A H.264 BASIC-UNIT LEVEL RATE CONTROL ALGORITHM FACILITATING HARDWARE REALIZATION

Ping-Tsung Wu[1], Tzu-Chun Chang[1], Ching-Lung Su[2,3], and Jiun-In Guo[1]

[1]Department of Computer Science and Information Engineering, National Chung Cheng University, Chia-Yi 621. Taiwan, R.O.C.
[2]Department of Electronics Engineering, National Yunlin University of Science Technology, Yun-Lin, 640, Taiwan, R.O.C.
[3]SoC Technology Center, Industrial Technology Research Institute, Hsinchu, Taiwan, ROC
[1]Emails: {wupt, ctc95m, jiguo}@cs.ccu.edu.tw [2]Email: kevinsu@yubtech.edu.tw

ABSTRACT

Rate Control plays an important role for video coding especially in video streaming applications with bandwidth constraints. The inherent sequential processing in H.264 basic unit (BU) level rate control algorithm makes it hard to be realized in a pipelined H.264 hardware encoder without increasing the processing latency. In this paper we propose a new H.264 BU-level rate control algorithm facilitating hardware realization. The proposed algorithm breaks down the sequential processing dependence in the original rate control algorithm in JM and reduces 28% for QCIF, 66% for CIF, 87% for D1 of hardware cycles while maintaining good video quality. Simulation results shows that the proposed algorithm reduces MAD's memory buffer size to be N_{unit} * 14bits, which amounts to 26% for QCIF, 59% for CIF, 83% for D1 reduction as compared to JM rate control. Moreover, the proposed algorithm possesses high feasibility for hardware realization.

Index Terms— H.264, Rate Control, BU Level

1. INTRODUCTION

In recent years, end-to-end video applications like video phones require stable video quality during communication between two parties. To achieve stable video quality, Rate Control (RC) plays an important role to maintain a good video quality under the constraint of finite varying bandwidth during video transmission. RC algorithms can dynamically adjust the Quantization Parameter (QP) values in order to achieve the specified target bitrates during video encoding. If you want to have high bitrates, you have to choose small QP values. If we disable the RC operations of a video encoder system, it will keep the fixed QP values and the output bitrates are not constant anymore. On the other hand, if we enable the RC operations, we will have constant bitrates in encoding video. Figure 1 shows the real compressed bits for the "Silence" sequence with RC disabling/enabling. The parameters for the RC operations are GOP=30, QP=24 and target bitrate=256kbps.

In the literatures, there have been many RC algorithms proposed to improve the quality for H.264 video encoding. Most of them are derived based on quadratic Rate Distortion (RD) model. The reference software (JM) of H.264 video encoder adopts the linear model to predict the Mean Absolute Difference (MAD) value for solving rate control and Rate Distortion Optimization (RDO) problems [1-2]. However, it suffers from the QP dilemma problem. In order to resolve this problem, all the exiting H.264 RC algorithms estimate QP or MAD values by using the information of previous MBs [4].

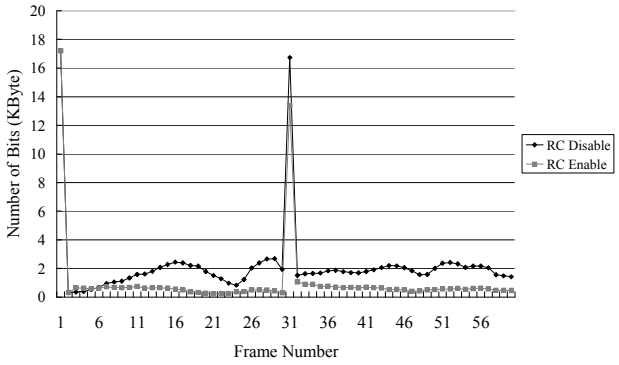

Figure 1. Real bits for "Silence" sequence with disabling/enabling RC operations

The RC algorithm of H.264 reference software JM is divided into three levels: i.e. Group of Picture (GOP) level, Frame level and Basic Unit (BU) level [1]. Among them, BU-level RC algorithm owns better performance in allocating the data bits than the frame-level RC for video encoding. There have been some papers proposed for H.264 RC using different levels [3-6]. All of these RC algorithms are implemented in software, which makes them hard to be realized in a pipelined H.264 video encoder design without increasing latency induced by the sequential RC processing requirement. In addition, the H.264 RC algorithm is much more complex than that of MPEG-4, which also induces high hardware cost in the hardware realization.

In order to solve the problems induced by the RC sequential processing requirement and high hardware complexity, we propose a new H.264 BU-level RC algorithm facilitating the hardware realization. The proposed RC algorithm can break down the sequential data processing dependency, which is beneficial for

hardware realization in a pipelined H.264 video encoder. In addition, we also greatly reduce the internal memory buffer size for storing the MAD values, which amounts to 26% for QCIF, 59% for CIF, 83% for D1 reduction as compared to JM RC. This advantage is resulted from a new predictor model to predict the MAD value and target bits when it is realized in hardware. Moreover, the proposed RC algorithm possesses almost the same video quality as compared to JM. With these features, the proposed RC algorithm could be easily integrated with H.264 video encoder hardware design with low hardware complexity and high video quality.

The rest of this paper is organized as follows. We first describe the H.264 RC algorithm in Section 2. Then we illustrate the proposed algorithm in Section 3. The simulation results on the proposed algorithm are described in Section 4. Finally, we give a brief conclusion in Section 5.

2. H.264 RATE CONTROL ALGORITHM

The H.264 RC algorithm is divided into three different levels, i.e. GOP level, frame level, and BU level. In the following sub-sections, we will introduce the H.264 RC algorithm in more details.

2.1. GOP Level Rate Control

The default GOP consists of one I frame and twenty-nine P frames in H.264. When the j[th] frame in the i[th] GOP is coded, the remaining bits of the rest frames in the GOP are calculated as Eq. (1) shows:

$$R_i(j)=\begin{cases}\frac{B}{F}\times N_i\text{-}V_i(j) & j=1\\ R_i(j-1)-t_i(j-1) & j=2,3,....N\end{cases} \quad (1)$$

In Eq. (1), B denotes the channel bandwidth. F is frame rate. N_i is the total number of frame in GOP. The $t_i(j\text{-}1)$ is the actual bits of the $(j\text{-}1)^{th}$ frame. $R_i(j)$ is the remaining bits of the rest frames. $V_i(j)$ is the virtual buffer, which can be computed by Eq. (2) and updated each frame. More detailed information could be found in [1].

$$V_i(j)=V_i(j-1)+t_i(j-1)-\frac{B}{F} \quad (2)$$

2.2 Frame Level Rate Control

The frame level rate control can be divided into two stages, i.e. the pre-encoding stage and the post-encoding stage, which are respectively illustrated in the following.

2.2.1 Pre-encoding Stage

Each frame will compute the QP values via the following two steps:

Step 1: Determine the target bits for each P frame. The target buffer level (Tbl) that is predefined for each frame could be obtained from Eq. (3):

$$Tbl_i(j)=Tbl_i(j-1)-\frac{Tbl_i(I)}{N_{Tp}-1} \quad (3)$$

where N_{Tp} is total number of P frames and $Tbl_i(I)$ is the initial value. After encoding the first P frame of the GOP, the initial value is set as indicated in Eq. (4) for the updates in every GOP.

$$Tbl_i(2)=V_i(2) \quad (4)$$

Then, the available target bits for the current frame are allocated as shown in Eq. (5) as

$$\tilde{T}_i=\frac{B}{F}+\gamma\times(Tbl_i(j)-V_i(j)) \quad (5)$$

where γ is a weighted coefficient with the value set to be 0.5. Meanwhile, the average remaining bits of the current frame are calculated according to Eq. (6):

$$\hat{T}_i(j)=\frac{R(j)}{Np} \quad (6)$$

where Np is the number of the rest P frames in the GOP. The final target bits can be computed from $\tilde{T}_i(j)$ and $\hat{T}_i(j)$ according to Eq. (7) as

$$T_i(j)=\beta\times\hat{T}_i(j)+(1-\beta)\times\tilde{T}_i(j) \quad (7)$$

where the β is a weighted coefficient with the value set to be 0.5.

Step 2: Compute the quantization parameter. The MAD of the current frame is predicted by a linear prediction model as shown in Eq. (8), which uses the actual MAD of the previous frame.

$$MAD_i(j)=C1*MAD_i(j-1)+C2 \quad (8)$$

where *C1* and *C2* are two coefficients. The initial value of C1 and C2 are respectively set to 1 and 0 [1]. They are updated frame by frame or BU by BU during encoding. Then the quantization step (Qstep) of the current frame can be computed according to the quadratic R-Q model [1-2], as shown in Eq. (9):

$$T_i(j)=\frac{X1\times MAD_i(j)}{Qstep}+\frac{X2\times MAD_i(j)}{Qstep^2} \quad (9)$$

where *X1* and *X2* are two coefficients which [2,8,9] used. When we get the Qstep values, the QP values can be calculated by using the relationship between the quantization step and the quantization parameter of H.264 [1].

2.2.2 Post-encoding Stage

After encoding a frame, the coefficients C1, C2, X1 and X2 coefficients are updated [1].

2.3 BU Level Rate Control

A frame is composed of N_{mb} MBs. A basic unit (BU) is composed of a group of continue N_{bu} MB. For example, a BU is composed of 4 MBs if N_{bu} =4. Here shows the steps for doing BU level rate control in JM.

Step 1: Predicte the MAD as shown in Eq. (8) by using the previous frame co-located BU's actual MAD.

Step 2: Compute the target bits for the i[th] BU, as shown in Eq. (10):

$$T_{bu} = R_{bu} \times \frac{MAD^2(i)}{\sum_{k=1}^{N_{unit}} MAD_k^2(j)} - \mathrm{B}_{\mathrm{header}} \quad (10)$$

where R_{bu} denotes the remaining bits of the current frame with the initial value set to be $T_i(j)$. $\mathrm{B}_{\mathrm{header}}$ denotes the average header bits for all coded BU. N_{unit} is the number of the total BUs which can be obtained by Eq. (11):

$$N_{unit} = \frac{N_{mb}}{N_{bu}} \quad (11)$$

Step 3: Compute the Qstep as shown in Eq. (9), and translate Qstep to QP according to the way specified in H.264 [1].

3. PROPOSED PIPELINED RC ALGORITHM

The H.264 RC algorithm described in Section 2 is not good for hardware realization due to the inherent sequential processing operations. We propose a low complexity H.264 pipelined RC algorithm to break down the sequential data processing dependency for facilitating the hardware realization with good video coding quality. For the H.264 hardware encoder design [7] with pipelined control as shown in Figure 2, the H.264 BU-level (with BU=1MB) RC algorithm cannot be directly applied in the hardware encoder without causing the latency delay. For example, as illustrated in Figure 2, when the MB1 is coded in IME stage, it needs the QP value that generated by the rate control on the previous MB (i.e. MB0) that is coded just in FME stage. Therefore, based on the original H.264 BU-level RC algorithm, it will cause the latency delay when it is realized in a pipelined design.

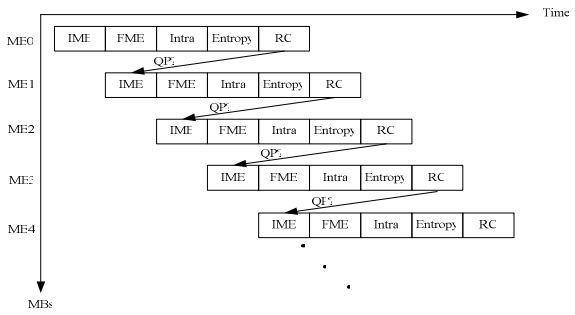

Figure 2. Pipelined scheduling in a H.264 hardware encoder

In order to solve the above mentioned problems, we first divide the rate control algorithm into two parts, i.e. Update QP model and Update RC&MAD model. Then we perform the Update QP model before doing IME for each MB from the 5th MB, and perform the Update RC&MAD model after doing Entropy coding, as shown in Figure 3 to release the sequential data processing dependency for the pipelined H.264 hardware encoder.

As shown in Figure 3, there is an interval of four MBs in both the Update RC&MAD model and Update QP model. When we want to calculate the target remaining bits for the Update QP model, the other three previous MB's total used bits are not calculated at all. Therefore, we propose a new algorithm to predict the total bits. In addition, the original H.264 RC algorithm also suffers from both high computational complexity and high local buffer requirement for storing intermediate parameters. For overcoming this problem, we use the average MAD in the previous frame instead of storing all the MAD values for BUs in the previous frame. Using this way can reduce large internal memory for storing MAD values.

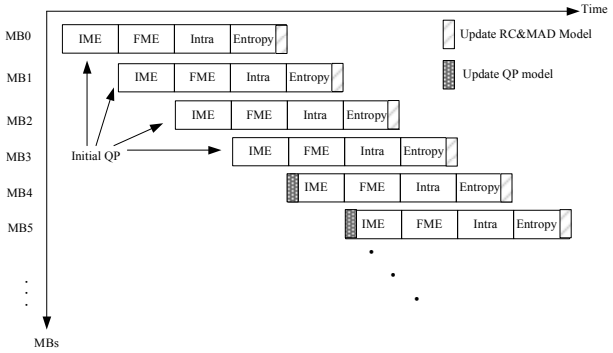

Figure 3. Proposed pipelined rate control algorithm for H.264

In the following, we describe the proposed pipelined RC algorithm with BU=1MB in terms of five steps.

Step 1: Initializing the Qp values for the first four BU's, as shown in Eq. (12):

$$\begin{aligned} &if\,(MB\_Number < 4) \\ &\quad Qp = Initial\_QP \end{aligned} \quad (12)$$

As shown in the Figure 3 in the pipeline architecture, the first four MBs don't have enough data to calculate the QP value. So we adopt the initial QP to solve this problem. If the 5th MB is encoded, the Update QP Model will be enabled to generate the QP values at IME stage for the encoded MB.

Step 2: Calculating the average value of MAD for the previous frame. The original H.264 RC algorithm uses the MAD of the previous co-located BU in order to predict the MAD of the current BU, which suffers from large buffer for storing the co-located MAD values in the hardware realization especially for high definition video. For reducing the memory buffer size, we adopt the average MAD value of the previous frame (i.e. *PFAVGMAD*) to predict the MAD of the current BU, as shown in Eq. (13):

$$PFAVGMAD = \left(\sum_{i=1}^{N_{unit}} MAD_i \right) \Big/ N_{unit} \quad (13)$$

Step 3: Using the PFAVGMAD to predict the MAD (i.e. *PdMAD*). Because we use the *PFAVGMAD* value to predict the MAD value, there will be some inaccuracy. For achieving accurate prediction on the predicted bits, we define a MAD ration, i.e. MAD_{ratio1}, for this purpose. We use the MAD_{ratio1} to improve the prediction accuracy, where the *PBUactMAD* denotes the previous BU's actual MAD value shown in Eq.(14):

$$\begin{aligned} PdMAD &= C1 \times PFAVGMAD \times MAD_{ratio1} + C2 \\ MAD_{ratio1} &= PFAVGMAD / PBUactMAD \end{aligned} \quad (14)$$

Step 4: Predicting the previous three BU's bits (i.e. PPBUBits), as shown in Eq. (15):

$$PPBUBits = PBUBits \times 3 \times MAD_{ratio2}$$
$$MAD_{ratio2} = PBUactMAD / PdMAD \quad (15)$$

where the *PBUBits* denotes the previous BU's actual bits and the *PPBUBits* denotes the predicted BU bits. For example, let us look at the MB4 under the Update QP model as shown in Figure 3. At that time just MB0 produces its actual bits but MB1, MB2, and MB3 don't, we predict their total bits according to Eq. (15) by using the MAD_{ratio2} to improve the prediction accuracy.

Step 5: Computing the target bits for the i^{th} BU. In order to reduce the compute complexity in computing the target bits for the i^{th} BU specified in Eq. (10), we calculate the target bits for the *i^{th} BU* according to Eq. (16):

$$T_{bu} = R_{bu} \times \left(\frac{PdMAD_t^2}{PFAVGMAD^2 \times NumofBU}\right) \times \mathrm{MAD}_{\mathrm{ratio2}} - \mathrm{B}_{\mathrm{header}} \quad (16)$$

In Eq. (10), it must calculate $(N_{unit}-1)+1/2 \times (N_{unit}-1)$ times in a frame. By multiplying the remaining *NumofBU's* value instead of summarizing the MAD^2, The complexity of calculating the target bits for the *i^{th}* frame is reduced from $O(n^2)$ to $O(n)$. In this way, we can greatly simplify its complexity. For QCIF sequence has 99 MBs in a frame, we compute our rate control algorithm for hardware design needs 9801 cycles per frame, and estimate Eq. (16) that needs 226 cycles per frame. We also predict Eq. (10) that needs 4185 cycles for one frame. Suppose that other conditions do not change, we has reduces the hardware cycles in terms of 28%. Under the same conditions, we can reduce 66% cycles for CIF, 87% cycles for D1.

4. SIMULATION RESULTS

We have realized the proposed RC algorithm on the JM10.2 [10]. Table 1 shows the simulation results of the proposed algorithm as compared to that of JM10.2 [10] with different video sequences at different bitrates. As shown in Table 1, the proposed RC algorithm possesses almost the same video quality as compared to that of H.264 JM10.2 [10].

Table 1. Simulation results and comparison

Sequence: Foreman, CIF, BU=1MB, Init_QP=24					
PSNR (dB)	Target Bitrate (kbps)	256	384	512	640
	JM 10.2 RC	32.99	34.76	35.93	36.85
	Proposed RC	32.99	34.77	35.94	36.84
PSNR Gain (dB)		0	0.01	0.01	-0.01
Sequence: News, CIF, BU=1MB, Init QP=24					
PSNR (dB)	Target Bitrate (kbps)	256	384	512	640
	JM 10.2 RC	37.98	40.12	41.64	42.73
	Proposed RC	38	40.17	41.67	42.77
PSNR Gain (dB)		0.02	0.05	0.03	0.04

Moreover, we use the average MAD instead of the previous frame co-located BU's MAD in the proposed RC algorithm, which results in only $N_{unit} \times 14bits$ required internal memory buffer size for storing MAD. This buffer size is under the assumption of using fixed point arithmetic with 5-bit integer and 9-bit fraction for representing MAD values. This feature is much helpful for the hardware realization of the proposed RC algorithm in a pipelined H.264 video encoder.

5. CONCLUSIONS

In this paper we have proposed a low complexity H.264 pipelined BU-level RC algorithm facilitating the hardware realization. The proposed algorithm not only breaks down the sequential data processing dependency in the original H.264 BU-level RC algorithm, but also greatly reduces the hardware cycles in terms of 28% for QCIF, 66% for CIF, 87% for D1 and buffer size requirements in terms of 26% for QCIF, 59% for CIF, 83% for D1 when realized in hardware. Moreover, the simulation results show that the proposed RC algorithm owns almost the same video quality as compared to that of H.264 JM 10.2.

6. REFERENCES

[1] K. P. Lim, G. Sullivan, T. Wiegand ,"Text Description of Joint Model Reference Encoding Methods and Decoding Concealment Methods," JVT-O079, Busan, Korea, April 2005.

[2] T. Chiang, and Y.Q. Zhang, "A new rate control scheme using quadratic rate distortion model," IEEE Transactions on CSVT, Issue 1, Volume 7, pp. 246-250, Feb. 1997.

[3] X. Yi, and N. Ling, "Rate control using enhanced frame complexity measure for H.264 video," IEEE Workshop on Signal Processing Systems, pp. 263-268, Oct, 2004.

[4] H. Yu, Z. Lin, and F. Pan, "An improved rate control algorithm for H.264," Proc. ISCAS'2005, Vol.1, pp. 312-315, May, 2005.

[5] M. Jiang, X. Yi, N. Ling, "Improved frame-layer rate control for H.264 using MAD ratio," Proc. ISCAS'2004, Vol.3, pp. III- 813-6, May, 2004.

[6] S. Su, S. Yu, and J. Zhou, "An improved Basic-Unit Layer Rate-Control Scheme on H.264," Proc. PDCAT'2005, pp. 815-819, Dec, 2005.

[7] T. C. Chen, Y. W. Huang, and L. G. Chen, "Analysis and design of macroblock pipelining for H.264/AVC VLSI architecture," Proc. ISCAS'2004, Vol.2, pp. II-273-6, May, 2004.

[8] H. J. Lee , T. H. Chiang and Y. Q. Zhang, "Scalable Rate Control for MPEG-4 Video," IEEE Transactions on CSVT, 10: pp. 878-894, 2000.

[9] A. Vetro, H. Sun and Y. Wang. "MPEG-4 rate control for multiple video objects," IEEE Transactions on CSVT, 9: pp. 186-199, 1999.

[10] http://iphome.hhi.de/suehring/tml/download/jm10.2.zip