

REAL-TIME PC-BASED SOFTWARE IMPLEMENTATION OF H.261 VIDEO CODE

Den-Yuan Hsiau¹ and Ja-Ling Wu^{1,2}

¹Communication and Multimedia Lab.

Dept. of Computer Science and Information Engineering

National Taiwan University

²Dept. of Information Engineering National Chi-Nan University, PULI, 545, Taiwan

Abstract

The complexity of a video codec has made it seem impossible to accomplish an audio-visual application without using application specific integrated circuits (ASICs). However, recent advancing of computing power for microprocessors has made the development of software-based multimedia applications more feasible. More and more complicated tasks with heavy computation load can be realized on the basis of the general purpose CPU. In this paper, a PC-based software H.261 video codec, which is most strictly related to the real-time constraint applications such as video-phone and video-conference, is presented. The whole video codec system works will on Pentium-133.

I. Introduction

H.261 is a video coding standard recommended by the International Telecommunication union (ITU). Its

target application is to provide a real-time video transmission over the integrated service digital network (ISDN). The typical applications of H.261 codec are the video-phone and video-conference systems. H.261 is a hybrid DPCM/DCT video codec and the coding kernel contains the motion estimation, the motion compensation and the discrete cosine transform modules. Due to the intensive computation involved, the current H.261 related products are almost all developed with specific designed hardware or high-end workstation-based. Thanks to the rapid advance in CPU computing power, the PC-based real-time software H.261 codec (as is shown in this paper) becomes feasible and practical.

II. Real-time software H.261 decoder

To develop a real-time H.261 decoder, the following modules should be fast and efficient:

- (i). Variable length code (VLC) decoder.

(ii). Inverse discrete cosine transform. We will describe them in the following, respectively.

A) High speed variable length code decoder (VLD)

In Huffman coding, fixed-length input symbols are mapped into variable-length codewords according to the frequencies of occurrence of the symbols. The Huffman encoding process is relatively straightforward. The symbol-to-codeword mapping table provided by the probability-based modeler is used to generate the codewords for each input symbol. On the other hand, the Huffman decoding process is somewhat more complex.

(i). Table look-up decoder:

H.261 utilizes the Huffman code to perform variable length coding. The most convenient way to implement VLD is to build a VLC table according to the syntax of the video bit stream. Since the Huffman coding is a kind of instantaneous coding and no code is the prefix of the others, we extend each variable length codeword to be a fixed length one with the original codeword appeared in the most significant bits. The fixed length is usually the length of the longest codeword of the original VLC. For example, the

Huffman Code table corresponding to MTYPE VLC [1] code is shown in Table 1.

From table 1, the codeword 00001 represents MTYPE=4 and the code length is 5 bits. The longest codeword in the MTYPE VLC table is of 10 bits long, so we append $10-5=5$ bits to codeword 00001 to generate the corresponding VLC table ready for decoding. In other words, the binary numbers 0000100000 to 0000111111 will map the corresponding codewords to the same information of MTYPE=4. Then, we can read 10 bits from the bit stream and regard them as an index to search in the table. One source symbol will be decoded out at the expense of one table look-up operation.

Theoretically, the table look-up approach is the fastest way to decode the variable length code. But it has a vital drawback from the memory utilization point of view. The memory requirement of a code table grows exponentially with the maximum code length. In the above example, we need a table of size = 2^{10} bits. In other words, we need memory size 1024 to identify only 10 MTYPE informations. The memory utilization is only $10 / 1024 = 0.98$

%, in this case.

(ii). Tree-based table look-up

decoder:

One of the effective ways for the table size reduction is to reduce the maximum allowable code length of the VLC table. After observing the VLC tables in H.261, we found that all the associated Huffman tree are unbalanced and grows toward single side. As a result, we may partition a larger VLC table into some smaller size tables according to the codeword length [6]. Table 2 demonstrates this idea by using the same example given in Table 1.

By partitioning the original table into 3 smaller size tables, the memory requirement dropped from 1024 to $(16+8+8)=32$. Obviously, we have released the memory requirement up to more than 30 times of the original table look-up one. The only overheads we have to pay is to determine which table should be adopted to decode the 10-bit code. The decoding process is similar to traverse a search tree by comparing the value of a node. Consequently, we call this VLC decoding process to be the "Tree-based table look-up" decoding.

(iii). Superscalar VLC decoder

From the information theory point of view, the purpose of VLC is to use short codes to represent high frequency events. So, it is reasonable to say that the short length code should occur more frequently and consecutively. However, we found that the low memory utilization comes mainly from the short codes. To conquer this problem, the **superscalar VLC decoder** is proposed. We demonstrate this approach by using one entry of the TCOEFF VLC table[1] as an example. The codeword 110 represents the $(run,level) = (0,1)$. If the code 110 is decoded in 7-bit VLC table when Tree-based table look-up is applied. Then, the memory utilization for this code is $1/2^7 = 1/128$, and the low memory utilization results from the duplication of the indices for the code 010 from 0100001 to 0101111. Actually, the least significant 4 bits can be one of the following bit patterns:

2 bits: 10 (End of Block)

3 bits: 110 (run=0, level=1), 111
(run=0, level=-1)

4 bits: 0110 (run=1,
level=1), 0111 (run=0,
level=-1),

which are legal VLC symbols. So, we can embed these 5 legal codewords into wasted entries corresponding to the code 010 when building tree-based VLC tables. The memory utilization of this entry becomes $(1+5)/128 = 6/128$ - 6 times of the previous one. In addition, since we may decode two symbols at one table look up operation, this technique also speeds up the decoding process without increasing the table size. We call this "superscalar VLC decoder" since it is an application of the "superscalar" concept developed in the area of microprocessor technology. It is easy to extend the idea to decode three or more symbols at one table look-up operation if needed. This technique is suitable for decoding processes which decode two or more VLCs from the same VLC table. For instance, the transform coefficients (TCOEFF) of JPEG, the motion vector data (MVD) and the TCOEFF of MPEG-1, MPEG-2, H.261 and H.263 and all other Huffman-based VLC coders are belongs to this category.

B)Fast Inverse Discrete Cosine Transform (IDCT)

Although there are many new fast algorithms developed for IDCT, we

adopt the fast IDCT proposed by Chen, Smith and Flarick [2] for implementing our software decoder. It is an 8-point 1-D IDCT with 11 multiplications and 29 additions. To implement 8x8 block, there are many zero coefficients in the DCT domain. We can perform different 8-point 1-D IDCTs according to the occurrence of non-zero coefficients. In other words, we perform only the necessary computations to accomplish the IDCT. By using different IDCTs, we eliminate the unnecessary computation and hence speed up the IDCT module. This is an application of the "input pruning" technique. The reason why we choose the row-column approach to implement 2-D IDCT comes from its simple structure which makes it easier for us to achieve further speeding up for 2-D IDCT. Furthermore, we benefit from the characteristics of human visual system based on the fact that the human eyes are less sensitive to the variation of high frequencies. If there exists non-zero coefficients in the high band of the 8-point IDCT, we regard them as zeros. Although this will result in an imprecise IDCT, the blemish is almost invisible.

III. Real-time software H.261 encoder

To meet the real-time constraint of the audio-visual communication, the most important task is to build a real-time video encoder. There are three

important modules closely related to the performance of an H.261 encoder:

- (i). Motion estimation (ME),
- (ii). bit rate control,
- (iii). forward discrete cosine transform (FDCT).

We will discuss them in the following, respectively.

A) Fast Motion Estimation (ME)

ME is one of the most important issues in video encoding. So, there have been many fast algorithms developed for ME. After exhaustively experimenting, we make a trade-off between computation load and the quality degradation and choose the conjugate direction search (CDS) algorithm [4] to be the motion estimator of our software H.261 encoder.

Furthermore, we modify the CDS to achieve further speeding up for motion estimation based on the technique presented in [5]. In H.261, ME is performed on the 16x16 luminance block. If the minimum absolute error (MAE) is adopted to be the distortion measure, we need 256 subtractions, 255 branches and 255 additions to accomplish each block matching. But it is usually unnecessary to compute all the errors between two blocks if the distortion measure of the candidate block

exceeds the current minimum. Figure 1 shows the relationship between the cumulative error and the current minimum.

In other words, we can eliminate unqualified candidate block when the accumulation of absolute errors exceed the current minimum. We call this an "early jump out" motion estimation (EJOME). On the basis of the EJOME, we speed up the motion estimation by applying the more advanced EJO approach. Because the jump out index occurs only when the cumulative error curve exceeds the current minimum line which is a horizontal one, we may jump out earlier if the current minimum is also an monotone increasing curve which is lower than the horizontal line.

Figure 2 demonstrates this idea clearly and we obtain an EJO index that will eliminate the disqualified block more earlier than the original EJO approach. The advanced EJO curve can be obtained by linear interpolating the cumulative error curve and the current minimum line.

B) Bit rate control

The bit rate controller plays an important role in our software H.261 encoder. It monitors the fullness of the network buffer to adjust the output bit rate and takes the responsibility of picture quality

keeping. The controller is divided into the following three layers:

- (i).Picture bit allocator (layer-1),
- (ii).GOB enhancement layer (layer-2),
- (iii). Macroblock bit allocator (layer-3),

(i). Picture bit allocator

This bit allocator considers the fullness of network buffer as well as the total coded bit of the current picture so as to allocate the target bit rate for the next picture. According to the target bit rate, the bit allocator will select one appropriate “quantization-list” (Q-list) which is composed of different quantizers with different scales. This Q-list will be used for encoding the next picture. There are many Q-lists in our software H.261 encoder. Since the Q-lists are obtained during the developing phase through a training process with a lot of training video sequences, there is very little overheads for this allocation process.

(ii). GOB enhancement layer

For the video-phone or video-conference applications, the scene of the video sequence are often composed of “head and shoulder” pictures so that there is only one

moving head with some facial expressions and one still background. Furthermore, people usually focus their attention to the facial expressions. We try to allocate more bits to the GOBs where the head often located on and less bits to the GOBs where the still background belongs to. Figure-3 shows how the GOBs are enhanced for the two different picture sizes defined in H.261.

To allocate more bits to the target GOBs, we reduce the corresponding quantization scales in the Q-list which is defined by the picture layer bit allocator. On the contrary, the scales in the Q-list for the background GOBs are increased.

(iii).Macroblock bit allocator

The macroblock bit allocator receives the Q-list coming from the GOB enhancement layer. First, it determines which coding mode is used to code this macroblock. If the distortion between the current block and the corresponding prediction block is larger than some predefined threshold, the INTRA coding mode is performed. Otherwise, the INTER coding mode is adopted. For the INTER coding mode, the bit allocator will select a proper quantizer from the

Q-list according to the magnitude of the distortion. We classified the distortion into several classes. By classifying the macroblock into one of the distortion classes, we can choose a quantizer from the Q-list to quantize the macroblock DCT coefficients. Finally, the information of the total coded bits of the current picture will then feedback to the picture bit allocator for encoding the next picture. Figure 4 shows the relationships among the three bitrate control layers.

C) Fast Forward Discrete Cosine Transform (FDCT)

The software H.261 encoder utilizes the fast FDCT algorithm proposed by Loeffler, Ligtenberg and Moschytz [3]. Again we adopt the row-column approach to implement the 2-D 8x8 FDCT. There are 12 multiplications and 32 additions needed for each 8-point 1-D FDCT. FDCT is a heavy computation module of the H.261 encoding process. According to our statistic analyses of the distortion between the current block and corresponding prediction block, the more zeros the FDCT will output. As a result, we need only to compute some but not all DCT coefficients and impose the others to be zero. This is the so-called "output pruning" process of the FDCT. According to the magnitude of

the distortions, we perform one of the four FDCTs of different sizes: three output pruning FDCTs (as shown in Figure-5) and one full 8x8 FDCT.

IV. Current performance of the proposed software H.261 video codec

The proposed software H.261 video codec achieves real-time video compression and decompression on the Pentium-133 PC, under Win95 operating system. Tables 3 to 5 show the current performances of the realized software H.261 codec on Pentium-133. In which the network environment was simulated by memory read-and-write operations.

V. Conclusion

In this paper, we have presented an H.261 video codec with high speed and efficient resource utilization over personal computers by taking full advantages of the computation power of the general purpose CPU. The novel techniques proposed and adopted in our software video codec are also suitable for other hybrid DPCM/DCT codecs, such as MPEG-1, MPEG-2 and H.263. It is believed that a video codec built only by using software on general purpose computers has growing opportunity to go into people's daily life. With the aid of the progresses in data compression

techniques, the wide spread of fast network backbones and the advanced in CPU power, the software-based multimedia communication application such as video-phone and video-conference will become a reality in the near future.

[6] R. Hashemian, "Memory Efficient and High-Speed Search Huffman Coding", *IEEE Trans. On Communications*, Vol. 34, No.10, pp.2576-2581, Oct. 1995.

REFERENCE

- [1] ITU-T Recommendation H.261, "Video codec for audiovisual services at $p \times 64$ kbits/s", Mar,1993.
- [2] W.H. Chen, C.H. Smith, and S.C. Fralick, "A Fast Computation Algorithm for the Discrete Cosine Transform", *IEEE Trans. , On Communications*, Vol. COM-25, pp.1004-1009, 1977.
- [3] C. Loeffler, A Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", *Proc. Int'l. Conf. On Acoustics, Speech, and Signal Processing* 1989, pp.988-991.
- [4] Ram Srinivasan and K.R. Rao, "Predictive coding based on efficient motion estimation", *IEEE Trans, Commununications*, Vol. COM-33, No 8, Aug. 1985, ppp 888-896.
- [5] Ho-Chao Huang, Yi-Ping Huang, and Wen-Liang Hwang, "Adaptive early Jump-out Technique for Fast Motion Estimation in Video Coding," to appear in the *IEEE Inte'l Conf. On Pattern Recog.*,1996.

MTYPE	VLC	Extended Codeword	Fixed Length Codeword
3	1	<u>1</u> xxxxxxxx	512 ~ 1023
9	01	<u>01</u> xxxxxxxx	256 ~ 511
8	001	<u>001</u> xxxxxxxx	128 ~ 255
1	0001	<u>0001</u> xxxxxx	64 ~ 127
4	00001	<u>00001</u> xxxxx	32 ~ 63
10	000001	<u>000001</u> xxxx	16 ~ 31
2	0000001	<u>0000001</u> xxx	8 ~ 15
6	00000001	<u>00000001</u> xx	4 ~ 7
5	000000001	<u>000000001</u> x	2 ~ 3
7	0000000001	<u>0000000001</u>	1

Table 1 The MTYPE VLC

Table	MTYPE	VLC	Original Fixed Length Codeword Index	New Extended Codeword	New Fixed Length codeword Index
Table A	3	1	128 ~ 1023	<u>1</u> xx	4 ~ 7
	9	01		<u>01</u> x	2 ~ 3
	8	001		<u>001</u>	1
Table B	1	0001	16 ~ 127	<u>0001</u> xx	4 ~ 7
	4	00001		<u>00001</u> x	2 ~ 3
	10	000001		<u>000001</u>	1
Table C	2	0000001	1 ~ 15	<u>0000001</u> xxx	8 ~ 15
	6	00000001		<u>00000001</u> xx	4 ~ 7
	5	000000001		<u>000000001</u> x	2 ~ 3
	7	0000000001		<u>0000000001</u>	1

Table 2 The tree-based VLC Table

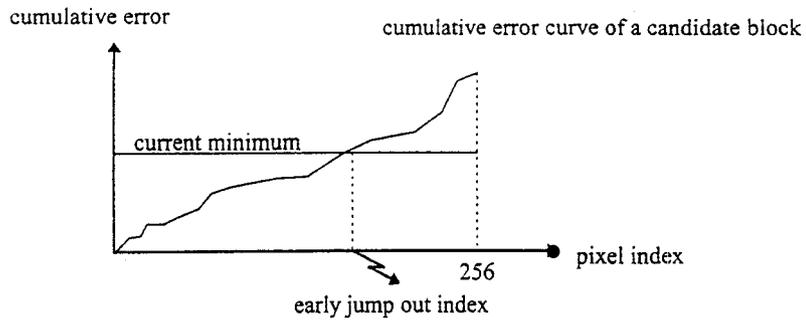


Figure - 1 The basic concept of the Early Jump

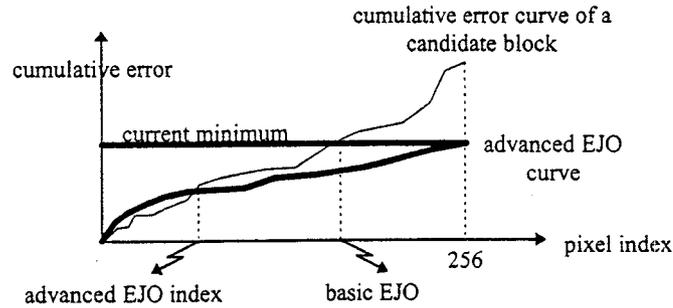


Figure - 2 The advanced EJO

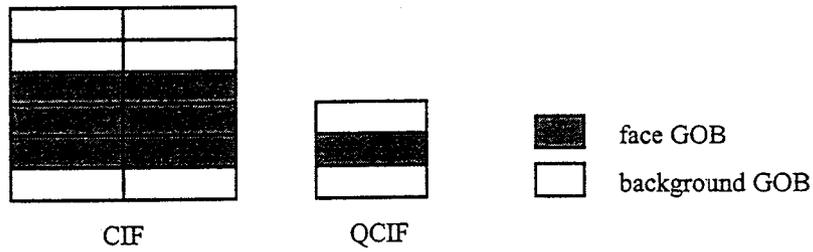


Figure-3 GOBs to be enhanced

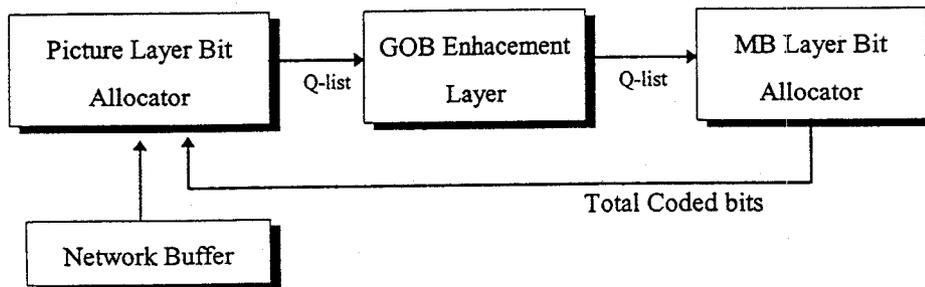


Figure-4- The relationship between the 3 layers bitrate

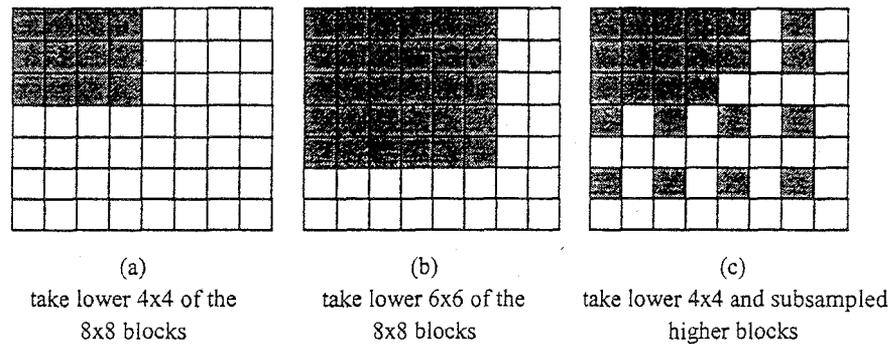


Figure-5 Output pruning FDCTs

Item format	Bit rate (p x 64 kbps)	Encoding Speed (fps)	Picture Quality PSNR (dB)		
			Y	Cb	Cr
CIF	p = 4	10 ~ 13	34.69	38.95	38.93
QCIF	p = 1	52 ~ 56	34.34	38.90	37.15

Table 3 Encoder only testing on Pentium-133

Item format	True Colors	256 Pseudo colors	Without Display
	(frames / sec)	(frames / sec)	(frames / sec)
CIF	21 ~ 24	55 ~ 57	131 ~ 136
QCIF	96 ~ 100	200 ~ 205	410 ~ 415

Table 4 Decoder only testing on Pentium-133

Item format	Codec Speed with true color display
	(Frames / sec)
CIF(p=4)	7 ~ 9
QCIF(p=1)	32 ~ 37

Table 5 Full codec test with simulated network enviroment on Pentium-133