Real Time Low Bit-Rate Video Coding Algorithm Using Multi-Stage Hierarchical Vector Quantization

K.Terada, M.Takeuchi, K.Kobayashi, K.Tamaru

Department of Electronics and Communication
Kyoto University, Japan
Email: terada@tamaru.kuee.kyoto-u.ac.jp

ABSTRACT

In this paper, we propose a low bit-rate coding algorithm for wireless communication based on multi-stage hierarchical vector quantization, motion compensation and differential pulse code modulation. Our method adapts bit allocation to spatial and temporal correlation. Conventional schemes based on discrete cosine transform (DCT) need a large amount of computation on both encoding and decoding. On the other hand, our proposed method consists of addition, subtraction and shift operation. It does not use multiplication. It can decode in real time on a conventional serial processor. Encoding by vector quantization (VQ), however, consumes a large amount of computation. We developed a new LSI to accelerate VQ. Our scheme can send 10 frames of QCIF video sequences through a 29.2kbps line. The quality of reconstructed image is over 30dB.

1. INTRODUCTION

Personal communication system (PCS) transceivers have widely spread. In the near future personal digital assistants (PDA) will become a complete voice/video phone transceiver. We are concerned about communication issues with PDA. Standard video codecs, such as MPEG1, MPEG2, H.261 and H.263, are based on discrete cosine transform (DCT). They consume a large amount of computation on both encoding and decoding. Thus they are not suitable for communication with PDA.

Vector Quantization (VQ) has proven to be a powerful technique for low bit rate image coding [1]. Compared with DCT-based techniques, a video sequence compressed by VQ can be easily decompressed and has high compression efficiency. On encoding, however, it consumes so much computation. Thus we developed a functional memory type parallel processor for VQ including 64 PEs (FMPP-VQ64) [2]. FMPP-VQ64 operates at 25MHz clock frequency and performs vector quantization to over 50,000 input vectors per second with 64 code vectors. FMPP-VQ64 is suitable for PDA, which power consumption is 66mW at 3.5V power supply.

Several VQ-based algorithms have been proposed for less computation and high compression ratio. For example, Hang and Haskell proposed interpolative VQ (IVQ) [3]. Their system sends a low resolution bilinear interpolated image across a side channel while vector-quantizing the residual. It can reduce blocking effect. Gersho and Shoham suggested hierarchical VQ (HVQ) technique [4]. They first introduced hierarchical structure into VQ-based algorithm. This method successively partitions large dimensional vectors into small dimensional sub-vectors. HVQ can exploit correlation in large dimensional vectors while avoiding the complexity obstacle of large dimensions. Ho and Gersho proposed multistage hierarchical VQ (MSHVQ) [5]. In multistage VQ (MVQ), after an original vector is vector-quantized, the residual vector which has the same dimension as the original one are quantized. MSHVQ technique uses various-dimensional vectors at each stage instead of fixed-dimensional vectors. They implemented multistage IVQ together to manage vector dimensions.

All the above VQ coding schemes were originally proposed for a still image. We will present a low bit-rate video coding algorithm based on MSHVQ. It enables simple bit rate control by adaptive bit allocation at each stage with small computational complexity.

In this paper, Section 2 gives a brief description of FMPP-VQ. Section 3 describes our new MSHVQ video coding scheme. Simulation results are presented in Section 4; and conclusions are drawn in Section 5.

2. FUNCTIONAL MEMORY TYPE PARALLEL PROCESSOR

FMPP-VQ64 is a Single Instruction system Multi Data stream (SIMD) parallel processor including 64 processor elements (PE). Each PE has 16 eight-bit words to store a 16-dimensional code vector in a codebook and an ALU to compute a distance between an input vector and a code one. These 64 PEs work simultaneously. FMPP-VQ64 can vector-quantize an input vector with 64 code vectors.

It is not desirable that the size of code vectors is limited to 64. While a small codebook enlarges distortions, a large codebook increases overhead bits to send. We generate a large codebook from a small codebook to rearrange elements in a code vector. We can generate 1024 code vectors from 64 vectors, since a primitive code vector turns into 15 derivative code vectors (Fig.1).

3. CODING ALGORITHM

3.1 Multi-Stage Hierarchical VQ

The performance of VQ cannot be increased without the expansion of vector dimensions to reduce correlation of input vectors. Low active area should be partitioned into a large dimensional vector, and high one into a small one. However, computation and memory requirements explode according to vector dimensions. We adopt a hierarchical method to determine block sizes according to activity of each area, and the technique of decimation and interpolation to reduce computational cost.

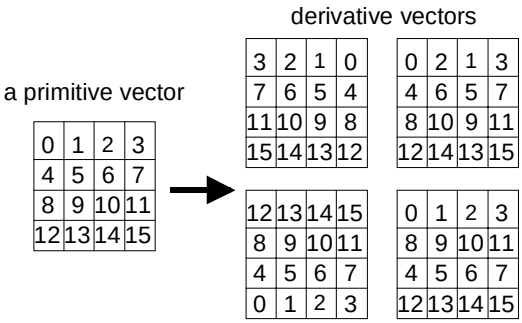

Fig.1: rearrangement of code vectors

Images are hierarchically compressed in four stages as shown in Fig.2. At Stage 1, a value that represents a 16×16 block is scalar-quantized and transmitted. At Stage 2 16 values each of which represents a 4×4 block are vector-quantized. At Stage 3 16 values each of which represents a 2×2 block are vector-quantized. At Stage 4, a 4×4 block is vector-quantized. We decimate 256 pixels into 16 at Stage 1, since low active area does not require high resolution. At the following stage vector dimensions is also reduced to 16. The vector dimensions is 16 all through the stages to share the same codebook.

Stage 1 performs decimation to obtain a value represented a16×16 block. The easiest way is spatial subsampling, which causes aliasing errors. One of other approaches is using the mean value of each block, which brings a blocking effect of a 16×16 square block. Our scheme uses the mean values of 8×8 blocks, located at the upper left corner of 16×16 blocks. On decoding, values in inter blocks are linear-interpolated. This method reduces blocking effect to an 8×8 square block. Its computational cost is smaller than general low-pass filter such as Quadrature Mirror Filter (QMF) [6], Symmetric Short Kernel Filter (SKFF) [7] and so on.

The decimation schemes at the subsequent stages are equivalent to above one. At Stage 2 the mean values of 2×2 blocks at the upper left corner of each 4×4 block constitutes a 4×4 vectors to be vector-quantized. At Stage 3 the upper left corner pels of each 2×2 block constitute a 4×4 vector.

3.2 Motion Compensation

There exists spatial and temporal correlation in video signals. Spatial correlation is exploited by VQ method, while interframe prediction technique takes advantage of temporal correlation. We use differential pulse code modulation (DPCM) and motion compensation (MC) as prediction method. In MC, full search block matching algorithm (BMA) is most popular, but it has large computational complexity. We use orthogonal search method [8] for the purpose of compressing in real time. This technique excels in the way of convergence, a few steps and robustness to noise. A motion vector (MV) is determined for each 16×16 block. The MC search window is fixed to 8×8 pels around the center of each block.

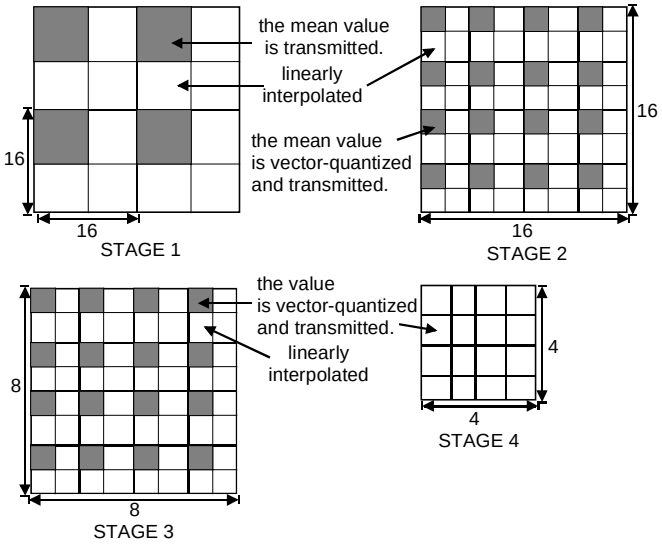

Fig.2: decimation and interpolation

3.3 Adaptive Algorithm For Video Coding

The coding algorithms of the first frame is shown in Fig.3. An image is first partitioned into 16×16 blocks and then the values represented the blocks are scalar-quantized. Differential signals between a decimated original image and an decimated interpolative surface are partitioned into 4×4 vectors at Stage 2. In the original MSHVQ of Ho and Gersho the differential signals between each stage are interpolated and decimated, while in our method original image and reconstructed image are individually decimated. This is because neighbor pels of the differential image are less correlative than those of original image and reconstructed image.

At Stage 2, each vector is vector-quantized and transferred. But not all vectors are transmitted. The algorithm to choose transmitted vectors is as follows.

(1) Calculate these two kinds of sum of absolute difference (SAD).

$$SAD_{original} = \sum_{i=1,\,j=1}^{16,16} |original - stage\,1|$$

$$SAD_{stage\,2} = \sum_{i=1,\,j=1}^{16,16} |stage\,2 - stage\,1|$$

$SAD_{stage\,2}$ gives the contribution of the vectors at Stage 2 to the improvement of image quality. The excess of $SAD_{original}$ over $SAD_{stage\,2}$ indicates the degree of the lowering of image quality brought by the transmission. The computation of $SAD_{stage\,2}$ needs the reconstructed image of Stage 2. To obtain the image, all differential blocks should be vector-quantized, but it requires a large computation. Thus we neglect VQ in computing $SAD_{stage\,2}$.

(2) All blocks are rearranged in descending order of $SAD_{stage\,2}$. These rearranged block are vector-quantized and transmitted until transmitted bit exceeds the allowable amount (2920 bit).

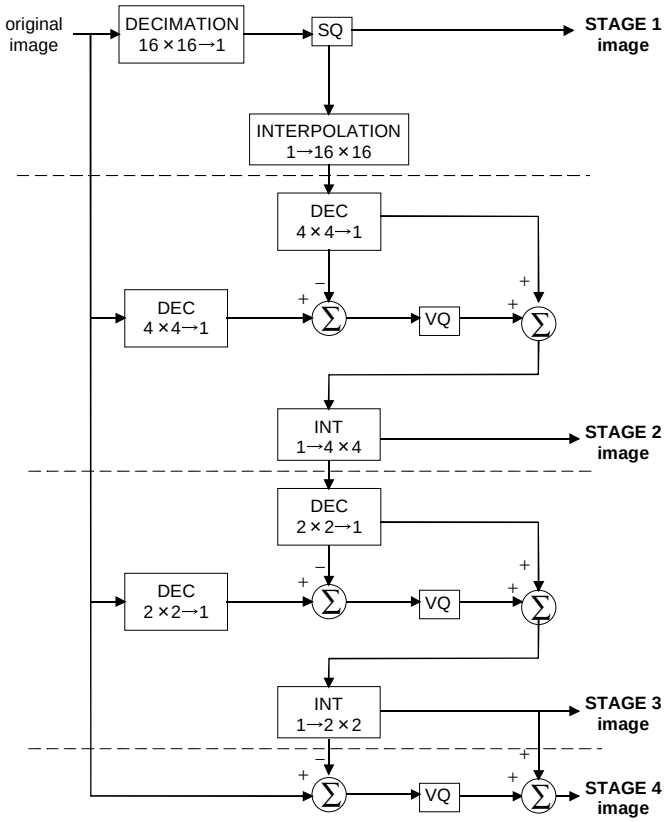

Fig.3: Block diagram of a encoder at the first frame

(3) If $SAD_{original} < SAD_{stage\,2} + threshold_{stage2}$,

the block is not transmitted at (2). In this case, output signals at Stage 2 lower the quality of the decoded image. The variable of $threshold_{stage2}$ is aimed to compensate errors brought by omitted VQ.

Then the differential signals between the decimated original image and the decimated interpolative surface are partitioned into 4×4 vectors at Stage 3. Each 4×4 vector is vector-quantized, and transmitted if the same condition as above is satisfied. The residual signals are parted into 4×4 blocks at Stage 4. The blocks are vector-quantized and transmitted if the condition (2) is satisfied. The condition (3) is not applied because of no decimation at the stage.

We explain the coding algorithm of the subsequent frames. The scheme is equivalent to that of the first frame except for using the motion compensated frame instead of the reconstructed frame at Stage 1. In MC, the following rule determines motion vectors to be transmitted.

(a) Calculate the following parameters.

$$SAD(x,y) = \sum_{i=1,\,j=1}^{16,16} \left| original - previous \right| \quad (x, y = \text{"up to } \pm 16\text{"})$$

$$SAD_{mc} = \min SAD(x,y) \quad ((x,y) \neq (0,0))$$

(b) MC mode is chosen if:

$$SAD(0,0) + threshold_{mc} > SAD_{mc}$$

The intention of $threshold_{mc}$ gives priority to the zero vector.

The bit allocation strategy above is robust to the change of video activity. Video coding based on DPCM forces a lot of signals to be transmitted in high video activity. We can take two strategies when transmitting through fixed bit-rate. One of them is to reduce temporal resolution, while the other is to decrease spatial resolution. The latter is better because video impression suffer from the decline of temporal resolution in high video activity. Images at low level stages, which have low spatial resolution, are first transmitted in our method. It works conveniently in high activity. The transmission in low video activity, however, lowers quality of a reconstructed image. Thus we impose the above condition (3) to prevent it. Our method can offer the way to adapt spatial resolution to video activity.

3.4 Codebook Design

Codebook design is one of the most important factors in VQ. We use a simple way to generate codebook in order to guarantee real-time encoding. An initial codebook is designed using the so-called 'pluning' method [1], where a small codebook is derived from a training sequence. We take vectors of 64 greater $SAD_{stage\,2}$ as an initial codebook out of all vectors at Stage 2. This is because the surface at Stage 2 of the first frame seriously affects the quality of subsequent frames. These initial primitive code vectors are transmitted at first. We mention the method to update the codebook. Only a single code vector is updated every frame. We use a modified Linde-Buzo-Gray algorithm [9] as the updating method. This technique is repeated until updated code vectors are convergent. In our approach LBG algorithm is applied once par frame. A primitive code vector creates fifteen derivative code vectors, which has the same mean and variance as the primitive one. Therefore, 16 vectors of all 1024 code vectors are updated, which have statistical property reflecting the property of the current input frame.

4. SIMULATION AND RESULT

Table 1 shows target variables on our simulations. The bit-rate of 29.2kbps is determined from the capacity of personal handy phone system (PHS) available in Japan. We have tested the performance of our method by encoding two widely known sequences: "Miss America" which has low spatial activity, and "Suzie" which has high temporal activity

Table 1: target variables

Image Size	176×144 pixels (QCIF)
Frame Rate	10Hz
Color	256 grade gray scale
Bit-rate	29.2kbps (0.115bpp)

Table 2 lists bit allocation at each stage. Flags indicate whether each block is encoded at each stage. At Stage 3 flags are

transmitted every 16×16 square block, and at Stage 4, every 8×8 square block. In our adaptive method mentioned above, every frame is compressed into a fixed size. The technique allocates MVs and the indexes of Stage 2 to a lot of bits in the frame of high video activity, while the indexes of Stage 3 and Stage 4 in the frame of low video activity.

Fig.4 shows the decoded video peak signal to noise ratio (PSNR) versus the video frame index performance for the two sequences. The graphs are not plotted during the transmission of the first three frames. This is because the initial codebook is transmitted. Fig.5 presents the bit allocation for the sequence of Suzie. Near the fifteenth frame the motion of Suzie is most active. The results indicate that transmitted bits are properly assigned for each stage.

In the sense of computational time, the compression of a frame except VQ takes 30ms on Pentium 200MHz. VQ is performed by FMPP-VQ64, which takes 48msec. Thus all coding time is 78msec. The decompression of a frame needs only 11msec on Pentium 200MHz.

5. CONCLUSION

We develop a low bit-rate video coding algorithm using a serial processor and FMPP-VQ64. The algorithm is based on MSHVQ, DPCM and MC. We incorporate adaptive bit allocation technique for video coding into MSHVQ, and adopt the combination of low computational technique of decimation and interpolation, orthogonal search MC and simple method of updating codebook, i.e. restricted LBG algorithm. Our scheme can encode QCIF images at 10 frames per second with maintaining quality of PSNR values of over 30dB at 29.2kbps.

Our current work is focussed on developing a real time low bit-rate image compression system for mobile videophone communication. The scheme mentioned here will soon realize such an attractive system.

6. REFERENCES

[1] A.Gersh and R.Gray, "Vector Quantization and Signal Compression", Boston: Kluwer Academic Publishers, 1992.

[2] K.Kobayashi, M.Kinoshita, M.Takeuchi, H.Onodera and K.Tamaru, "A Memory-based Parallel Processor for Vector Quantization: FMPP-VQ", *IEICE Trans. on Electron*, vol. E80-C, no.7, pp.970-975, 1997

[3] H.Hang and B.Haskell, "Interpolative Vector Quantization of Color Images", *IEEE Trans. on Comm*, vol. COM-36, pp.465-470, April 1988.

[4] A.Gersho and Y.Shoham, "Hierarchical Vector Quantization for Speech Coding", *ICASSP*, pp.10.9.1-10.9.4, April 1984.

[5] Y.Ho and A.Gersho, "Variable-Rate Multi-Stage Vector Quantization For Image Coding", *ICASSP*, pp.1156-1159, April 1988.

[6] D.Esteban and C.Galand, "Application of quadrature mirror filters to split band voice coding schemes", *ICASSP*, 1977

[7] D.Le Gall and Tabatabai, "Subband coding of digital images using symmetric short kernel filters and arithmetic coding techniques", *ICASSP*, 1982.

[8] A.Puri, H.M.Hang and D.L.Schilling, "An efficient block matching algorithm for motion-compensated coding", *ICASSP*, pp.1063-1066, Dallas, April 1987.

[9] Y.Linde, A.Buzo and R.M.Gray, "An algorithm for vector quantizer design", *IEEE Trans. On Comm*, vol. COM-28, pp.84-95, Jan. 1980.

Table 2: bit allocation

Stage 1	index	8 bit × 99 = 792 bit
Stage 2	flag	99bit
	index	10 bit × the number of encoded blocks
Stage 3	flag	99 bit
	index	10 bit × the number of encoded blocks
Stage 4	flag	1+396bit
	index	10 bit × the number of encoded blocks
MV	flag	99bit
	vector	8 bit × the number of MVs
Updating code vector		8 bit × 16 = 128 bit

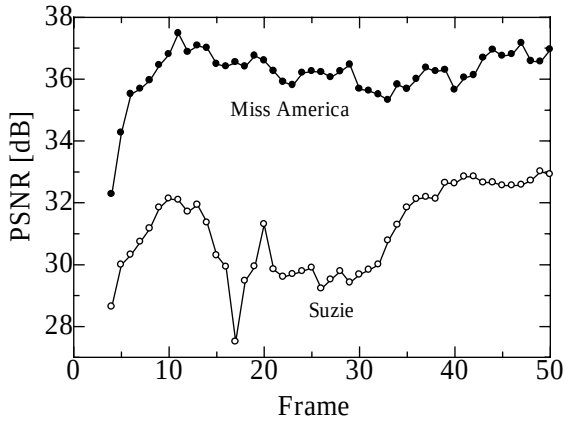

Fig.4: PSNR versus the frame index

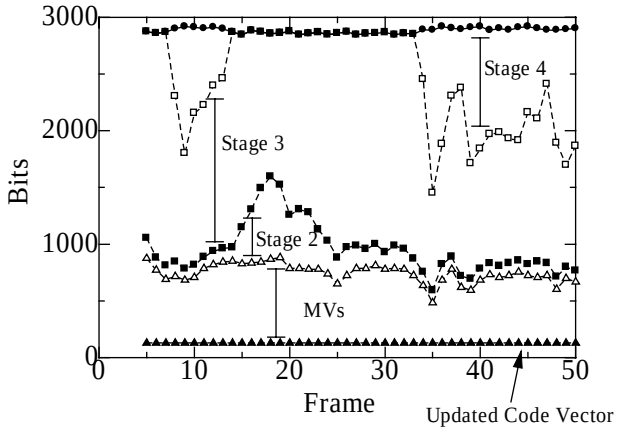

Fig.5: Bit Allocation for the sequence of Suzie