Hardware Accelerator for Vector Quantization by Using Pruned Look-Up Table

Conference Paper *in* Lecture Notes in Computer Science · May 2005
DOI: 10.1007/11424925_105 · Source: DBLP

4 authors, including:

Pi-Chung Wang
National Chung Hsing University
110 PUBLICATIONS **363** CITATIONS
SEE PROFILE

Chun-Liang Lee
Chang Gung University
44 PUBLICATIONS **147** CITATIONS
SEE PROFILE

Hung-Yi Chang
National Kaohsiung First University of Science and Technology
33 PUBLICATIONS **121** CITATIONS
SEE PROFILE

Hardware Accelerator for Vector Quantization by Using Pruned Look-Up Table

Pi-Chung Wang[1], Chun-Liang Lee[1], Hung-Yi Chang[2], and Tung-Shou Chen[3]

[1] Telecommunication Laboratories, Chunghwa Telecom Co., Ltd.
7F, No. 11 Lane 74 Hsin-Yi Rd. Sec. 4, Taipei, Taiwan 106, R.O.C.
{abu, chlilee}@cht.com.tw
[2] Department of Information Management,
I-Shou University, Kaohsiung, Taiwan 840, R.O.C.
leorean@isu.edu.tw
[3] Institute of Computer Science and Information Technology,
National Taichung Institute of Technology, Taichung, Taiwan 404, R.O.C.
rcchens@ntit.edu.tw

Abstract. Vector quantization (VQ) is an elementary technique for image compression. However, searching for the nearest codeword in a codebook is time-consuming. The existing schemes focus on software-based implementation to reduce the computation. However, such schemes also incur extra computation and limit the improvement. In this paper, we propose a hardware-based scheme "*Pruned Look-Up Table*" (PLUT) which could prune possible codewords. The scheme is based on the observation that the minimum one-dimensional distance between the tested vector and its matched codeword is usually small. The observation inspires us to select likely codewords by the one-dimensional distance, which is represented by bitmaps. With the bitmaps containing the positional information to represent the geometric relation within codewords, the hardware implementation can succinctly reduce the required computation of VQ. Simulation results demonstrate that the proposed scheme can eliminate more than 75% computation with an extra storage of 128 Kbytes.

1 Introduction

VQ is an important technique for image compression, and has been proven to be simple and efficient [1]. VQ can be defined as a mapping from k-dimensional Euclidean space into a finite subset C of R^k. The set C is known as the *codebook* and $C = \{c_i | i = 1, 2, \ldots, N\}$, where c_i is a *codeword* and N is the codebook size. To compress an image, VQ comprises two functions: an encoder and a decoder. The VQ encoder first divides the image into $N_w \times N_h$ blocks (or vectors). Let the block size be k ($k = w \times h$), then each block is a k-dimensional vector. VQ selects an appropriate codeword $c_q = [c_{q(0)}, c_{q(1)}, \ldots, c_{q(k-1)}]$ for each image vector $x = [x_{(0)}, x_{(1)}, \ldots, x_{(k-1)}]$ such that the distance between x and c_q is

O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3483, pp. 1007–1016, 2005.

the smallest, where c_q is the closest codeword of x and $c_{q(j)}$ denotes the jth-dimensional value of the codeword c_q. The distortion between the image vector x and each codeword c_i is measured by the *squared Euclidean distance*, i.e.,

$$d(x, c_i) = \|x - c_i\|^2 = \sum_{j=0}^{k-1} [x_{(j)} - c_{i(j)}]^2. \quad (1)$$

After the selection of the closest codeword, VQ replaces the vector x by the *index* q of c_q. The VQ decoder has the same codebook as that of the encoder. For each index, VQ decoder can easily fetch its corresponding codeword, and piece them together into the decoded image.

The codebook search is the major bottleneck in VQ. From equation (1), the calculation of the squared Euclidean distance needs k subtractions and k multiplications to derive k $[x_{(j)} - c_{i(j)}]^2$s. Since the multiplication is a complex operation, it increases the total computational complexity of equation (1). Therefore, speeding up the calculation of the squared Euclidean distance is a major hurdle. Furthermore, an efficient hardware implementation is also attractive to reduce the VQ computation.

Many methods have been proposed to shorten VQ encoding time [2,3,4,5,6]. These schemes emphasize computation speed, table storage and image quality. The existing schemes focus on software-based implementation to reduce the computation. However, such schemes also incur extra computation and limit the improvement. Moreover, these schemes did not utilize the geometrical information implied in the codewords. In this work, we propose an adaptive scheme "*Pruned Look-Up Table*" (PLUT) which selects the computed codewords. The new scheme uses bitmaps to represent the geometric relation within codewords. Accordingly, the search procedure could refer the information to sift unlikely codewords easily. Since the lookup procedure is simple enough, the proposed scheme is suitable for hardware implementation. With the bitmaps containing the positional information to represent the geometric relation within codewords, the hardware implementation can succinctly reduce the required computation of VQ. Simulation results demonstrate the effectiveness. The rest of this paper is organized as follows. The proposed scheme and implementation are presented in Section 2. Section 3 addresses the performance evaluation. Section 4 concludes the work.

2 PLUT Method

To compress an image through VQ, the codebook must be generated first. The codebook is gathered through approach, like the Lindo-Buzo-Gray (LBG) algorithm [7], based on one or multiple images. The quality of the compressed images ties to whether the codebook is well trained, i.e., the squared Euclidean distance between the tested vector and the matched codeword in the adopted codebook is small. Thus, a well trained codebook could improve the compression quality. As implied in the equation of squared Euclidean distance calculation,

a well-trained codebook can lead to the implication that the one-dimensional distance, $|x_{(j)} - c_{M(j)}|$ where $0 \leq j \leq k-1$, between the tested vector x and the matched codeword c_M should be relatively small.

To further verify our assumption, the distribution of the smallest one-dimensional distance $\min_{j=0}^{k-1} |x_{(j)} - c_{M(j)}|$ between the tested vectors and their matched codewords is presented in Fig. 1. The codebook is trained according to the image "Lena", then the six images are compressed by full search VQ. The quality of the images are estimated by the *peak signal-to-noise ratio* (PSNR), which is defined as PSNR=$10 \cdot log_{10}(255^2/MSE)$ dB. Here the *mean-square error* (MSE) is defined as MSE= $(1/m)^2 \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} [\alpha_{(i,j)} - \beta_{(i,j)}]^2$ for an $m \times m$ image, where $\alpha_{(i,j)}$ and $\beta_{(i,j)}$ denote the original and quantized gray level of pixel (i, j) in the image, respectively. A larger PSNR value has been proven to have preserved the original image quality better. For the compressed images with better quality, including "Lena" and "Zelda", most of their smallest one-dimensional distances are less than 8. Furthermore, 99% smallest one-dimensional distances are less than 4. However, the ratio is reduced to 93% $\sim$ 97% for the other images since their quality of compression is also decreased.

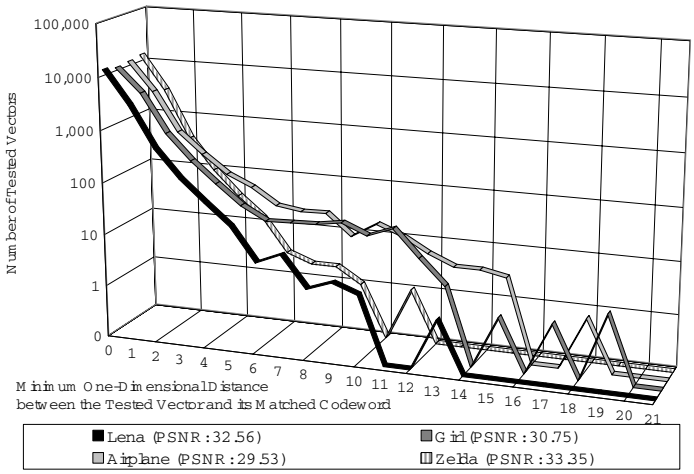

Fig. 1. The Distribution of the Smallest One-dimensional Distance for Different Images

We use a two-dimensional VQ as an example. There are two codewords, C_1 (3, 1) and C_2 (2, 2). To calculate the nearest codeword for the tested vector, V_1 (1, 2), the squared Euclidean distances to C_1 and C_2 are 4 and 2, respectively. Hence C_2 is chosen as the result. Also, C_2 is the nearest codeword for V_2 at (2,3).

Since the smallest one-dimensional distance between the tested vector and the selected codeword is small with a well-trained codebook, the property can be utilized to fasten VQ computation. Our idea is to represent the positional information by bitmaps and refer the bitmaps to select likely codewords. For each

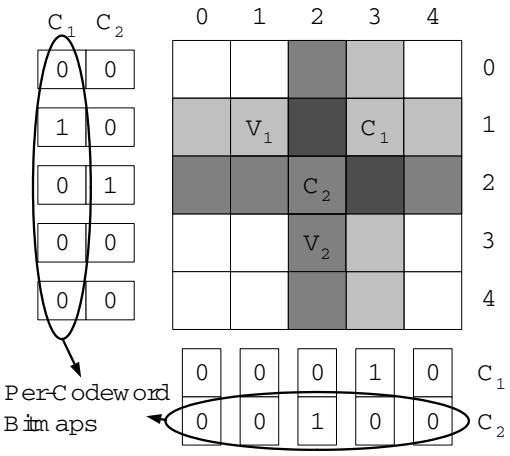

Fig. 2. Two-dimensional Per-Codeword Bitmaps ($R = 0$)

codeword i, we adopt k per-codeword bitmaps to record their positional information for each dimension. Each bitmap consists of m bits to correspond every position. The b_{th} bit in the per-codeword bitmap for dimension j of codeword i is set to one if b is within a certain range of $c_{i(j)}$, say R. The per-codeword bitmaps for the previous example are shown in Fig. 2. The range R is equal to zero. For the first tested vector V_1, it is within the designated range of C_1 in the vertical dimension, and only C_1 is considered for vector quantizing. Similarly, V_2 is within the range of C_2 in the horizontal dimension. Thus C_2 is selected as the closest codeword for V_2 directly.

Although the scheme could sift likely codewords easily, it is not totally accurate. In Fig. 2, C_1 is presumed as the closest codeword for V_1. However, C_2 is the one with the smallest Euclidean distance to V_1, and *false match* is caused. In addition, two kinds of bricks would cause problems: unoccupied bricks (e.g. bricks at (0,0) or (1,3)) and repeatedly occupied ones (e.g. bricks at (2,1) or (3,2)). If the tested vectors locate in the unoccupied bricks, they are not assigned to any codeword, i.e. every codeword must be computed to decide the closest one, and there is no speedup. For the vectors locating in the repeatedly occupied bricks, the codewords whose range occupies the vectors would be calculated for the Euclidean distance, thus the speedup is lessened.

To less the problem, a wider range could be adopted, as shown in Fig. 3 where the renewed bitmaps for $R = 1$ are presented. With the new range, most bricks are occupied by at least one codeword's square. However, the conjunct bricks are also increased due to the larger occupied region.

A suitable range is thus important to the performance of the proposed scheme since a wider range will increase the number of candidates while a narrow range might result in a null set. In our experiments, various ranges are investigated to evaluate the performance and the image quality. Next, the construction/lookup procedure of the searchable data structure is introduced.

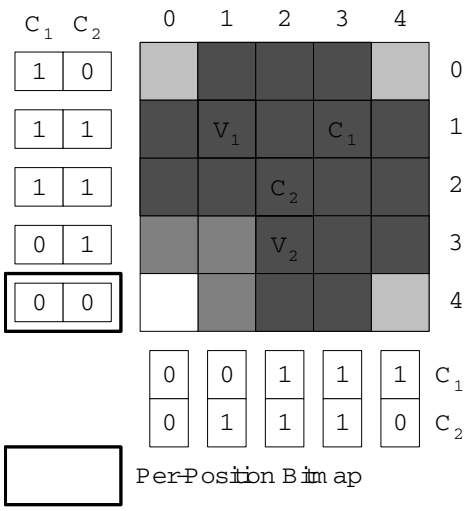

Fig. 3. Two-dimensional Per-Codeword Bitmaps ($R = 1$)

2.1 The Construction of the Searchable Data Structure - Positional Bitmaps

Although the per-codeword bitmaps could present the positional information, they are not searchable. This is because accessing bitmaps for each codeword is inefficient. To utilize the bitmaps based on the proposed concept, the per-position bitmaps are generated from the per-codeword bitmaps. In Fig. 3, we also illustrate the relationships between the per-position bitmaps and the per-codeword bitmaps.

The per-position bitmap for position p at dimension j is defined as $B^R_{j,p}$, where D is the preset range. The i_{th} bit is defined as $B^R_{j,p}(i)$ which is set to one if $p - R \leq c_{i(j)} \leq p + R$. The pseudo code is given in Fig. 4. For each range R, the required storage is $m \times N$ per dimension. With a typical 16-dimensional codebook with 256 entries and 256 gray levels, the occupied memory is 128 Kbytes.

Bitmap-Filling Algorithm
For each dimension $j, \forall j \in \{0, k-1\}$ **BEGIN**
  For each position $p, \forall p \in \{0, m-1\}$ **BEGIN**
    For each codeword $i, \forall i \in \{0, N-1\}$ **BEGIN**
      If $p - R \leq c_{i(j)} \leq p + R$, $B^R_{j,p}(i) = 1$.
      Otherwise, $B^R_{j,p}(i) = 0$.
    END
  END
END

Fig. 4. Bitmap-Filling Algorithm

2.2 The Lookup Procedure

The **PLUT** scheme combines bitmap pruning and **TLUT** to achieve fast processing. For a tested vector, the j_{th} value x_j is used to access the bitmap B^R_{j,x_j}. Each set bit indicates that the corresponding codeword is within a range R from the tested vector at dimension j. Accordingly, the Euclidean distance is calculated by accessing TLUT. The pseudo code for lookup procedure is listed in Fig. 5. First, the multiple bitmaps are performed **OR** operations to derive the representative bitmap D^R. To check whether the i_{th} bit in D^R is set, we further perform **AND** operation with D^R and a pre-generated bitmap with only i_{th} bit set $(00\ldots010\ldots0)$. If the value is larger than zero, then codeword i is one of the candidate.

Vector Quantization by PLUT Algorithm
For each vector x **BEGIN**
  Fetch the B^R_{j,x_j}, where $j \in dim$.
  $D^R = \bigcup_{j\in\{0,k-1\}} B^R_{j,x_j}$.
  For each set bit $D^R(i)$ **BEGIN**
    Calculate Euclidean distance $d(x, c_i)$ where $d(x, c_i) = \sum_{j=0}^{k-1} TLUT_1[|x_{(j)}, c_{i(j)}|]$.
    If $d(x, c_i) \leq min\_distance$ **BEGIN**
      $min\_distance\_id = i$
      $min\_distance = d(x, c_i)$
    END
  END
  $min\_distance\_id$ is the quantized index for x.
END

Fig. 5. Vector Quantization by PLUT Algorithm

We use the previous example in Fig. 2 to explain the procedure, where $R = 0$. For the tested vector V_1 "11", the second per-position bitmap "00" at x-axis and second one "10" at y-axis are fetched. The representative bitmap "10" is derived by performing **OR** to these two bitmaps. Consequently, the representative bitmap is performed **AND** operation with "10" to indicate that the first codeword is one of the candidate and the computation for the squared Euclidean distance between V_1 and C_1 is thus carried out. Next, the representative bitmap is performed **AND** operation with "01" again. Since no set bit is found in the resulted bitmap, the calculation for the squared Euclidean distance between V_1 and C_2 is omitted.

2.3 Hardware Implementation

The hardware implementation is preferable for the PLUT scheme. This is because PLUT requires memory bus with N-bit wide (typically $N = 256$). Even in the modern software platform, the memory bus is less than 128 bits. In Fig.

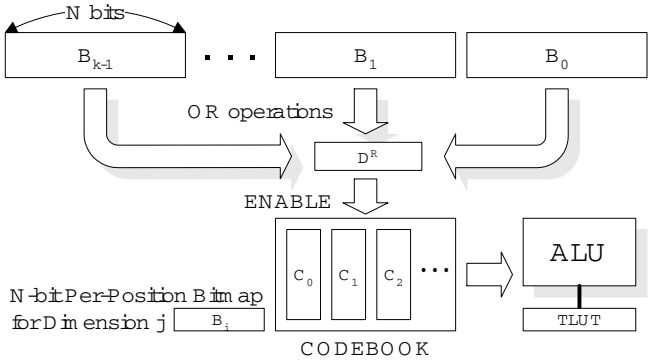

Fig. 6. Hardware Implementation of PLUT Scheme

6, we present a conceptual model for the hardware implementation. This implementation includes K independent RAM modules for per-position bitmaps. Bitmap of each dimension is located in a storage. To perform the search, the per-position bitmaps are fetched from RAM modules simultaneously and performed **OR** operation. Then, the resulted bitmap D^R enables the codewords in the candidate for calculating the Euclidean distance in ALU. Notably, this architecture is suitable for parallelized hardware or pipelining.

3 Performance Evaluation

We have conducted several simulations to show the efficiency of PLUT. All images used in these experiments were 512×512 monochrome still images, with each pixel of these images containing 256 gray levels. These images were then divided into 4×4 pixel blocks. Each block was a 16-dimensional vector. We used image "Lena" as our training set to generate codebook C. In the previous literature [1, 2], the quality of an image compression method was usually estimated by the following five criteria: compression ratio, image quality, execution time, extra memory size, and the number of mathematical operations. All of our experimental images had the same compression ratio, hence only the latter four criteria are listed to evaluate the performance of the proposed scheme. The quality of the images are estimated by the PSNR, which is addressed in Section 2. The extra memory denotes the storage needed for executing PLUT scheme. As for the mathematical operations, the number of the calculated codewords is also considered since the operations for each codeword are identical. In addition, the compression time is evaluated based on software implementation since the performance of hardware implementation can be illustrated from the number of calculated codewords.

The decompressed images based on the PLUT scheme with different ranges are shown in Fig. 7. Basically, the image quality of PLUT is improved gradually as the range increases, such as the PSNR value for $R = 0$ is worse than that

(a) R = 0 (PSNR=29.852) (b) R = 1 (PSNR=31.254)

(c) R = 2 (PSNR=31.823) (d) R = 4 (PSNR=32.500)

Fig. 7. The Decompressed Lena Images of PLUT Scheme

for $R = 1$ and $R = 2$. However, the quality of some area shows different trend, as shown in the circles of Fig. 7(a),7(b),7(c). This is mainly because for several blocks, there is no candidate derived by PLUT with $R = 0$, thus full search is executed for these blocks. As the range increases to 1 or 2, some codewords are selected for calculation of Euclidean distance. Nevertheless, the codewords cannot yield better precision than full search. The occurrence of such faults ties to the quality of the used codebook. Also, these faults can be alleviated by adopting larger range or enabling full search as the squared Euclidean distance is larger than a certain value. As shown in Fig. 7(d), the image quality is almost identical to VQ and TLUT while PLUT range is enlarged to 4.

The performance of the software-based implementation is illustrated in Table 1. The experiments were performed on an IBM PC with a 500-MHz Pentium CPU. VQ indicates the vector quantization without any speedup. The ranges

for PLUT vary from 0 to 8. With a smaller range, the image quality is degraded since the occurrence of false matches is increased. Nevertheless, the calculated codewords are reduced by the per-position bitmaps, the execution time is lessened as well. Full search requires no extra storage while TLUT needs 256 bytes. For PLUT scheme, the extra storage is 128 Kbytes for bitmap and 256 bytes for TLUT. If the hardware implementation is considered, the bitwise operation cab be parallelized to further shorten the vector quantizing time.

Table 1. The Performance of PLUT with Different Ranges

Lena	Full Search	TLUT	PLUT Scheme				
			R=0	R=1	R=2	R=4	R=8
PSNR	32.56	32.56	29.85	31.25	31.82	32.50	32.55
Time (sec.)	1.30	1.09	0.23	0.42	0.53	0.67	0.83
Codewords	256	256	19	44	59	78	99
Storage (byte)	0	256	128K (PLUT) + 256 (TLUT)				

Table 2 illustrates the performance of PLUT based on different images. For the images with better compression quality in full search, PLUT generates more candidates since the codewords are usually close to the compressed blocks. While the range is enlarged to 4, PLUT can derived compressed images with comparable quality to full search while requiring only half execution time.

Table 2. The Performance of PLUT based on Different Images (N=256)

Images	Lena			Girl			Airplane			Zelda		
Metrics	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR	Code-words	Time	PSNR
Full Search	256	1.30	32.56	256	1.30	30.75	256	1.30	29.53	256	1.30	33.35
TLUT	256	1.09	32.56	256	1.11	30.75	256	1.11	29.53	256	1.09	33.35
PLUT,R=0	19	0.23	29.85	17	0.21	29.08	14	0.18	27.57	20	0.24	31.98
PLUT,R=1	44	0.42	31.25	40	0.39	30.14	32	0.33	28.86	44	0.42	33.06
PLUT,R=2	58	0.53	31.82	54	0.50	30.35	44	0.41	29.15	59	0.54	33.25
PLUT,R=4	78	0.67	32.50	72	0.64	30.45	58	0.52	29.35	78	0.67	33.32

In summary, with $R = 2$, the proposed scheme can reduce more than 50% computation without losing image quality. If a hardware implementation is adopted, 25% computation can be further eliminated since only a fourth of codewords are calculated for squared Euclidean distance. Therefore, only a fourth of computation is required.

4 Conclusion

In this study, we present a new novel algorithm "PLUT" for codebook search in VQ. The new scheme is based on the observation that the minimal one-dimensional distance between the tested vector and the matched codeword is usually small. To represent the geometrical information, PLUT adopts bitwise data structure, which is simple and storage efficient. By setting a given range, the PLUT can sift out unfeasible codewords easily, hence it is suitable for hardware implementation. A conceptual hardware implementation is also revealed. Since the performance of PLUT ties to the quality of codebook, PLUT is suitable for high-quality image compression. The performance evaluation further demonstrates that 75% computation can be reduced with an extra 128 Kbytes storage.

References

1. Gersho, A., Gray, R. M.: Vector Quantization and Signal Compression. Boston, MA: Kluwer (1992).
2. Chen, T. S., Chang, C. C.: An Efficient Computation of Euclidean Distances Using Approximated Look-Up Table. IEEE Trans. Circuits Syst. Video Technol., Vol. 7 (2000) 594-599.
3. Davidson, G. A., Cappello, P. R., Gersho A., Systolic architectures for vector quantization, IEEE Trans. Acoust. Speech, Signal Processing, Vol. 36 (1988) 1651-1664.
4. Park, H., Prasana, V. K.: Modular VLSI architectures for real-time full-search-based vector quantization. IEEE Trans. Circuits Syst. Video Technol., Vol. 3 (1993) 309-317.
5. Ramamoorthy, P. A., Potu, B., Tran, T.: Bit-serial VLSI implementation nof vector quantizer for real-time image coding. IEEE Trans. Circuits Syst., Vol. 36 (1989) 1281-1290.
6. Rizvi, S. A., Nasrabadi, N. M.: An efficient euclidean distance computation for quantization using a truncated look-up table. IEEE Trans. Circuits Syst. Video Technol., Vol. 5 (1995) 370-371.
7. Linde, Y., Buzo, A., Gray, R. M.: An algorithm for vector quantizer design. IEEE Trans. Communications, Vol. 28 (1980) 84-95.
8. Chang, H. Y., Wang, P. C., Chen, R. C., Hu, S. C.: Performance Improvement of Vector Quantization by Using Threshold. Lecture Notes in Computer Science, Vol. 3333 (2004) 647-654.