Analysis of Motion Estimation Algorithm in HEVC

Spring 2014
Multimedia Processing - EE5359

Advisor: Dr. K. R. Rao
Department of Electrical Engineering
University of Texas, Arlington

Tuan Ho
1001006444
tuanpho@mavs.uta.edu

Update: 3/01/2014
Acronym

- 2DLOG: Two dimensional logarithmic
- CTU: Coding Tree Unit
- CU: Coding Unit
- FSS: Four steps search
- HEVC: High Efficiency Video Coding
- HD: High Definition
- JCT-VC: Joint Collaborative Team on Video Coding
- PU: Prediction Unit
- ME: Motion Estimation
- MPEG: Moving Picture Experts Group
- NTSS: New three steps search
- OSA: Orthogonal search algorithm
- OTA: One-at-a-time search
- SAO: Sample Adaptive Offset
- T.B.D: To Be Determined
- TSS: Three steps search
- VCEG: Video Coding Experts Group
Overview about HEVC [1]:


The Joint Collaborative Team on Video Coding (JCT-VC) has been established to work on this project.

The Joint Collaborative Team on 3D Video Coding Extension Development (JCT-3V) has been established to work on 3D video coding extensions of HEVC and other video coding standards [1].

Figure 1 shows the block diagram of an HEVC encoder (with decoder components modeling in light gray) [1].
HEVC as an Hybrid-video coding scheme with quadtree-based techniques

HEVC, like its predecessor H.264, is an hybrid-video coding scheme that comprises of a block-based prediction followed by a transform coding; however, the block-based paradigm in HEVC has drastically improved by the introduction of the quadtree-based techniques that gives more flexibility for both the prediction and transform coding.

In HEVC input video signal is split into smaller partitions called coding tree units (CTU) by quadtree signaling process. The CTU has one luma CTB (coding tree block) whose size can be 64x64, 32x32 or 16x16 of luma sample data, corresponding chroma CTBs and syntax data. Larger size typically gives better compression.

Figure 2 shows an example of a 128x128 pixels area divided into variable-sized Coding Units (CUs). The first 64x64 area (I) is divided into four 32x32 CUs (second depth). The second 64x64 area (II) is encoded as just one 64x64 CU (first depth). The third 64x64 area (III) is divided into four 32x32 CUs, two of which are divided into four 16x16 CUs, one of which is divided into four 8x8 CUs (fourth depth). Finally, the fourth 64x64 area (IV) is divided into four 32x32 CUs and two of them are divided into four 16x16 CUs (third depth) [15].
Another example of the quadtree process [23] is shown in Figure 3.

![Figure 3: Example of a coding quadtree with a 64x64 coding tree block that is divided into smaller, variable-size coding blocks (above). As an example of further subdivisions, the 32x32 coding block number 14 is partitioned one into two prediction blocks (below left) and one into variable-size transform blocks using the residual quadtree (below right). [23]](image)

The quadtree-based techniques offers great flexibility and efficiency for the prediction and transform coding; however, it also significantly raises the computational complexity, especially in the motion estimation [6].

**Motion Estimation**

Motion estimation is the process of finding the motion vectors to reduce the temporal redundancy among frames in the video signal. To obtain such vector, blocks in the current frame are compared with those in the reference frames (the number of the frames to be compared can be one or more); an area for where the search is performed is defined to narrow down the finding and ease the processing computation. A motion vector is formed when a block in the reference frame
found to be best-matched, i.e. having the most similarity with the one being searched. **Figure 4** gives an illustration for the motion estimation process.

![Figure 4: Motion Estimation Illustration [3]](image)

Recently, to enable the real-time application of the beyond-HD encoding, the video codec has been widely adopted at hardware implementation level. Therefore, it is crucial to find fast and power-efficient hardware-friendly algorithm for video codec, especially for motion estimation.

To achieve good results, many search algorithms have been proposed for H.264 to find best motion vector for the largest 16x16 block of samples. As the largest block size in HEVC increases upto 64x64, it drastically raises the workload for the estimation process compared to H.264. Research [16,17] shows that the running time of motion estimation takes upto 81% portion of the entire encoding process. As a result, fast algorithms for HEVC motion estimation are essential to reduce the entire computational complexity.

![Figure 5: These execution timeresults were obtained by profiling the encoding of the “BasketballDrive” Full HD (1920×1080) sequence. [17]](image)
As it is necessary to perform the search/compare for all the blocks inside a frame within the reference picture, which can be as large as HD size 1920x1080, the motion estimation consumes lots of computation time for processing. This makes it the most exhausted process in the video encoding. As the thirst for higher resolution increases drastically in video codecs, especially HEVC, motion estimation's computational complexity becomes extremely critical.

**Interpolation**

HEVC interpolation process is described in Figures 6 - 8.

![HEVC Interpolation: illustration of a block; A(i,j) are the integer pixels, a(i,j) and c(i,j) are half pixels, b(i,j) are quarter pixels](image)

![HEVC Interpolation Filter Coefficients](image)
\[
\begin{align*}
a_{0,j} &= \left( \sum_{i=-3}^{-1} A_{i,j} qfilter[i] \right) \gg (B - 8) \\
b_{0,j} &= \left( \sum_{i=-3}^{-4} A_{i,j} hfilter[i] \right) \gg (B - 8) \\
c_{0,j} &= \left( \sum_{i=-2}^{-4} A_{i,j} qfilter[1 - i] \right) \gg (B - 8) \\
d_{0,0} &= \left( \sum_{i=-3}^{-3} A_{0,j} qfilter[j] \right) \gg (B - 8) \\
h_{0,0} &= \left( \sum_{i=-3}^{-4} A_{0,j} hfilter[j] \right) \gg (B - 8) \\
n_{0,0} &= \left( \sum_{j=-2}^{-4} A_{0,j} qfilter[1 - j] \right) \gg (B - 8)
\end{align*}
\]

*Figure 8: HEVC Interpolation Filters [1]*
**Approaches**

**Optimized Search Algorithms instead of Full-search for ME**

Despite of its quality, the full-search algorithm is time-consuming and not suitable for implementation in real-time process. Lots of alternative methods have been proposed in the recent years [6]. Figures 9 and 10 shows some of the fast search algorithms.

![Figure 9: Fast search algorithms - (a) Two dimensional logarithmic (2DLOG), (b) Three steps search (TSS), (c) New three steps search (NTSS) [6]](image)

![Figure 10: Fast search algorithms - (a) Four steps search (FSS), (b) One-at-a-time search (OTA), (c) Orthogonal search algorithm (OSA) [6]](image)

**Hierarchical Search for ME**

To increase the efficiency of search algorithm and obtain higher quality, the searches can be further divided by two main steps. The first is to achieve an
integer best-match using fast search algorithms. From this new anchor the sub-pixel search is performed to create more accurate motion vector. This hierarchical search is shown in Figure 11 [6].

Complexity Control Employment

By limiting the number of the depths in CU partition, the computational complexity can be reduced. [15]

Figure 12 shows the average percentage of computational complexity demanded for encoding CUs belonging to each depth of the tree structure. CU depths are presented on the y-axis of Figure 12 and the computational complexity is presented on the x-axis. As mentioned before the nature of the data structures used in HEVC leads to nested encoding loops, such that CUs at higher tree depths are encoded inside CUs at smaller tree depths. This is shown in Figure 12 as follows: for each CU depth (vertical axis), the computational complexity is divided into three components: (1) the complexity of performing inter prediction for CUs at the current depth (in grey); (2) the complexity of coding the same image area as
CUs at a higher depth (in white); and (3) the complexity of other operations (in black) [15].

![Figure 12: Complexity of processing regarding to CU Depth [15]](image)
References


[12] HEVC Specification: Complexity control of high efficiency video encoders for power-constrained devices http://phenix.it-


