19]. These solutions result in multiple chips, since these implementations require a large number of PEs. In addition, they also need large I/O bandwidth with the host.

Recently, some tree search based architectures have been proposed [9] [12]. These architectures employ $O(\log N)$ PEs and $O(kN)$ memory. Each PE has a pipelined multiplier to compute the $L_2$ metric (Euclidean distance). The design in [12] has external memory to allow the PEs to be modular, while the design in [9] has local memory within each PE to support fast access to data and requires fixed I/O bandwidth with the host. One deficiency of these designs is that they cannot handle large codebooks efficiently; the design in [12] requires $O(\log N)$ I/O bandwidth, the design in [9] requires large on-chip memory (local memory) in the PEs. Each PE requires different amount of memory which increases exponentially, since each level of the tree is mapped onto a PE. Thus, these designs result in multiple chips for large $N$.

There is no known single PE implementation in the literature which can operate at video rate. To achieve a single PE implementation at video rate, it is necessary to design a fast tree search based VQ algorithm. Also, intensive multiplication operations needed to compute the Euclidean distance should be eliminated in the search, since multipliers

---

1 An affine subspace of $\mathbb{R}^d$ of dimension $k-1$ is called a hyperplane. Alternately, a hyperplane is a set of points, $z$ satisfying $a_1z_1 + a_2z_2 + \cdots + a_dz_d = b$, with not all $a$'s equal to zero.

2 All logarithms in this paper are to base 2.
result in high PE area complexity.

In this paper, we propose a new tree search algorithm for VQ using the clustering technique in [10]. This search algorithm takes $O(\log N)$ time on a sequential machine to map an input vector to an index, assuming unit time corresponds to one addition operation and the input vector is available in the memory. Note that other search algorithms employed in [4] [5] [7] [9] [12] assume unit time to correspond to a multiplication operation. The proposed search does not require multiplication operation. Also, it requires significantly less memory to store the search information (by a factor of $O(k)$), compared with conventional search algorithms for VQ. An architecture suitable for VLSI implementation is shown based on the proposed tree search. The proposed architecture employs a single PE and $O(N)$ external memory. Compared with known architectures supporting tree search [9] [12], it has a simple PE structure.

This paper is organized as follows. In Section 2, we briefly introduce VQ and its computational requirements. A new search algorithm for VQ based on clustering is presented in Section 3. Details of the proposed architecture are presented in Section 4. In Section 5, concluding remarks and comparisons with known designs are made.

2 Vector Quantization

In this section, we briefly describe vector quantization. Additional details can be found in [1] [11] [14] [15]. We also describe applications of VQ in image compression to identify the computational requirements which should be handled in real-time applications.

2.1 Vector Quantization

In VQ, the input is decomposed into vectors. The index of the nearest codevector in the codebook based on a given distortion measure is transmitted. This index is used to identify the input vector at the decoder. The procedure is repeated for successive input vectors. Fig. 1 shows a block diagram of a vector quantizer. The encoder sends the index of the nearest neighbor for each input vector. At the receiver, table look-up (which takes $O(1)$ time) is used to identify the codevector. The size of the codebook $N$ is determined by the bit-rate $r$ and the dimension of the vectors $k$, where $r = \lceil \log_2 N \rceil / k$ bits/sample, $0 < r < 1$. In practice, $k$ and $N$ up to 64 and 4096, respectively, have been employed [5].

The codebooks are designed using a clustering technique. Clustering is to partition a given set of training vectors in $k$-dimensional space into $N$ groups, called clusters, so that vectors within each cluster are near to each other. Each cluster is represented by a vector which is the center of the cluster. One objective function for clustering is to minimize the maximum distance between the vectors in each cluster and the center of the cluster for a given $L_q$ metric, $1 \leq q \leq \infty$. Computing an optimal clustering (which can be used to generate the codevectors using training data) is NP-Complete for $k \geq 2$ [10].
The sequential worst case time complexity of the nearest neighbor search is $O(kN)$ in full search, assuming unit time corresponds to a multiplication. The full search provides locally optimal output for a given codebook. The tree search in [3] can map an input vector to a codevector in $O(k \log N)$ time without backtracking such that each codevector occupies a Voronoi cell. At each node, the input vector is compared with two codevectors. Tree search employing hyperplane tests [4] can be used for performing the search. This can also reduce the memory requirements. Each node of the tree has a hyperplane information which is represented as $\sum_{i=1}^{k} a_i z_i = c$, where $a_i$ is a coefficient of the hyperplane and $z_i$ is an element of a vector in the hyperplane. The hyperplane is a bisector between two codevectors $c_i$ and $c_j$ which correspond to the codevectors in a node of the tree in [3] (see [4] for more details). Each node makes a comparison to check if the input vector is inside a predetermined hyperplane or not. For a given input vector, the search is performed at $O(\log N)$ nodes and $O(k)$ multiplications are required for a comparison operation at each node of the tree. It leads to $O(k \log N)$ operations for each input vector. The tree search employing hyperplane tests [4] requires less memory (by a factor of 2), compared with the usual tree search. This encoding has been shown to be competitive with the ones using the full search [11]. This approach has two deficiencies. Full search over $O(2^k)$ ($= N$) codevectors is recommended at each external node of the tree [4]. Also, generating the hyperplanes, such that each hyperplane has equal number of Voronoi cells inside and outside of it and maximizing the number of Voronoi cells where internal space is not cut through by the hyperplanes, is computationally expensive [24].

Figure 1: Block Diagram of a Vector Quantizer
2.2 Requirements for Real-Time Image Processing

Digital image data transmission is an application area for VQ. Transmission of 512 x 512 images with 8 bits/pixel at 30 frames/s requires 63 Mbits/s bandwidth without data compression. If VQ at 0.5 bit/pixel bit rate is used, a communication channel bandwidth of 3.9 Mbits/s would suffice. For spatial domain picture coding, input vectors of size ranging from 16 to 36 have been employed [5]. Each word has up to 12 bits [12][20]. For coding TV signals using 512 x 480 images, the available time for processing the input vectors with \( k = 16 \) is 1.184 \( \mu s \) and it increases to 2.368 \( \mu s \) with \( k = 32 \) [19].

Assuming a rate of \( r = 0.5 \) bit/pixel, \( k = 32 \), and 512 x 512 images with 8 bits/pixel at 30 frames/s are employed, it requires \( N = 2^k = 2^{16} \) codevectors. Assuming full search is employed, the number of PEs needed for real-time operations in [19] is \( 2^{24} \). The number of multiplication operations needed for an input vector is \( 2^{24} \). The resulting architecture has to handle \( 2^{24} \) multiplication operations per frame. Thus, any single PE implementation cannot perform full search in real-time for the above image data.

For the same image data, the number of PEs used in tree search is 16, and the number of multiplications needed for an input vector is 512. The total number of multiplications per frame is \( 2^{24} \), which results in over 240 MOPS (Million Operations Per Second). The \( i \)th PE has memory size of \( 32 \times 2^i \) words, \( 0 \leq i \leq 15 \), in the known tree search architectures [9][12]. If a single chip implementation is attempted, the total size of the on-chip memory will be \( k \times N = 2^{24} \) words which is currently infeasible to implement. Also, if off-chip memory is employed, it would require more than 136 I/O pins for data communications alone, assuming that each element of the vector is represented by 8 bits. Thus, known tree search algorithms, which result in multiple chips, are not suitable for a single PE implementation.

In image processing applications, higher computational requirements arise in order to achieve desired performance with VQ. The available time for encoding an input vector increases as the dimension of the codevectors increases, assuming a source with fixed scalar throughput rate. It is necessary to design a new tree search algorithm having lower computational complexity for a single PE implementation such that the PE can operate at input data rate.

3 Tree Search Using Clustering

In this section, we introduce a new tree search algorithm based on the clustering algorithm in [10]. The performance of this algorithm on image data is illustrated using simulation results.
3.1 Algorithms for Clustering

Some clustering algorithms have been proposed to generate an approximate codebook for VQ [10] [14]. The algorithm proposed in [14] (known as the LBG algorithm) is the most popular method for generating codebooks. The LBG algorithm for deriving a codebook using a set of training vectors is iterative. After choosing an initial codebook (which can be arbitrary or can be a previously used one), iteration begins assigning each training vector to its codevector which satisfies a given distortion measure using full search. Next, the codevectors are modified to minimize the error between the codevectors and the training vectors. Its time complexity is \(O(MN)\) on a sequential machine where \(M\) is the number of training vectors and \(N\) is the number of codevectors.

The clustering algorithm in [8] employs multidimensional tree search method proposed in [2]. In each iteration, it combines nearer two codevectors as a new codevector. Its time complexity is \(O(M \log M)\) which is independent of the desired number of codevectors.

An efficient approximation algorithm for clustering is shown in [10]. Based on the algorithms for the farthest pair problem and box decomposition [10], clustering can be done in \(O(M \log N)\) time on a sequential machine, where \(M\) is the number of training vectors in \(R^k\) and \(N\) is the number of codevectors.

Algorithm 1 shows the basic idea of clustering in [10]. Let \(b = J_1 \times J_2 \times \ldots \times J_k\), where each \(J_i\) is an interval, denote a box in \(k\) dimensional space. Let box(S) be the box containing all the data in the training set \(S\). shrunk(b) denotes the smallest box containing the training vectors in \(b \cap S\). In this algorithm, a chosen box is split into two boxes using a hyperplane which is perpendicular to the longest interval of \(b\). If this hyperplane splits a box \(b\) into two boxes \(b_1\) and \(b_2\), then these two boxes are successors of \(b\). For example, if the first dimension of a box \([h_{1L}, h_{1H}] \times [h_{2L}, h_{2H}] \times \ldots \times [h_{kL}, h_{kH}]\) has the longest interval, then the bisecting simple hyperplane for the box is \(z_1 = (h_{1L} + h_{1H}) / 2\). Successors of this box are \([h_{1L}, z_1] \times [h_{2L}, h_{2H}] \times \ldots \times [h_{kL}, h_{kH}]\) and \([z_1, h_{1H}] \times [h_{2L}, h_{2H}] \times \ldots \times [h_{kL}, h_{kH}]\). The box to be split is chosen such that it is one of the two boxes among the boxes in \(B\) whose mutual distance is the largest and has larger volume. This is computed using the farthest pair algorithm. Finally, we can obtain \(N\) shrunken boxes whose union is equal to \(S\), where \(N\) is the desired number of codevectors. The center of each box becomes a codevector for VQ. Additional details including an analysis of the time complexity of this method (which is \(O(M \log N)\)) can be found in [10].

3.2 A New Tree Search Algorithm for VQ

By a simple modification of Algorithm 1, \(N\) \(k\)-dimensional unshrunken boxes can be obtained using the hyperplanes used to split the boxes. At the end of algorithm, \(N\) unshrunken boxes...
B ← \{box(S)\};
repeat
  pick a box b from B;
  choose disjoint boxes \(b_1\) and \(b_2\) in b satisfying certain constraints;
  \[ B ← (B - \{b\}) ∪ \{\text{shrunk}(b_1), \text{shrunk}(b_2)\}; \]
until \(|B| = N\).

Algorithm 1: Basic idea of clustering algorithm

Boxes are obtained such that each unshrunken box contains the shrunken box and there is no training vector in the intersection of the unshrunken box and the shrunken box. The total volume of the \(N\) unshrunken boxes is equal to that of the search space in \(R^k\). A codevector is chosen from each box. Due to the box decomposition scheme employed, each hyperplane employed in the proposed search can be represented as \(x_i = c\), for some \(i\). The total number of hyperplanes employed to generate the boxes is at most \(N\). Thus, each hyperplane can be represented using two values \((i, c)\): \(i\) is an index and \(c\) is a constant. The proposed search does not compute any \(L_p\) metric, since given a hyperplane \((i, c)\) it only checks if the \(i\)th element of the input vector is greater than or equal to the given constant \(c\).

The clustering algorithm in [10] results in a balanced tree. Thus, a binary tree of height \(O(\log N)\) to search for a codevector can be easily constructed during the clustering procedure in Section 3.1. Initially, we have an unshrunken box containing all the training vectors which covers the complete search space. By splitting the training vectors using a simple hyperplane, the root of the tree has two children. The simple hyperplane information used to split the training vectors is stored at the root. The left (right) child has all the vectors such that the value in the dimension corresponding to the index of the simple hyperplane has less than (greater than or equal to) the constant associated with the hyperplane. This idea is recursively applied until the tree has the desired number of leaves.

When an input vector arrives at a node of the tree, the input element corresponding to the index of the hyperplane stored at the node is compared with the constant corresponding to the hyperplane. The node outputs a "0" or "1" based on the result of the comparison. Selection of an element as well as the comparison within each node can be performed fast. The path from the root to a leaf node, which is the cascaded results of the comparison, becomes the index of the codevector corresponding to an input vector.

The above search can be performed in \(O(\log N)\) time units on a sequential machine, since the tree has height \(O(\log N)\), assuming the input vector is available in the memory. \(O(N)\) memory is sufficient to implement the search algorithm, since each node has two words of data associated with it and the total number of nodes in the tree is \(O(N)\). The result of the search, which is a path in the tree, is the index of the desired codevector.
In this algorithm, multiplication operation is not required for any $L_q$ metric, $1 \leq q \leq \infty$, since each node in the tree compares a constant $c$ with an element of the input vector.

Compared with known tree search algorithms, the computational requirement of our search method is independent of the dimensionality of the input vectors. For a given $S \times T$ image, known tree search algorithms require $O(\frac{ST}{2} \times k \log N) = O(STk)$ operations, where $N = 2^r$, $0 < r < 1$. On the other hand, our search algorithm requires $O(ST)$ operations, since $\max(k, \log N)$ is $k$ in this case. Thus, our method is more efficient for large $k$.

We have performed a simulation of the above clustering and the search algorithm. Fig. 2 shows an original $256 \times 256$ image used as an input VQ system. $4 \times 4$ windows were employed on the original image. Fig. 3 shows the recovered image encoded at 0.5 bit/pixel, using $k = 16$, and $N = 256$.

![Figure 2: Original 256 x 256 lenna image](image1)

![Figure 3: Image encoded at 0.5 bit/pixel](image2)
4 A VLSI Architecture for Vector Quantization

In this section, we show an architecture based on the algorithm in Section 3.2. This design is suitable for single chip implementation in current VLSI technology. This is shown by an estimate of the die size using layout tools. For sake of simplicity, assume that the search tree in Section 3.2 has exactly \( \log N \) levels, numbered 0 to \( \log N - 1 \) (in reality, the number of levels is at most \( \log N + 1 \)). Also, notice that \( k \) is greater than \( \log N \), since \( N = 2^r \) and \( 0 < r < 1 \), in practice.

The organization of the proposed architecture is shown in Fig. 4. It consists of an external memory module, a Memory Index Register (MIR), and a PE. Information about the hyperplanes used in the search is contained in the external memory module. It is desirable to store the codebook externally, since some VQ schemes such as classified VQ, adaptive VQ, etc. need more than one codebook or require change of codebook during the computation. Also, it leads to significant reduction in on-chip area and allows the use of commercially available SRAM/DRAM chips. MIR stores the address of the location to be accessed in the external memory module using the result of the comparison in the PE. At the end of the tree search, it also contains the index corresponding to the input vector.

The information at each node of the tree are stored in the external memory as follows: the hyperplane index and the constant corresponding to the \( j \)th node (from the left) in the \( i \)th level of the tree is stored in the \( (2^i + j) \)th location, \( 0 \leq i \leq \log N - 1, 0 \leq j \leq 2^i - 1 \). The left (right) child of a node can be obtained by adding 0 (1) to the left shifted node index. Thus, the path from the root of the tree to a node corresponds to the binary representation of the index of the node. Each location occupies \( (w + \log k) \) bits: \( w \) bits for hyperplane constant, \( \log k \) bits for hyperplane index. Fig. 5 shows a memory organization obtained from such a mapping.

For analysis of the design, we assume that the data is loaded into the registers at the beginning of a clock cycle and the loaded data is available during the same cycle. For example, a two-phase clock (denoted \( \phi_1, \phi_2 \)) can be used to achieve this.

The PE has the following data ports: an input data channel \( D_i \), an index channel \( I_e \), a data channel \( H_c \) for the hyperplane constant, and a data channel \( H_i \) for the index of the hyperplane. \( D_i \) and \( H_c \) are \( w \)-bit wide. In practice, \( 8 \leq w \leq 12 \). \( H_i \) is \( \log k \)-bit wide. \( I_e \) is one-bit wide. In the following, we assume that the computation begins at \( t = 0 \) and a window denotes a block of \( k \) cycles starting at time \( t \).

The internal structure of the PE is shown in Fig. 6. The PE has a comparator, and \( O(k) \) registers for storing the input vector, the data from \( H_c \) and \( H_i \), and \( I_e \). \( B_d \) consists of \( k \) registers acting as a word level shift register. \( R_d \) consists of \( k \) registers. \( B_d \) is used as a data buffer. \( R_d \) stores the input to be used for comparison during \( \log N \) cycles over a window. The register \( R_e \) stores the data from \( H_c \) which is the hyperplane constant and the register \( R_i \) stores the value from \( H_i \) which is the hyperplane index. The operation of the PE is shown in Algorithm 2. Note that these operations are performed over \( k \) cycles.
Figure 4: The architecture
An element of the input vector is fed into the PE every cycle through the channel $D_c$. During the $j$th cycle of a window, $0 \leq j \leq \log N - 1$, the PE fetches the hyperplane information from the external memory through $H_c$ and $H_i$, and compares an element of the input vector and the hyperplane constant. The PE outputs the result of the comparison onto $I$. This one bit signal is loaded into the Least Significant Bit (LSB) of $MIR$. The contents of $MIR$ is used to fetch the data corresponding to the next level of the tree. During the $(\log N)$th cycle of each window, the output is available in $MIR$. During the last cycle of each window, $MIR$ is set to "1" to access the root of the search tree for the next input vector.

The latency of the array is $k + \log N$ time units. The throughput rate of the above design is $1/k$, since an index corresponding to an input vector is output every $k$ cycles after the array is filled. Thus, it can operate at input data rate. For an input image of $512 \times 512$ pixels/frame with 8 bits/pixel at 30 frames/sec, the proposed search algorithm requires less than 8 MOPS. Thus, the proposed architecture can operate in real-time, since it is easy to achieve over 20 MOPS in such an architecture.

Estimate of the die size of the chip (including the PE and $MIR$) in the above architecture was performed using layout tools. Standard cell approach [22] was used to generate a layout, assuming $k = 16$, $N = 2^{12}$, and $w = 8$. The die size was found to be $2.2 \times 3.3 \text{mm}$ using 1.25 CMOS technology. Most silicon area is consumed by the registers and wires. The longest path in the design was from $R_d$ to the comparator. The
for $j = 0$ to $k - 1$
  begin
    Right shift $D_j$ and $D[0] \leftarrow D_j$; //use leading edge of $\phi_1$
    if $j \leq \log N$ then
      begin
        Left shift MIR;
        Compare $R[0][i]$ and $R_o$ and store into LSB of MIR; //use leading edge of $\phi_2$
      end
    if $j = k - 1$
      begin
        $D[0][k - 1] \leftarrow D[0][k - 1] [//use leading edge of $\phi_2$
        MIR $\leftarrow 1$; $j \leftarrow 0$;
      end
  end

Algorithm 2: Operation of the PE
delay time in the longest path was found to be less than 25ns using SPICE simulation. The number of I/O pins in the proposed design is 40 including signal and power ports. The number of I/O pins can be reduced into 28 by using off-chip MIR. The proposed design is more compact, compared with the known designs in the literature [9] [12].

5 Conclusion

Using a new tree search algorithm based on the clustering algorithm in [10], we have proposed an architecture for real-time VQ which has a simple PE organization and an external memory. The proposed architecture requires less memory (by a factor of $O(\delta)$) compared with the known architectures [9] [12] [19]. It is feasible to implement the architecture using a single chip and external memory. Table 1 shows a comparison of the features of the proposed architecture with those of the known architectures in the literature.

<table>
<thead>
<tr>
<th>Design in</th>
<th>Number of PEs</th>
<th>Multiplication Operation</th>
<th>Complexity of a PE</th>
<th>Memory Size</th>
<th>I/O Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10]</td>
<td>$O(kN)$</td>
<td>Yes</td>
<td>Low</td>
<td>$O(kN)$</td>
<td>$O(N + k)$</td>
</tr>
<tr>
<td>[7]</td>
<td>$O(k)$</td>
<td>Yes</td>
<td>High</td>
<td>$O(kN)$</td>
<td>$O(4w)$</td>
</tr>
<tr>
<td>[9]</td>
<td>$O(\log N)$</td>
<td>Yes</td>
<td>High</td>
<td>$O(kN)$</td>
<td>$O(\log N + \log^* N)$</td>
</tr>
<tr>
<td>[12]</td>
<td>$O(\log N)$</td>
<td>Yes</td>
<td>High</td>
<td>$O(kN)$</td>
<td>$O(\log N + \log^* N)$</td>
</tr>
<tr>
<td>This paper</td>
<td>1</td>
<td>No</td>
<td>Low</td>
<td>$O(N)$</td>
<td>$O(2w + \log k + \log N)$</td>
</tr>
</tbody>
</table>

Table 1: Comparison with related works

References


