A Computation-Aware Scheme for Motion Estimation in H.264

Fu-Chieh Chuang and Jin-Jang Leou
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan 621, Republic of China
E-mail: jjleou@cs.ccu.edu.tw

Abstract
In this study, a computation-aware scheme for motion estimation (ME) in H.264 is proposed. The objective of the proposed scheme is to properly distribute the available ME computations of the H.264 encoder. Here, the temporal motion vector prediction technique is used to get the predicted motion vector (PMV) of each macroblock (MB). The sum of absolute components of the PMVs of all the MBs in a video frame is used as the measure to allocate the target computation to a video frame. The proposed scheme contains four phases: 1) frame-level computation allocation, 2) MB-level computation allocation, 3) mode-level computation allocation, and 4) pixel-level computation allocation. As compared with the comparison scheme, the proposed computation-aware schemes for motion estimation in H.264 usually have the better performance (average PSNR improvement and bit rate increment) in most simulation cases.

Keywords: H.264 video coding, computation-aware video coding, motion estimation.

1. Introduction
In general, the computational complexity of the H.264 encoder is higher than that of the H.264 decoder and the most time-consuming portion of the H.264 encoder is ME (motion estimation) [1]. Because fast search ME operations may consume over 80% of the computational power of the H.264 encoder, conventional fast ME algorithms [2-4] usually focus on reducing computations, i.e., reducing the number of checking points required for each MB or reducing the computation complexity of a checking point. However, if the total computations are not sufficient to finish the whole video coding (or ME) procedure, the encoding process may be forced to terminate suddenly.

Tai, et al. [5] proposed a computation-aware scheme, which contains two phases: (1) frame-level computation allocation and (2) block-level computation allocation. Within the frame-level phase, two parameters, \( C_{FP} \) and \( C_{BP} \), stand for backward prediction and forward prediction, respectively, when the \( i \)th frame is handling. \( C_{BP} \) is the average of remaining computation power given to the \((n-i+1)\) frames that are not coded, whereas \( C_{FP} \) is got from the relationship between the amount of computation power which has been used and the mean square error (MSE) gained by using a block matching algorithm (BMA). Based on the two parameters, \( C_{BP} \) and \( C_{FP} \), a smooth section will be decided to give an acceptable range of computation power of the \( i \)th frame. Within the block-level phase, a so-called predicted computation-distortion benefit (PCDB) list is established, according to the order of initial mean square error (MSE) values at \( mv(0, 0) \) of all blocks in the current video frame. Once the PCDB list is initialized, their approach greedily allocates the computation power (in unit of checking points) to the first block on the PCDB list in order to perform the next step. The PCDB list is updated by the returned MSE value, which corresponds to the new motion vector found in the search step. This process is repeated until the target computation is exhausted.

In the frame level, the initial MSE of a frame should be calculated before the ME process can start. In the block level, all initial MSEs of all blocks in a frame should be calculated before the PCDB list is established, and at each PCDB list updating, one more MSE value will be calculated and if the order of the PCDB list changes, the unfinished ME of the current block will be forced to stop its ME process. The initial MSE calculations of all blocks in a frame and switching the ME process due to changing of the PCDB list are some kind of redundancy. To overcome the weaknesses of the existing approach, in this study, a
computation-aware scheme for motion estimation (ME) in H.264 is proposed.

2. Proposed scheme

2.1. Overview

The proposed computation-aware scheme for ME in H.264 is shown in Fig. 1. For a video sequence containing \( n \) frames, let \( C_{\text{total}} \) be the total computation allocated to the \( n \) frames, \( f_i \) be the \( i \)th frame, and \( M_{ij} \) be the \( j \)th MB. In this study, there are four phases: (1) frame-level computation allocation, (2) MB-level computation allocation, (3) mode-level computation allocation, and (4) pixel-level computation allocation. Frame-level computation allocation determines the target computation (\( CF_i \)) for \( f_i \), MB-level computation allocation determines the target computation (\( CMB_j \)) for \( MB_j \), mode-level computation allocation distributes \( CMB \) to two different ME modes (MB and block), and pixel-level computation allocation distributes each mode to two different pixel ME modes. In the frame and MB levels, the remaining computations are fed backward to update the available computations for the remaining frames and MBs, respectively.

In general [2], the number of checking points is treated as the measurement unit of computation power. In H.264, the \( SAD \) (sum of absolute differences) calculation is performed in a row-by-row manner and each partial \( SAD \) value containing a new row will be compared with the minimal \( SAD \) value, \( SAD_{\text{min}} \), at that time. Based on the row-based partial \( SAD \) computation, \( SAD \) computations of different search positions may differ. Therefore, in this study, computation allocation is in terms of the row-based partial \( SAD \) (SADR). To unify two different block sizes, i.e., 8×8 and 16×16, in H.264, the one-row partial \( SAD \) computation of 8×8 blocks, SADR, is used as the computation measure.

In this study, temporal motion vector (MV) prediction [6] is employed to allocate total (available) computation (\( C_{\text{local}} \)) at the frame and MB levels. Temporal MV projection tracks the motion vector of a moving object from one frame to the next frame. For an MB in frame \( f_i \), the corresponding temporally neighboring MVs in the previous frame \( f_{i-1} \) whose “motion-projected” MBs in the current frame will overlap the MB in frame \( f_i \). Here, if an MB in frame \( f_i \) has no temporally motion-projected overlapping MBs, the predicted MV (PMV) of \( f_{i-1} \) is set to \((0,0)\). If an MB in frame \( f_i \) has only one temporally motion-projected overlapping MB, the PMV of such an MB is set to the corresponding MV of the temporally motion-projected overlapping MB in frame \( f_{i-1} \). If an MB in frame \( f_i \) has two or more temporally motion-projected overlapping MBs, the PMV of the MB is set to the corresponding MV of the temporally motion-projected overlapping MB which has the longest \( MV_{SAC} \) value. Here if the two components of an MV is \((MV_x, MV_y)\), the sum of absolute components (SAC) of the MV is defined as

\[
MV_{SAC} = |MV_x| + |MV_y|
\]

and the total sum of absolute components of all the \( K \) MVs in frame \( f_i \) is given by

\[
TMV_{SAC,i} = \sum_{k=1}^{K} MV_{SAC,i,k}
\]

where \( MV_{SAC,i} \) is the \( MV_{SAC} \) value of the \( k \)th MV in \( f_i \).

2.2. Frame-level computation

For frame \( f_i \), let \( CBP_i, CFP_i \), and \( CR_i \) be the backward PMV computation, forward PMV computation, and real MV computation for all MBs within frame \( f_i \), respectively. The average computation cost (in terms of SADR) per \( MV_{SAC} \) for the pervious \( i-1 \) frames is defined as

\[
\mu_{ave} = \frac{\sum_{j=1}^{n-i} CR_j}{\sum_{j=1}^{n-i} TMV_{SAC,j}}.
\]

If the predicted \( TMV_{SAC,j} \) value for frame \( f_i \) is denoted as \( PTMV_{SAC,i,j} \), the forward PMV computation for frame \( f_i \) can be given by

\[
CFP_j = \mu_{ave} \times PTMV_{SAC,i,j}.
\]

Because the related information for the “remaining” frames is not available for frame \( f_i \), the backward PMV computation for frame \( f_i \) is set to the remaining computations divided by the number \((n-i+1)\) of remaining frames within the video sequence, i.e.,

\[
CBP_j = \left( C_{\text{total}} - \sum_{j=1}^{n-1} CR_j \right) / (n-i+1).
\]

To avoid over-use of the available computation, it is claimed [2] that \( CFP_j \) smaller than or equal to \( CBP_j \) implies that insufficient computations are allocated to the previous \( i-1 \) frames and thus more computation can be allocated to frame \( f_i \). On the contrary, \( CFP_j \) larger than \( CBP_j \) indicates that too much computation is allocated to the previous \( i-1 \) frames and the average computation allocated to each of the remaining \( n-i+1 \) frames should be gradually decreased. In this study, the target computation \( CF \) for frame \( f_i \) is given by

\[
CF_j = \begin{cases} 
\alpha \times CBP_j, & \text{if } CFP_j \leq CBP_j, \\
\min(\alpha \times CBP_j, CFP_j), & \text{otherwise},
\end{cases}
\]

where \( \alpha \) is a given constant.
2.3. MB-level computation

After the target computation \( CF_i \) for frame \( f_i \) has been determined, if there are \( P \times Q \) MBs in frame \( f_i \), let the current macroblock of frame \( f_i \) be \( MB_{p,q} \), \( PMV_{SAC(p,q)} \) be the predicted MV of \( MB_{p,q} \) and \( PTMV_{SAC(i)} \) be the predicted TMV of \( f_i \). The target computation for MB \( p,q \), \( CMB_{p,q} \), can be allocated in compliance with the \( PMV_{SAC(p,q)} \) proportion taken in \( PTMV_{SAC(i)} \), which is given by

\[
CMB_{(p,q)} = CF_i \times \frac{PMV_{SAC(p,q)}}{PTMV_{SAC(i)}} \tag{8}
\]

To avoid the situation that \( PMV_{SAC(p,q)} = 0 \) would make \( CMB_{p,q} = 0 \), a basic value is added to \( PMV_{SAC(p,q)} \) when storing \( PMV_{SAC} \) to each MB in \( f_i \). In this study, the basic value is 1. Moreover, to prevent another situation that the allocated computation for \( MB_{p,q} \) has not been exhausted, the remaining computations of \( MB_{p,q} \) would be fed backward to update \( CMB_{p,q+1} \).

2.4. Mode-level computation

Table 1 shows the percentage of blocks whose number of search positions for finding the best MV in each Inter-mode of the QCIF “Akiyo,” “Silent,” and “Foreman” sequences. We can find that over 75% of blocks of the four prior modes (Inter16×16, Inter16×8, Inter8×16, and Inter8×8) search less than 30 positions, and 20% of blocks search over 110 positions. In Inter8×4 and Inter4×8 modes, the percentage of blocks searching less than 20 positions is 55%, and 40% of blocks search over 110 positions. 99% of the blocks of the last mode (Inter4×4) search less than 20 positions.

Based on this observation, in this study, after the target computation \( CMB \) for an MB is allocated for the first-grade mode, the remaining computations will be allocated for the second-grade mode until \( CMB \) is exhausted.

2.5. Pixel-level computation

In H.264 JM96, motion estimation (ME) is conducted into integer-pixel ME and then fractional-pixel ME around the position obtained by the integer-pixel ME. For integer-pixel ME, motion vector prediction is used to select a start search position for ME search in H.264. The distance between the best MV (BMV) determined by the fast search strategy in H.264 and the predicted motion vector (PMV) in H.264 is defined as \( MVD \):

\[
MVD = |BMV_x - PMV_x| + |BMV_y - PMV_y| \tag{9}
\]

As shown in Table 2, about 90% of \( MVD \) is smaller than 1, i.e., the predicted MV is very close to the best MV.

Because fractional-pixel ME can provide significantly better compression performance than integer-pixel ME, in this study, the fractional-pixel ME has the higher priority to the integer-pixel ME.

3. Simulation Results

Six QCIF video sequences, “Akiyo,” “Silent,” “News,” “Foreman,” and “Stefan,” are used to evaluate the performance of the proposed scheme. Each sequence consists of 100 frames, in which the first video frame is an I-frame, followed by 99 P-frames. The search range is set to 16 and quantization parameter (QP) is set to 28. Four performance measures are employed in this study. They are: (1) the average PSNR improvement (dB) with respective to JM_FME, (2) the average bit rate per second (kbps), (3) the total number of search positions, and (4) the total encoding time (s).

Here, one existing fast search algorithm and three comparison computation-aware schemes are implemented. They are: (1) JM_FME, which is the fast search algorithm in H.264 JM96, (2) CA_FME [5], which is the computation-aware scheme for software-based ME with JM_FME, (3) PRO FME, which is the proposed computation-aware scheme with JM_FME, (4) PRO DC, which is the proposed computation-aware scheme with adaptive dual-cross search algorithm. Some simulation results are listed in Table 3.

4. Concluding remarks

Based on the simulation results obtained in this study, given different available motion estimation computations (MECs), both the average PSNR and bit rate increment of the two proposed schemes are better than that of the comparison scheme CA_FME.
5. References


Table 1. The relationship between the number of search positions (SP) and percentages of modes chosen after ME.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Sequence</th>
<th>SP=0</th>
<th>SP=20</th>
<th>SP=40</th>
<th>SP=60</th>
<th>SP=80</th>
<th>SP=120</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-16</td>
<td>Akiyo</td>
<td>90.12</td>
<td>2.49</td>
<td>0.10</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>Silent</td>
<td>81.11</td>
<td>9.42</td>
<td>0.07</td>
<td>0.06</td>
<td>0.32</td>
<td>9.02</td>
</tr>
<tr>
<td></td>
<td>Foreman</td>
<td>43.59</td>
<td>10.22</td>
<td>0.04</td>
<td>0.01</td>
<td>6.11</td>
<td>20.03</td>
</tr>
<tr>
<td>Inter-8</td>
<td>Akiyo</td>
<td>85.36</td>
<td>9.64</td>
<td>0.07</td>
<td>0.07</td>
<td>0.30</td>
<td>25.51</td>
</tr>
<tr>
<td></td>
<td>Silent</td>
<td>83.52</td>
<td>8.27</td>
<td>0.08</td>
<td>0.34</td>
<td>8.30</td>
<td>91.11</td>
</tr>
<tr>
<td></td>
<td>Foreman</td>
<td>32.50</td>
<td>18.77</td>
<td>0.38</td>
<td>0.34</td>
<td>8.30</td>
<td>91.11</td>
</tr>
<tr>
<td>Inter-8x8</td>
<td>Akiyo</td>
<td>45.79</td>
<td>17.59</td>
<td>0.17</td>
<td>1.52</td>
<td>3.84</td>
<td>36.69</td>
</tr>
<tr>
<td></td>
<td>Silent</td>
<td>46.30</td>
<td>19.42</td>
<td>0.27</td>
<td>1.51</td>
<td>5.46</td>
<td>15.75</td>
</tr>
<tr>
<td></td>
<td>Foreman</td>
<td>8.34</td>
<td>0.38</td>
<td>0.01</td>
<td>0.01</td>
<td>25.76</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 2. Percentages of different MVDs.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akiyo</td>
<td>93.77</td>
<td>5.69</td>
<td>0.49</td>
<td>0.04</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Silent</td>
<td>85.59</td>
<td>11.29</td>
<td>2.39</td>
<td>0.19</td>
<td>0.05</td>
<td>0.03</td>
<td>0.02</td>
<td>0.06</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>Foreman</td>
<td>81.39</td>
<td>13.88</td>
<td>2.88</td>
<td>0.83</td>
<td>0.27</td>
<td>0.13</td>
<td>0.04</td>
<td>0.01</td>
<td>0.29</td>
<td>0.29</td>
</tr>
<tr>
<td>Inter-8</td>
<td>55.95</td>
<td>15.83</td>
<td>9.92</td>
<td>4.41</td>
<td>0.09</td>
<td>0.41</td>
<td>0.09</td>
<td>0.41</td>
<td>0.09</td>
<td>0.41</td>
</tr>
<tr>
<td>Inter-4</td>
<td>55.96</td>
<td>34.61</td>
<td>7.38</td>
<td>1.21</td>
<td>0.46</td>
<td>0.25</td>
<td>0.04</td>
<td>0.04</td>
<td>0.07</td>
<td>0.07</td>
</tr>
<tr>
<td>Inter-4</td>
<td>59.12</td>
<td>30.58</td>
<td>7.26</td>
<td>1.50</td>
<td>0.58</td>
<td>0.22</td>
<td>0.15</td>
<td>0.17</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>Average</td>
<td>75.28</td>
<td>18.65</td>
<td>4.23</td>
<td>0.90</td>
<td>0.37</td>
<td>0.20</td>
<td>0.09</td>
<td>0.09</td>
<td>0.23</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Table 3. Simulation results for the “Akiyo” sequence with different percentages of available search positions of three comparison schemes.

<table>
<thead>
<tr>
<th>Bit Rate (kbps)</th>
<th>Average PSNR (dB)</th>
<th>Average bit rate increment (kbps)</th>
<th>Average search positions per frame (k)</th>
<th>Time speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>38.50</td>
<td>33.70</td>
<td>80.02</td>
<td>25.795 x 10^3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CA</td>
<td>FME</td>
<td>PRO</td>
<td>CA</td>
</tr>
<tr>
<td>90%</td>
<td>-0.02</td>
<td>-0.00</td>
<td>0.00</td>
<td>-0.01</td>
</tr>
<tr>
<td>80%</td>
<td>-0.02</td>
<td>-0.00</td>
<td>0.00</td>
<td>-0.01</td>
</tr>
<tr>
<td>70%</td>
<td>-0.19</td>
<td>-0.01</td>
<td>-0.01</td>
<td>-0.28</td>
</tr>
<tr>
<td>60%</td>
<td>-0.30</td>
<td>-0.00</td>
<td>-0.00</td>
<td>-0.44</td>
</tr>
<tr>
<td>50%</td>
<td>-0.43</td>
<td>-0.00</td>
<td>-0.00</td>
<td>-0.60</td>
</tr>
<tr>
<td>40%</td>
<td>-0.48</td>
<td>-0.00</td>
<td>-0.00</td>
<td>-0.60</td>
</tr>
<tr>
<td>30%</td>
<td>-0.50</td>
<td>-0.00</td>
<td>-0.00</td>
<td>-0.60</td>
</tr>
<tr>
<td>20%</td>
<td>-0.58</td>
<td>-0.00</td>
<td>-0.00</td>
<td>-0.60</td>
</tr>
<tr>
<td>10%</td>
<td>-0.63</td>
<td>-0.00</td>
<td>-0.00</td>
<td>-0.60</td>
</tr>
</tbody>
</table>

Fig. 1. The proposed computation-aware scheme.