A REALTIME SOFTWARE MPEG TRANSCODER USING A NOVEL MOTION VECTOR REUSE AND A SIMD OPTIMIZATION TECHNIQUES

Yuzo Senda and Hidenobu Harasaki

C&C Media Research Laboratories, NEC Corporation
1-1, Miyazaki 4-chome, Miyamae-ku, Kawasaki 216-8555 JAPAN
Email: yuzo@ccm.CL.nec.co.jp

ABSTRACT

A realtime software MPEG transcoder has been developed. A novel motion vector reuse and a SIMD optimization techniques are introduced to accelerate the transcoder without any quality degradation. Mean absolute error approximation criteria are employed in the reuse technique to refine scaled motion vectors. The developed transcoder on Pentium II 266MHz runs 2.5 times as fast as realtime, when scaling an MPEG-1 bitstream to half size.

1. INTRODUCTION

Video distribution and retrieval services over the Internet and intranets are becoming popular. As these networks do not guarantee effective transfer rate, video server must have an ability to adapt the transfer rate change. A scalable encoding method seems to be a promising solution, however, the method cannot be used easily because of its complexity and peculiarity. MPEG-1 [1] is widely used instead. There are a lot of hardware decoders available in the market. Moreover, operating systems bundle software decoders. Though MPEG-1 does not support scalability, transfer rate adaptation is achievable by providing multiple bitrate bitstreams. If a server could serve multirate bitstreams, each client can select and play a bitstream suitable upon network condition and its decoding power.

Cost vs. quality tradeoff exists in generating multirate bitstreams. When placing a hardware encoder for each bitstreams, the cost increases proportionally to the number of supported bitstreams, while the picture quality of each bitstream maintains optimum. A single processor cannot generate all bitstreams by software in realtime. To archive cost effective and high quality multirate encoder, we employ a cascade combination of hardware encoder and software multirate transcoder. This is because the hardware maximizes the picture quality by its large area motion estimation and software transcoder helps to minimize the total encoder system cost. In some cases, realtime transcoding is not required. For example, in a video on demand (VOD) service, an offline transcoder may be used to generate subordinate bitstreams. However, a realtime transcoder is necessary for providing a live broadcasting service.

Recently, general purpose processors employ SIMD instructions for multimedia processing. Utilizing the instructions, a realtime software MPEG-2 decoder has been developed [2]. By applying the software implementation technique employed in the decoder and a novel motion vector reuse technique, we have developed a realtime software MPEG-1 transcoder. This paper explains the techniques and evaluates the performance of the developed transcoder.

2. PROPOSED TRANSCODING METHOD

Figure 1 shows a block diagram of a video broadcasting system. A hardware encoder first generates the reference bitstream from an input video signal. A software multirate transcoder then decodes the reference

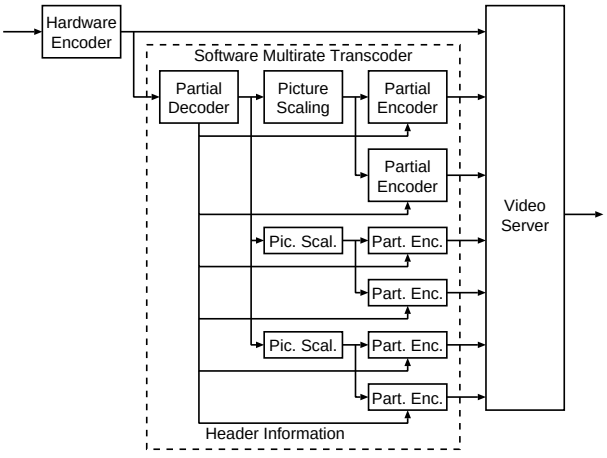

Figure 1: A video broadcasting system employing a software multirate transcoder

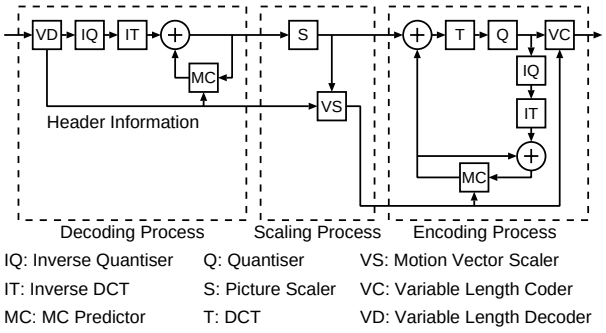

Figure 2: The block diagram of a video transcoder

bitstream, generates a down-scaled picture, and re-encodes it with a lower bitrate. A video server finally broadcasts the bitstreams to a network.

A DCT domain processing is known as a fast transcoding method [3]. The method statistically reduces the number of operations. Since the method needs two bytes to represent a pel, the necessary memory access bandwidth for a motion compensated predictor (MC) becomes twice as much. In video processing with a general purpose processor, the memory bandwidth is a bottleneck [4, 5]. So the DCT domain processing is not suitable for this application. An ordinary spatial domain processing is adopted in our transcoder.

In transcoding without spatial scaling, a decoding and an encoding processes can share the MC [6, 7]. This paper, however, proposes cascade processes of a decoding, a scaling and an encoding shown in Figure 2. The difference in data path from a simple concatenation of the processes is elimination of frame reordering. The elimination reduces both necessary memory size and processing delay.

2.1. Reuse of Motion Vectors

Motion estimation can be significantly simplified when utilizing the motion vectors contained in the reference bitstream. The usage of simply scaled motion vectors for transcoding with spatial scaling results in serious quality degradation [8]. To avoid it, our transcoder evaluates scaled motion vectors and their neighbors, and then selects the best ones. Figure 3 shows an example. In 1/2 scaling, four macroblocks in the decoded picture are corresponding to a macroblock in the encoding picture. The transcoder uses 1/2 scaled motion vectors as candidates.

It is easy to compute the mean absolute error (MAE) of an integer motion vector by direct block matching. However, the MAE computation for a non-integer motion vector costs several times as much as for an integer one because of half-pel interpolation. To reduce the cost, approximate criteria [9, 10] are introduced. With the criteria, the MAEs of half-pel motion vectors as well as that of interpolative prediction can be evaluated by following simple equations.

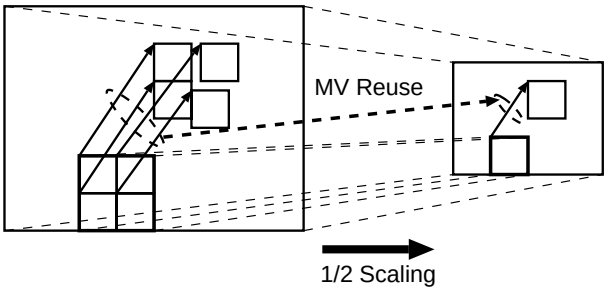

Figure 3: Reuse of motion vectors

$$\tilde{E}_{i+0.5,j} = \frac{7}{16}(E_{i,j} + E_{i+1,j}) , \quad (1)$$

$$\tilde{E}_{i,j+0.5} = \frac{7}{16}(E_{i,j} + E_{i,j+1}) , \quad (2)$$

$$\tilde{E}_{i+0.5,j+0.5} = \frac{13}{64}(E_{i,j} + E_{i+1,j} + E_{i,j+1} + E_{i+1,j+1}) , \quad (3)$$

$$\tilde{E}_{intp} = \frac{7}{16}(E_{forw} + E_{backw}) , \quad (4)$$

where $E_{i,j}$ is the MAE of an integer motion vector (i, j), $\tilde{E}$ is the estimated MAE, E_{forw} and E_{backw} are the MAEs of forward and backward prediction, $\tilde{E}_{intp}$ is the estimated MAE of interpolative prediction. As a result, predicted picture generation for evaluation is eliminated.

To reduce the number of candidate motion vectors to be evaluated, following categories are introduced.

(a) When both the horizontal and the vertical components of a scaled motion vector are integer, only the MAE of the vector is computed.

(b) When either of the components is non-integer, the MAEs of neighboring two integer motion vectors are computed. The MAE of the half-pel motion vector between the two is then estimated with the criteria.

(c) When both of the components are non-integer, the MAEs of neighboring four integer motion vectors are computed. The MAEs of the five half-pel motion vectors surrounded by the four are then estimated with the criteria.

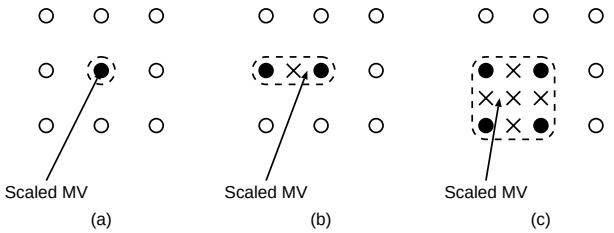

Figure 4: Three categories of scaled motion vectors and their search ranges

Figure 4 shows the three categories and their search ranges. Block matching is carried out only for solid-circle markers which mean integer candidate motion vectors. The approximate criteria are applied to cross markers which mean non-integer candidate ones. The search range depends on a scaled motion vector.

2.2. Implementation

Our transcoder also employs the software implementation techniques introduced to a realtime software MPEG-2 decoder [2]. One is a SIMD optimization technique. DCT and IDCT takes a large number of computations, so they were considered as the bottleneck of multimedia processing. Since general purpose processors have employed SIMD instructions for multimedia processing, DCT and IDCT are no longer intensive for the processors. For example, four coefficients can be processed in parallel by using Intel's MMX technology [11, 12]. Processing time is then reduced to one quarter at the best. The MC accesses reference pictures by eight pels, and generates half-pel and interpolative predicted pictures by four pels. Since MPEG-1 generally handles one quarter resolution of MPEG-2, all data can be located on the L2 cache of the processor. Therefore, the MPEG-1 transcoder does not pay a cache miss penalty, which is still a large problem in the MPEG-2 decoder.

Another technique is a concurrent variable length decoder (VLD) [13]. It is not easy to decode a variable length code in parallel because the start position of a variable length code depends on its previous code [14]. In other words, VLD is a sequential process. To accelerate VLD, two symbol concurrent decoding is introduced. The current decoding is applicable only for two successive short codes. Our VLD runs twice as fast as a conventional one at the maximum.

3. PERFORMANCE

The implemented techniques are evaluated in this section. A video CD (1.15Mbps, 352 pels, 240 lines, 29.97 Hz) is used as the reference bitstream. The transcoder runs on Intel Pentium II 266 MHz, and generates a spatially half scaled bitstream (384 Kbps). Figure 5 shows the average processing time of a one-second piece of the bitstream. The accelerating ratios are 4.0 with the motion vector reuse technique, 4.9 with the software implementation technique, and 14.0 with both techniques. Compared with the period of a piece, the transcoder runs 2.5 times as fast as realtime. Figure 6 shows the detail of the processing time. In general, simplification leads degradation, however, the motion vector reuse technique improves +0.03 dB in SNR. Therefore, the technique does not cause any quality degradation.

Even when the average processing time is less than one second, processing time for each frame depends on the bitstream. The above result does not guarantee realtime transcoding with a frame period delay. Figure 7 shows the distribution of processing time measured with 0.01-second accuracy. The transcoder takes 0.39 seconds in average, however, 0.5 seconds in the worst case. To guarantee realtime software transcoding in a minimum processing delay, emergency processing is necessary to avoid an overrun. However, emergency processing usually causes significant quality degradation. When assuming processing time histogram as a normal distribution, overtime probability can theoretically be defined by the average μ and the standard deviation σ of processing time. In processing the video CD, μ=0.392 and σ=0.040. To avoid degradation oc-

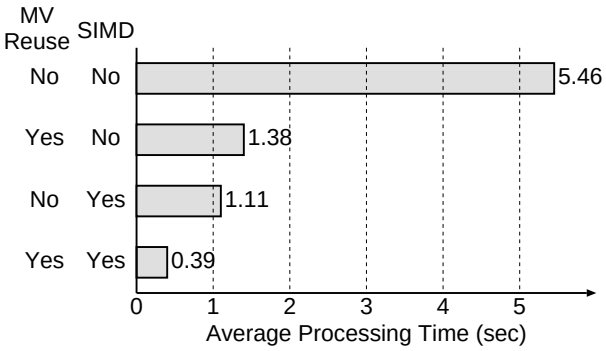

Figure 5: Average processing time in video transcoding with or without the proposed techniques

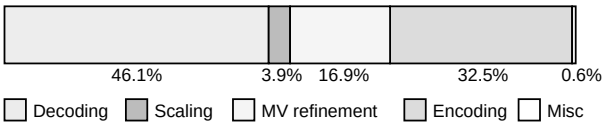

Figure 6: Percentage of processing time in video transcoding with the proposed techniques

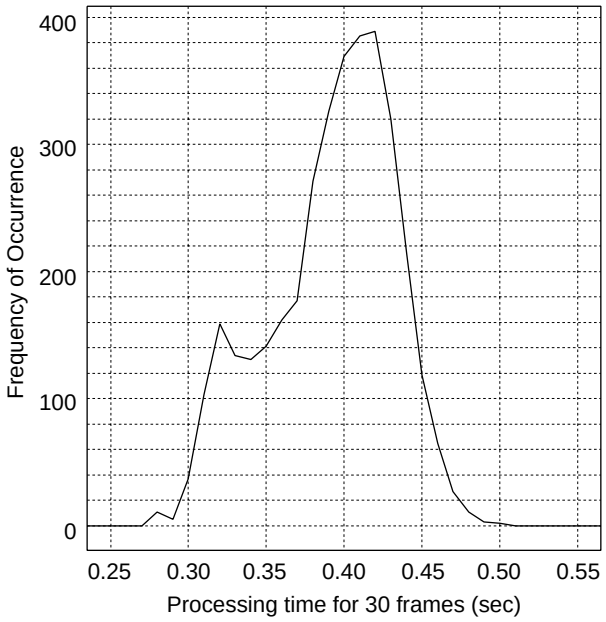

Figure 7: Distribution of processing time for 30 frames

currence, $\mu + 3\sigma$ second processor time is allocated for transcoding of each piece. Another approach to ensure realtime processing is to enlarge buffers to absorb fluctuation of processing time. In other word, buffering delay is introduced to guarantee the realtime processing. Assuming processing time is a first order autoregressive process, the average μ_n and the standard deviation σ_n of processing time of n pieces are

$$\mu_n = n\mu \tag{5}$$

$$\sigma_n = \sigma\sqrt{n + \sum_{i=1}^{n-1} 2(n-i)\,\rho^i} \tag{6}$$

where ρ is an autocorrelation factor, 0.625 in the video CD. When n=3 as an example, σ_n is 2.506σ, not 3σ. Consequently, the buffer enlargement has an effect to reduce σ.

4. CONCLUSION

A realtime software MPEG transcoder has been developed. Introduced motion vector reuse and SIMD optimization techniques accelerates the transcoder 14 times as fast as a straight forward implementation. Both techniques do not introduce any quality degradation. When scaling an MPEG-1 bitstream to half size, the transcoder on Pentium II 266MHz runs 2.5 times as fast as realtime. The transcoder has been employed in our multimedia-on-demand system namely HYPERMS-Lite.

5. REFERENCES

[1] ISO-IEC/JTC1/SC29/WG11, “Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s”, IS11172 (1992)

[2] M. Ikekawa, D. Ishii, E. Murata, K. Numata, Y. Takamizawa, M. Tanaka, “A Real-time Software MPEG-2 Decoder for Multimedia PCs”, ICCE-97, WAM 1.1, pp. 2-3 (1997)

[3] S.-F. Chang, D. G. Messerschmitt, “Manipulation and Compositing of MC-DCT Compressed Video”, IEEE JSAC, Vol. 13, No. 1, pp. 1-11 (Jan. 1995)

[4] D. F. Zucker, M. J. Flynn, R. B. Lee, “Improving Performance for Software MPEG Players”, COMPCON’96, pp. 327-332 (Feb. 1996)

[5] K. Nadehara, H. Lieske, I. Kuroda, “Software MPEG-2 Video Decoder on a 200-MHz, Low-Power Multimedia Microprocessor”, ICASSP-98, Vol. 5, pp. 3141-3144 (May 1998)

[6] D. G. Morrison, M. E. Nilsson, M. Ghanbari, “Reduction of the bit-rate of compressed video while in its coding form”, Packet Video, D17.1 (Sep. 1994)

[7] G. Keensman, R. Hellinghuizen, F. Hoeksema, G. Heideman, “Transcoding of MPEG bitstreams”, Signal Processing: Image Communication, Vol. 8, No. 6, pp. 481-500 (Sep. 1996)

[8] N. Bjork, C. Christopoulos, “Transcoder Architecture for Video Coding”, IEEE Trans. Consumer Electronics, Vol. 44, No. 1, pp. 88-98 (Feb. 1998)

[9] Y. Senda, H. Harasaki, M. Yano, “A Simplified Motion Estimation Using an Approximation for the Real-Time MPEG-2 Encoder”, ICASSP-95, Vol. 4, pp. 2273-2276 (May 1995)

[10] Y. Senda, H. Harasaki, M. Yano, “Theoretical Background and Improvement of a Simplified Half-Pel Motion Estimation,” ICIP-96, Vol.3, pp.263-266 (Sep. 1996)

[11] Intel, “Intel Architecture MMX(TM) Technology, Programmer’s Reference Manual”, Order No. 243007-001 (Mar. 1996)

[12] E. Murata, M. Ikekawa, I. Kuroda, “Fast 2D IDCT Implementation with Multimedia Instructions for a Software MPEG2 Decoder”, ICASSP-98, Vol. 5, pp. 3105-3108 (May 1998)

[13] D. Ishii, M. Ikekawa, I. Kuroda, “Parallel Variable Length Decoding with Inverse Quantization for Software MPEG-2 Decoders”, SiPS-97, pp. 500-509 (Nov. 1997)

[14] E. Holmann, A. Yamada, T. Yoshida, S. Uramoto, “Real-Time MPEG-2 Software Decoding with a Dual-Issue RISC Processor”, VLSI Signal Processing IX, pp. 105-114, (Oct. 1996)