Algorithm-level Optimization for Real-time H.264/AVC Encoder

Seung-Hwan Kim and Yo-Sung Ho

Department of Information and Communications
Gwangju Institute of Science and Technology (GIST)
1 Oryong-Dong, Buk-Gu, Gwangju, 500-712, Korea
Phone: (+82) 062-970-2258 Fax: (+82) 062-970-3164 E-mail: {kshkim, hoyo}@gist.ac.kr

Keywords: H.264/AVC, Fast mode decision, Algorithm optimization.

Abstract - The H.264 standard achieves higher compression efficiency than previous video coding standards with the rate-distortion optimized (RDO) method for mode decision. The outstanding coding performance of H.264, however, comes at the cost of significantly increased complexity. In this paper, we present algorithm-level optimization methods: input parameter selection, fast inter-mode decision, and efficient combination of motion estimation and mode decision. Simulation results show that our optimized H.264 encoder achieves real-time encoding for Video Graphics Array (VGA) and eXtended Graphics Array (XGA) on a commercial personal computer without introducing serious quality degradations.

1. INTRODUCTION

The H.264 standard was developed through the Joint Video Team (JVT) from the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group (MPEG) standardization. H.264 is one of the most exciting developments in video coding [1].

H.264 improves the rate distortion performance by exploiting advanced video coding technologies, such as variable block size motion estimation, multiple reference prediction, spatial prediction in intra coding, context based variable length coding (CAVLC) and context-based adaptive binary arithmetic coding (CABAC). Test results of H.264/AVC show that it significantly outperforms existing video coding standards in both peak signal-to-noise ratio (PSNR) and subjective visual quality [2].

To achieve high coding efficiency, we use the Lagrangian rate-distortion optimization (RDO) technique in H.264/AVC and decide the best coding mode for each macroblock. In order to choose the best coding mode, we calculate the rate-distortion (RD) cost of every possible mode and select the mode of the minimum RD cost. This process is repeatedly carried out for all the possible modes for every macroblock. This type of brute force-searching algorithm is far more demanding the computational complexity than any other video coding algorithm. However, the outstanding coding performance of H.264 comes with the cost of significantly higher complexity, making it too difficult to be applied widely. Therefore, computational complexity reduction is important to perform real-time encoding software on a personal computer.

In this paper, we present our algorithm-level optimization including input parameter selection and fast inter mode decision, and efficient combining technique between motion estimation and mode decision. We also apply code-level optimization techniques of memory rearrangement and single-instruction-multiple-data (SIMD) instruction sets.

2. COMPLEXITY ANALYSYS OF H.264/AVC

Fig. 1 shows the high-level execution-time analysis of the H.264 JM9.5 reference encoder [3]. In the experiment, Intel® VTune™ Performance Analyzer20 [4] is used as the profiling tool to evaluate the software performance and obtain the complexity profile of the reference and optimized encoders. We have also tested the FOREMAN sequence (300 frames, CIF format, IPPPPP… frame structure) on an Intel Pentium-4 3.0GHz PC with 512 MB memory under the Microsoft Windows XP.

As shown in Fig. 1, the most time-consuming modules of H.264 encoder are mode decision and motion estimation including interpolation. Therefore, by optimizing these modules in the top priority, we develop an efficient H.264 encoder that is capable of working in real-time with high coding efficiency.

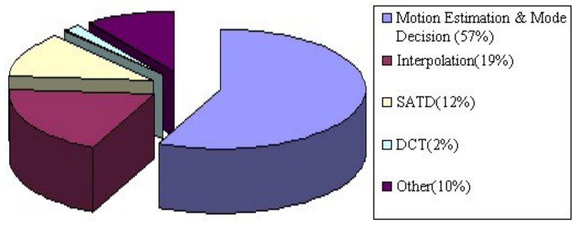


Fig. 1. Complexity analysis of H.264/AVC

3. ALGORITHM-LEVEL OPTIMIZATION

In this Section, we present three efficient fast encoding techniques for real-time encoder. Firstly, we introduce an efficient input parameter selection method by comparing the RD performance and complexity according to activation of each input parameter. Secondly, we propose an efficient fast inter mode decision (EFMD) method using the early skip mode decision and an efficient mode comparison method. Based on EFMD, we propose an efficient combined motion estimation and mode decision (ECMEMD). In ECMEMD, we change the general structure of motion estimation process in H.264 reference software and propose a new fast mode decision method.

3.1 Selection of each coding option

H.264 reference software provides many coding options to achieve the higher coding efficiency. In order to design the efficient real-time encoding software, we do not need to use all the coding options but select several efficient options. In order to estimate the efficiency of each coding option, removing each option one by one, we compare the coding efficiency and encoding time.

A group of experiments were carried out on the six CIF (352×288) sequences: FOREMAN, COASTGUARD, CARPHONE, CONTAINER, FOOTBALL, and AKIYO. In order to evaluate the encoding time of each option, the following calculation of time difference (ΔTime) is defined by

$$\Delta Time = \frac{T_{\mathrm{Re}moved\_Option} - T_{Full\_Option}}{T_{Full\_Option}} \times 100(\%) \qquad (1)$$

where T_{Full_Option} represents the total encoding time for using all options listed in Table 1. PSNR and bit-rate differences are calculated according to the numerical averages between the RD-curves derived from full option and the removed option, respectively. In Table 1, we represent the results for difference in PSNR and bitrate for each option. Using Table 1, we can estimate the efficiency of each coding option. From Table 1, we can observed that intra prediction mode in the inter frame

Removed Option	ΔPSNR (dB)	ΔBits (%)	ΔTime
Intra 16×16, 4×4	- 0.07	1.23	- 53.48
Sub-pixel ME	- 0.39	35.7	- 16.28
Hadamard	- 0.05	-0.2	- 5.46
Inter 16×8, 8×16	- 0.03	4.66	- 9.84
Inter 8×8	- 0.03	0.94	- 1.1
Inter 4×8, 8×4, 4×4	- 0.08	2.46	- 13.64

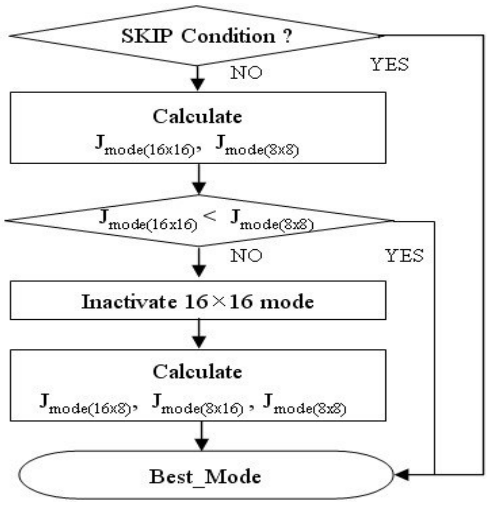
Table 1. Differences in PSNR and bitrate between full option and each removed option (QP=26, 28, 30, 36)

3.2 Fast inter mode decision

We propose an efficient fast mode decision algorithm (EFIMD) using the early skip mode decision and an efficient mode comparison method. The flowchart of the EFIMD is depicted in Fig. 2.

The SKIP mode refers to the 16x16 mode where neither motion vector nor residual information is encoded. Hence, it has the lowest complexity in the mode decision process since no motion search is required.

The proposed EFMD can be mainly divided into three steps. In the first step, we find the motion vector of the SKIP mode and calculate RD cost to determine whether the best macroblock mode is the SKIP mode or not. If one of the previously encoded neighboring blocks is determined to be the SKIP mode and SKIP cost of the corresponding block is less than the given threshold, we determine the best mode of corresponding block is the SKIP mode. In the second step, we calculate RD costs for 16×16 and 8×8 modes. If $J_{mode(16\times16)}$ is less than $J_{mode(16\times16)}$, the best mode is determined as 16×16 mode. Otherwise, we go to the third step and find the best mode among 16×8, 8×16 and 8×8 modes.



SKIP Condition ?
NO
YES
Calculate
$J_{mode(16x16)}$, $J_{mode(8x8)}$
$J_{mode(16x16)} < J_{mode(8x8)}$
NO
YES
Inactivate 16×16 mode
Calculate
$J_{mode(16x8)}$, $J_{mode(8x16)}$, $J_{mode(8x8)}$
Best_Mode

Fig. 2. Motion estimation structure in reference software

3.3 Combined mode decision and motion estimation

In this section, we jointly optimize the mode decision and motion estimation processes. The motion estimation process can be divided into two parts: Integer-Pel and Fractional-Pel motion estimation. In particular, Fractional–Pel (half-pel and quarter-pel) ME requires a large amount of complexity because it requires an interpolation process separately from motion search. For example, half-pixels and quarter-pixels are interpolated by applying the 6-tap FIR filter and bilinear filter, respectively. Hence, it requires much amount of computational complexity to find motion vector from integer pixel accuracy to quarter pixel accuracy at a time.

Therefore, we decompose the motion estimation process into three independent stages, such as integer pixel, half pixel, and quarter pixel estimation. Combining these three motion estimation stages with a fast mode decision method including early SKIP mode decision, we propose a new efficient combined motion estimation and mode decision (ECMEMD) scheme. Fig. 3 shows the flowchart of the proposed fast mode decision algorithm.

In Fig. 3, $J_{mode\_I\,(M)}$ and $J_{mode\_H\,(M)}$ represent RD costs calculated by using integer-pel motion vector and half-pel motion vector in a given mode M, respectively. The proposed ECMEMD is divided into four steps. In the first step, we determine the SKIP condition as the same way in EFMD in Fig.2. In the second stage, we find integer-pel motion vector and calculate the RD costs from the motion vectors. If $J_{mode\_I(16\times16)}$ is less than $J_{mode\_I(8\times8)}$, the best mode is determined as the 16×16 mode and we further fulfill the motion estimation process to find sub-pixel (half-pixel and quarter-pixel) motion vector. Otherwise, go to the next step and we select the best mode from the $J_{mode\_H(16\times8)}$, $J_{mode\_H(8\times16)}$, and $J_{mode\_H(8\times8)}$.

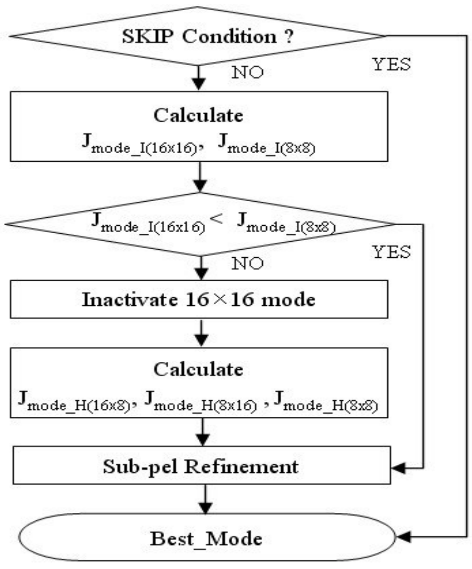

SKIP Condition ?
NO
YES
Calculate
$J_{mode\_I(16x16)}$, $J_{mode\_I(8x8)}$
$J_{mode\_I(16x16)} < J_{mode\_I(8x8)}$
NO
YES
Inactivate 16×16 mode
Calculate
$J_{mode\_H(16x8)}$, $J_{mode\_H(8x16)}$, $J_{mode\_H(8x8)}$
Sub-pel Refinement
Best_Mode

Fig. 3. Flow chart of the proposed ECMEMD

In order to optimize C code, we also apply several SIMD instruction sets used in [6]. Sum of absolute differences (SAD) and sum of squared differences (SSD), Integer transform, and inverse integer transform are implemented using a set of SIMD matrix operations. For detail information of Code-level optimization techniques using SIMD instruction sets, we recommend to refer [6].

4. EXPERIMENTAL RESULTS

We have performed our optimized encoder on a Pentium-4 3.0GHz personal computer equipped with 512 MB main memory and the Windows XP OS. The runtime complexity is profiled using the Intel® VTune 20 performance analyzer. Six CIF (352×288) sequences of 300 frames, including FOREMAN, COASTGUARD, CARPHONE, CONTAINER, FOOTBALL, and AKIYO are tested. Table 2 shows the simulation conditions.

Reference Software	**JM 9.5**
GOP Structure	**IPPPPP…**
Frame Rate	**30**
Entropy Coding	**CAVLC**
Quantization Parameter	**26, 28, 30, and 36**
Search Range	**16**
Number of Reference Frame	**1**
RD Optimization	**on**

Table 2. Simulation conditions

In our experiments, four different levels of optimized H.264 encoders (JM, JM_sel, EFMD, and ECMEMD) are compared with runtime and coding performance. "JM" encoder represents the same as original JM 9.5 and all input options indicated in Table 2 are used. "JM_sel" encoder is the same as JM but disable several input options, such as Hadamard, Inter (4×8, 8×4, 4×4), intra prediction in the inter macroblock. In EFMD and ECMEMD, input parameters are set as the same as JM_sel and code-level optimization is also added.

In Table 3 and Table 4, we represent the runtime comparison for four different H.264 encoders. According to simulation results, the encoding speed of the JM is only about 0.47 CIF fps. With the help of controlling several input parameter, JM_sel achieves a speed-up factor of 3.5, leading to the speed of about 1.64 CIF fps. With the proposed fast inter mode decision and code-level optimization methods, EFMD and ECMEMD are further enhanced to achieve up to 75 CIF and 161 CIF fps, respectively.

Sequence	**JM**	**JM_sel**	**EFMD**	**ECMEMD**
FOREMAN	**645.92**	**192.57**	**4.031**	**1.875**
COASTGUARD	**769.05**	**210.87**	**5.008**	**2.266**
CARPHONE	**573.86**	**176.41**	**3.789**	**1.684**
CONTAINER	**555.01**	**174.63**	**3.503**	**1.607**
FOOTBALL	**820.70**	**223.27**	**5.350**	**2.500**
AKIYO	**473.73**	**124.51**	**2.692**	**1.235**
Average	**639.71**	**183.71**	**4.062**	**1.861**

Table 3. Runtime comparison (QP=28)

Sequence	**JM**	**JM_sel**	**EFMD**	**ECMEMD**
FOREMAN	**570.52**	**188.87**	**3.711**	**1.734**
COASTGUARD	**679.56**	**210.87**	**4.523**	**2.094**
CARPHONE	**505.96**	**171.53**	**3.403**	**1.547**
CONTAINER	**482.85**	**170.21**	**3.159**	**1.483**
FOOTBALL	**705.20**	**218.52**	**5.033**	**2.344**
AKIYO	**388.46**	**121.36**	**2.631**	**1.205**
Average	**555.43**	**180.06**	**3.743**	**1.735**

Table 4. Runtime comparison (QP=30)

In Table 5 and Table 6, we represent PSNR and bitrate difference from "JM." For sequences without intensive motions, the optimized encoder yields very close coding performance as compared to the non-optimized JM encoder. For sequences with relatively large motions, the optimized encoder introduces about 0.6 dB degradation for the same bit rate. From Simulation results, we confirm that our proposed optimization H.264 encoders (EFMD and ECMEMD) can achieve a significant speed-up without introducing serious quality degradation.

Sequence	**JM_sel**	**EFMD**	**ECMEMD**
FOREMAN	**-0.18dB**	**-0.31dB**	**-0.54dB**
COASTGUARD	**-0.15dB**	**-0.28dB**	**-0.37dB**
CARPHONE	**-0.14dB**	**-0.26dB**	**-0.49dB**
CONTAINER	**-0.12dB**	**-0.25dB**	**-0.30dB**
FOOTBALL	**-0.19dB**	**-0.34dB**	**-0.65dB**
AKIYO	**-0.09dB**	**-0.12dB**	**-0.15dB**
Average	**-0.15dB**	**-0.26dB**	**-0.42dB**

Table 5. PSNR Difference from "JM" (QP=28)

Sequence	JM_sel	EFMD	ECMEMD
FOREMAN	1.4%	2.1%	4.2%
COASTGUARD	2.1%	3.2%	3.7%
CARPHONE	1.3%	2.3%	5.9%
CONTAINER	1.2%	1.3%	3.1%
FOOTBALL	1.7%	1.9%	4.3%
AKIYO	0.3%	0.6%	1.1%
Average	1.33%	1.9%	3.25%

Table 6. Bitrate Difference from "JM" (QP=28)

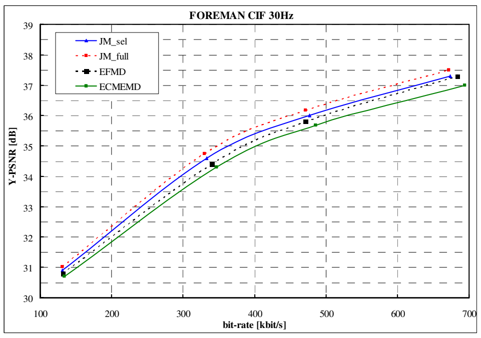


Fig. 7. Comparison of RD curves

In Table 7, we have tested a subset of the test sequences in the MPEG Call for Proposals on multiview video coding and the results of the number of encoding frames per second are shown. Four VGA (640×480) sequences of 250 frames, including Ballroom, Race1, Flamenco2, and Exit and two XGA (1024×768) sequences of 250 frames, including Uli and Breakdancers are tested, respectively.

Sequence (Resolution)	EFMD (fps)	ECMEMD (fps)
Ballroom (640×480)	27.15	62.01
Race1 (640×480)	27.84	62.54
Flamenco2 (640×480)	25.75	58.72
exit (640×480)	31.05	71.43
Uli (1024×768)	10.05	21.22
Breakdancers (1024×768)	11.14	24.52

Table 7. The number of encoding frames per second (QP=30)

5. CONCLUSIONS

In this paper, we have proposed an efficient algorithm-level optimization technique for real-time H.264 software encoder. In order to optimize the H.264 reference software, we propose a fast mode decision algorithm including early SKIP mode decision and combined motion estimation and mode decision. From simulation results, we verify that our optimized H.264 encoder can compress the video sequences of eXtended Graphics Array (XGA: 1024×768) format at the speed of 24fps without introducing serious quality degradations.

ACKNOWLEDGEMENTS

This work was supported in part by MIC through RBRC at GIST and in part by MOE through the BK21 project.

REFERENCES

[1] Joint Video Team of ITU-T and ISO/IEC JTC 1, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), Doc. JVT-G050, March 2003.
[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. CSVT., vol.13, no. 7, pp. 560-576, July 2003.
[3] JVT reference software version 9.5, available online at: http://iphome.hhi.de/suehring/tml/download/old_jm/.
[4] Intel Corp., "Intel® VTune™ Performance Analyzer."
[5] Y.L. Lai, Y.Y. Tseng, C.W. Lin, Z. Zhou, and M.T. Sun, "H.264 Encoder Speed-Up via Joint Algorithm/Code-Level Optimization," in Proc. Visual Communication and Image Processing (VCIP), July 2005.
[6] Available online at: http://forum.doom9.org/shothread.php?t=89979.