Joint DBF and SAO Parallel Filtering Based on Multithread Load Balancing

Hao Ma¹, Dong Hu¹,²,³*, and Yi Li¹

¹ Jiangsu Province’s Key Lab of Image Procession and Image Communications, Nanjing, 210003, China
² Education Ministry’s Key Lab of Broadband Wireless Communication and Sensor Network Technology, Nanjing, 210003, China
³ Education Ministry’s Engineering Research Center of Ubiquitous Network and Heath Service, Nanjing University of Posts and Telecommunications, Nanjing, 210003, China

*Email: hud@njupt.edu.cn

Abstract. This paper presents a joint parallel loop filtering algorithm based on multi-thread load balancing in HEVC decoding, which implements the parallel processing of deblocking filtering (DBF) and sample adaptive compensation (SAO). Because of the diversity of video, the texture of different regions in an image is also different, which leads to various CTU partition methods. Therefore, the number of the boundary to be filtered is greatly different, resulting the computation load among multiple threads unbalanced in parallel processing. To solve this problem, an area division scheme is proposed, which divides the image into multiple areas, and the number of boundaries to be filtered in each area is similar. Then, the mapping relationship table is used to allocate these areas to multiple threads for parallel processing, so as to achieve the load balancing among the filtering threads. Finally, the cache technology is used to combine DBF and SAO to reduce the delay between them and improve the overall parallelism of the loop filter. Experimental results show that the performance of the proposed load balancing joint filtering algorithm is 8.15% higher than the previous scheme.

1. Introduction

In the H.265/HEVC standard, blocking effect and ringing effect still exist after loop filtering, because the coding framework adopted is the same as the previous standard, which is a block-based mixed coding framework. Blocking effect and ringing effect are two common phenomena in the process of video decoding. These phenomena have a great impact on the accuracy of video decoding and restoration, and seriously affect the video viewing experience. The H.265/HEVC standard adopts loop filtering (ILF) technology to reduce the loss of image quality caused by these two kinds of interference [1], including deblocking filtering (DBF) and pixel sample adaptive compensation (SAO). The deblocking filter is similar to that in H.264/AVC, but there are many improvements that make it easier to design parallel structures and reduce computational complexity [2]. SAO, on the other hand, aims to reduce the ringing effect that may occur after de-blocking filtering, so as to further enhance the quality of the reconstructed image.

According to the coding and decoding framework of H.265/HEVC standard, the subsequent images need to refer to the previous images after loop filtering in the process of inter-frame prediction [3]. Therefore, loop filtering plays an important role in the standard. We will study the loop filtering...
module, analyze the data characteristics of deblocking filtering and sample adaptive compensation, and design a parallel loop filtering scheme with better performance based on these analyses [4].

The rest of this paper is organized as follows. In section 2, the deblocking filtering method and fast fusion loop filtering method based on parallel data level are introduced, and the existing problems are analysed. Section 3 introduces the DBF+SAO joint filtering method based on multithreaded load balancing in detail. We present the results and analysis in section 4, and then give a brief conclusion in section 5.

2. Related Works

2.1 Deblocking Filter Based on Data Level Parallel
In the process of deblocking filtering, the horizontal filtering in the vertical direction is carried out first, and then the vertical filtering in the horizontal direction is carried out [5]. Because there is no data level dependency between adjacent blocks in the process of horizontal and vertical filtering, HEVC deblocking filter is generally considered to be suitable for data level parallel (DLP). However, unbalanced workload will increase the idle time of some threads, so using DLP alone will not get the maximum parallel performance. In reference [6], a data level parallel deblocking filter is proposed. Deblocking filter is divided into two stages: horizontal filter and vertical filter. The average image is segmented according to the number of CTU lines in each stage, so that deblocking filter can be applied in parallel under similar workload.

2.2 Fast Fusion Loop Filtering
In reference [7], a scheme of fast fusion loop filtering is proposed. The loop filtering is regarded as a whole. Considering the dependence between deblocking filtering and Sao, a structure of CTU like unit is proposed to realize the coupling of two links, thus improving the overall efficiency of loop filtering.

2.3 Task of This Work
According to the above analysis, although the previous studies have greatly optimized the deblocking filter, they have not realized the load balance between threads in parallel filtering, that is, the impact of local texture complexity on thread workload has not been considered in detail, and the combination of DBF and SAO has not been considered. Therefore, this paper proposes a combined filtering method of DBF + SAO based on Multithreading load balancing.

3. Formatting the Text

3.1 Load Balancing based on CTU Partition Depth Estimation
The different texture of image region makes the different features of CTU division [8]. According to the depth of CTU division and the TU and PU division information, the filtering boundary in the image can be estimated, and the total workload can be distributed to multiple threads equally, so as to realize the load balancing among multiple threads in the parallel block-removing filtering as much as possible [9]. Therefore, it is necessary to design the calculation estimation model and the region division scheme. However, it is necessary to estimate the computational complexity before deblocking filtering for each image, which will lead to additional computational overhead. Therefore, it is necessary to divide the estimated area reasonably in order to reduce the impact of additional calculation.
First, estimate the number of boundaries to be filtered, because most of the computational complexity of deblocking filtering is generated during filtering and boundary strength calculation. At the same time, only when at least one of the two adjacent blocks is in the intra coding mode can the chrominance component boundary filtering be performed. For P-frames and B-frames, there are fewer intra-coded blocks, so that the demand of chrominance filtering is relatively small. Therefore, only the filtering boundary of the luminance component CB is considered here. As shown in Figure 1, it is a 64x64 size CTU, and it is assumed here that it is not at the boundary of the image. The lines of different thickness are used to represent the CU of different depth, and the circle pattern is used to mark all possible boundaries to be filtered when the corresponding luminance component CB is deblocking filtering.

In H.265/HEVC standard, when a CU is encoded by intra prediction mode, it can be divided into $2N \times 2N$ or $N \times N$ size ($n = 8, 16, 32$) PU. When PU appears in CU with $N \times N$ as the minimum size, its boundary will coincide with that of the intra coded CU. For the inter prediction coding CU, it can be divided into symmetrical and asymmetric PU of various sizes. However, in the way of quadtree, many PU boundaries will overlap with TU boundaries. TU only has symmetrical partition type, which is relatively simple to analyze, and it is also the basic unit of transformation, and transformation is also one of the main factors that produce block effect. Therefore, the filtering boundary of CU can be estimated according to the partition information of TU in CU and the depth of CU. The test video sequence used in this design does not adopt any parallel coding scheme, that is, a frame contains only one slice, and then a frame can be divided into CTU lines in this slice. Therefore, only in the frame header, according to the syntax elements in slice header or PPS, can we judge whether the deblocking filter switch of all CTU in the current frame is on or off. When the complexity of filtering is estimated, it is in the order of raster scanning. When traversing, if the judgment switch is off, the current frame will not be filtered. Therefore, this paper explores the opening of filter switch.

$$\text{Comp}_{frame} = \sum_{j=0}^{L-1} \text{Comp}_{CTU}^{i,j}$$

(1)

The computational complexity of a frame filter is the sum of the computational complexity of each CTU filter. Therefore, the computational complexity estimation of a frame can be in the formula (1). $\text{Comp}_{frame}^i$ represents the computational complexity estimation of deblocking filtering in frame i, L is the number of CTU in a frame, and $\text{Comp}_{CTU}^{i,j}$ represents the estimation of the complexity of filtering calculation of the j-th CTU in frame i, as shown in formula (2), $\text{Comp}_{CTU}^{i,j,k}$ represents the
estimated complexity of filtering calculation of the k-th quadtree leaf node CU in the j-th CTU in the i-th frame, and N represents the number of quadtree leaf nodes CU in the CTU. The estimated value of the complexity of filtering calculation of each leaf node CU in CTU is accumulated, and the corresponding estimated value of CTU is obtained. Therefore, the final estimation of the complexity of filtering calculation for each CU is to estimate the number of filtering boundaries of the luminance component CB in the CU.

\[
\text{Comp}_{CTU}^{i,j} = \sum_{k=0}^{N-1} \text{Comp}_{CU}^{i,j,k}
\]  

(2)

As mentioned above, the number of 8×8 block boundaries in the luminance component CB can be estimated according to the depth of the quadtree leaf node CU and the TU division information, that is \(\text{Comp}_{CU}\). For a 64×64 size CTU, if the current CU does not have TU division, the relationship between the 8×8 block boundary number of the depth and brightness component is as follows: the 8×8 block boundary number is 16, 8, 4 and 2, respectively, corresponding to the CU depth of 0, 1, 2 and 3. If a CU is TU divided, the corresponding number of 8×8 block boundaries of the corresponding luminance component is 32, 16, 8 and 2. Based on this, the relation table of 8×8 block boundary numbers of CU depth and luminance component CB can be obtained, as shown in Table 1.

| CU depth | CU size   | TU not Divided | TU Divided |
|----------|-----------|----------------|------------|
| 0        | 64×64     | 16             | 32         |
| 1        | 33×32     | 8              | 16         |
| 2        | 16×16     | 4              | 8          |
| 3        | 8×8       | 2              | 2          |

Table 1. Estimated number of filter boundaries.

It can be seen from the table that the number of boundaries is doubled in all three cases except for CU depth of 3 and CU size of 8×8, when the CU size is 8×8, the TU size will be smaller than 8×8, which is not in line with the execution bar of deblocking filter. Therefore, the array corresponding to CB filter boundary of luminance component in two CU can be generated, which respectively corresponds to whether the current CU has been TU divided. Array 1 is \{32, 16, 8, 2\}, indicating that CU has been TU divided, array 2 is \{16, 8, 4, 2\}, indicating that CU has not been TU divided.

Put CU depth CU_Depth as array index, initializes CTU complexity CTU_Comp is 0. Traverse all CTU in the order of raster scanning to obtain CU. During the scanning process, the main thread stores the location index of each CTU in order. After the estimation of the computational complexity of a frame deblocking filter is completed, the region of the image should be divided according to the overall complexity, so that the workload allocated by each thread is balanced as much as possible. Basically, a frame of image is divided into N regions according to the number of allocated threads and the estimated total computational complexity, which is estimated by assigning the same amount of computation to all regions. The average amount of work allocated for each area is defined as equation (3). Among them, \(\text{Comp}_{frame}^{j}\) represents the overall estimated calculation workload in a frame. \(r_x\) represents the X (0≤x≤N-1) region after a picture is divided into n regions. W represents the estimated calculation workload contained in a given area, which is a mean value and an ideal value.

\[
W(r_x) = \frac{\text{Comp}_{frame}^{j}}{N}
\]

(3)

After that, the computational complexity of CTU in each area should be compared with \(W(r_x)\) value is as close as possible so that the workload can be evenly distributed to each thread. This design
determines the boundary of each area by the order of raster scanning. Then a normalized key value pair mapping table is generated to store the boundary information of these regions. First of all, starting from the first CTU of an image, the CB block boundary estimation of the luminance component of each quadtree leaf node CU of each CTU is accumulated, that is, the sum of all $\text{Comp}_{CTU}$ from the first CTU to the current CTU is calculated. Then, the accumulated value is normalized by the maximum estimation calculation complexity of CTU. The normalized value is the key of the mapping table, and the corresponding CTU index is the value, so as to form a key value pair. Such key value pair mapping calculation is carried out for each CTU. For example, the key and value of the m-th CTU in frame i in the mapping table are expressed as formula (4) and formula (5). Among them, floor is the down rounding function, 128 is the maximum calculation complexity (corresponding to the filter boundary number of luminance component CB) when 64x64 size CTU is divided into 64 8×8 size CU units, because from table 1, it can be seen that the filter boundary number of 8×8 size CU is 2 no matter whether TU division is carried out or not. In addition, in formula (4), using the right shift operation instead of the division operation can reduce the calculation amount.

$$key = \text{floor} \left( \frac{\sum_{i=0}^{n-1} \text{Comp}_{CTU}^i}{128} \right)$$

$$value = m$$

Through the above scheme, we can balance the amount of filtering calculation in each thread as much as possible. For the BasketballDrive video case, the workload distribution between threads before and after equalization is shown in Figure 2.

3.2 DBF + SAO Joint Parallel Loop Filtering

In the loop filtering, SAO depends on the data after the horizontal boundary vertical filtering in the deblocking filtering. For the SAO processing of grating sequence, before the SAO of all adjacent CTU is completed, the boundary samples of CTU should be stored, that is, for the previously mentioned 45° example, the boundary samples of the top and right CTU need to be stored. Different from the data dependence in the grating sequential Sao process, the parallel SAO processing needs to consider eight directions of data dependence. The decoder should store eight directions of adjacent samples for all CTB to ensure that the data dependence in parallel processing can be met. As shown in Figure 3(a).
According to the conventional filtering order, when processing the compensation of pixels at the boundary of CTU, each thread needs to read the reference points of eight CTU around the boundary of CTU in the system cache. This reading process needs more time, so there will be a large delay between the two links. The local cache of each working thread can be used to store the pixel samples of eight neighboring CTU around each CTU at the boundary. In this way, the pixels inside and outside the boundary of a CTU form two pixel cache lines, as shown in Figure 3(b). After binding the thread to each tile core, the image is divided into multiple regions with equal computation amount according to the CTU partition characteristics of image regions, and these regions are allocated to multiple threads to achieve load balancing. Multiple cores or threads process the pixels in their respective designated areas in parallel. By using the local cache of each filtering thread, the pixel sample points at the boundary of CTU are stored. First, the vertical boundary is filtered horizontally. At the same time, by using the cache, a column of pixel data on both sides of the boundary of the adjacent CTU after filtering is copied and cached to the local cache of the thread, which can be used for boundary mode decision and compensation operation of subsequent SAO.

Figure 3. Data dependency relationship between CTU boundary pixel samples and pixel sample buffer on both sides of adjacent CTU boundary.

Figure 4. Overall timing diagram of loop filtering.

After each thread completes the horizontal filtering of its own area, it performs the vertical filtering. At the same time, it copies and caches a row of pixel data at the upper and lower sides of the edge of the filtered adjacent CTU, which is also used for the subsequent SAO boundary mode decision and compensation. In this way, increase the cache space when the thread performs deblocking filtering. After filtering, access the data in the cache instead of accessing the data in the memory, reducing the internal time delay of the loop filtering module, as shown in Figure 4. In the past, the whole frame
deblocking filtering is required to be completed before SAO can be carried out. However, for the fusion loop filtering scheme proposed in reference [10], there are more redundant data buffers due to the increase of data communication frequency between the two links. In this design, as long as a thread completes the deblocking filtering of the area, it can immediately carry out the SAO of the current area. In this way, when multiple threads are in parallel, the delay between two links can be reduced and redundant data interaction can be reduced.

4. Experiment

4.1. Experiments Design
We implement our algorithm on the Tilera-GX36 multi-core platform, and use libde265 as the baseline implementation. Using BasketballDrive of 1080P, Traffic of 1600P and ParkJoy of 2160P as experimental sequences. The test video streams used in the experiment are all encoded by HEVC software x265 [11,12], and no parallel mode is used. In this experiment, the parallel speedup is used to evaluate the algorithm of this subject. For example, in formula (6), $fps_{parallel}$ and $fps_{sequential}$ represent the frame rates of multi-core parallel decoding and single core serial decoding respectively. FPS can be obtained from formula (7), where $Num_{frame}$ is the number of frames in the video sequence, and $Time$ represents the decoding time.

$$speedup = \frac{fps_{parallel}}{fps_{sequential}}$$  \hspace{1cm} (6)

$$fps = \frac{Num_{frame}}{Time}$$  \hspace{1cm} (7)

4.2. Results and Discussion
Experiments are carried out on the joint parallel loop filtering scheme. The fast fusion loop filter algorithm proposed in [10] is selected as the comparison scheme, and the decoding acceleration ratio under different quantization parameters (QP) is obtained, and the experimental results are analysed.

Figure 5. Comparison of experimental results under four QP values.
Table 2. Parallel acceleration comparison experimental results and comparison.

| QP | Core numbers | BasketballDrive (FILF) | BasketballDrive (CILF) | Traffic (FILF) | Traffic (CILF) |
|----|--------------|------------------------|------------------------|--------------|--------------|
| 8  | 1            | 1                      | 1                      | 1            | 1            |
|    | 6            | 5.63                   | 5.82                   | 5.77         | 5.91         |
|    | 10           | 6.62                   | 7.55                   | 7.04         | 8.08         |
| 22 | 14           | 8.11                   | 9.23                   | 8.45         | 9.57         |
|    | 17           | 9.12                   | 10.21                  | 9.77         | 10.83        |
|    | 24           | 9.81                   | 10.69                  | 10.13        | 11.08        |
|    | 36           | 9.83                   | 10.71                  | 10.15        | 11.09        |
| 27 | 1            | 1                      | 1                      | 1            | 1            |
|    | 6            | 5.21                   | 5.64                   | 5.42         | 5.86         |
|    | 10           | 6.14                   | 7.17                   | 6.62         | 7.56         |
|    | 14           | 7.73                   | 8.92                   | 8.03         | 9.14         |
|    | 17           | 8.74                   | 9.86                   | 9.31         | 10.07        |
|    | 24           | 9.67                   | 10.55                  | 9.82         | 10.73        |
|    | 36           | 9.68                   | 10.56                  | 9.83         | 10.74        |
| 32 | 1            | 1                      | 1                      | 1            | 1            |
|    | 6            | 4.41                   | 4.98                   | 4.76         | 5.13         |
|    | 10           | 5.43                   | 6.59                   | 5.89         | 6.87         |
|    | 14           | 6.94                   | 8.23                   | 7.34         | 8.41         |
|    | 17           | 8.26                   | 9.61                   | 8.83         | 9.98         |
|    | 24           | 9.45                   | 10.33                  | 9.71         | 10.52        |
|    | 36           | 9.46                   | 10.34                  | 9.72         | 10.53        |
| 37 | 1            | 1                      | 1                      | 1            | 1            |
|    | 6            | 4.39                   | 4.57                   | 4.68         | 4.76         |
|    | 10           | 5.12                   | 5.98                   | 5.47         | 6.35         |
|    | 14           | 6.47                   | 7.67                   | 6.98         | 8.06         |
|    | 17           | 7.94                   | 9.32                   | 8.52         | 9.54         |
|    | 24           | 9.18                   | 10.19                  | 9.46         | 10.28        |
|    | 36           | 9.16                   | 10.17                  | 9.44         | 10.27        |

Fast fusion loop filter is a kind of CTU unit which is formed by coupling DBF and SAO, and parallel processing is carried out with this kind of CTU unit as granularity. Two kinds of video sequences, 1080p BasketballDrive and 1600p Traffic, are tested, and the parallel speedup is used as the evaluation standard. The experimental acceleration ratio data are shown in Table 2. Draw the comparison line chart under four QP values, as shown in Figure 5. (a)-(d). It can be seen from the figure that at the beginning, the number of core threads is small, and the parallel acceleration ratio of the proposed scheme is only slightly higher than that of the fast fusion loop filter, so the improvement is not obvious. However, with the increase of the number of core threads, the parallel speedup ratio
increases gradually. When the number of core threads reaches about 24, the parallel speedup ratio of the two schemes reaches the peak.

5. Conclusion
This paper first analyzes the data processing characteristics of deblocking filter and sample adaptive compensation in the loop filter module, and then analyzes the data relationship between them. It is found that the load balancing among the filtering threads is not realized in each research. Therefore, based on theory and problem analysis, a joint parallel loop filtering algorithm based on multithreading load balancing is implemented. Finally, we implement this method on Tilera-GX36 multi-core platform. Experimental results show that the efficiency of the proposed loop filter is improved compared with the previous parallel loop filter algorithm.

References
[1] Sullivan G J, Ohm J R and Wiegand T 2012 Overview of the High Efficiency Video Coding (HEVC) Standard Transactions on Circuits and Systems for Video Technology. TCSVT 22 1649
[2] Shu J and Hu D 2016 A parallel HEVC encoder scheme based on Multi-core platform National Conference on Electrical, Electronics and Computer Engineering. NCEECE (2015) 12 375
[3] Eldeken A F, Fouad M and Salama G I 2015 High throughput parallel scheme for HEVC deblocking filter International Conference on Image Processing. ICIP (2015) 35 1538
[4] Wen S P, Chen M Z Q, Yu X H, Zeng Z G and Huang T W 2017 Fuzzy control for uncertain vehicle active suspension systems via dynamic sliding-mode approach IEEE Transactions on Systems, Man and Cybernetics: Systems 47(1) 24-32
[5] Wen S P, Chen M Z Q, Zeng Z G, Huang T W and Li C J 2017 Adaptive Neural-Fuzzy Sliding-Mode Fault-Tolerant Control for Uncertain Nonlinear Systems IEEE Transactions on Systems, Man and Cybernetics: Systems 47(8) 2268-78
[6] Ikeda M, Tanaka J and Suzuki T 2011 Parallel deblocking filter ITU-T SG16 WP3 14 1755
[7] Li Y, Hu D and Zhang W X 2019 Fine Granular Multilevel Parallel Algorithm for HEVC Decoding Based on Tilera-GX36 Multi-core Platform International Conference on Computer Science and Application Engineering Conference. ICCSAEC (2019) 12 127
[8] Wen S P, Zeng Z G, Chen M Z Q and Huang T W 2017 Synchronization of switched neural networks with communication delays via the event-triggered method IEEE Transactions on Neural Networks and Learning Systems 28(10) 2334-43
[9] Wen S P, Huang T W, Yu X H, Chen M Z Q and Zeng Z G 2016 Aperiodic sampled-data sliding-mode control of fuzzy systems with communication delays via the event-triggered method IEEE Transactions on Fuzzy Systems 24(5) 1048-57
[10] Bossen F and Frank G 2012 Common Test Conditions and Software Reference Configurations JCTVC-H1100 JCT-VC p 2123
[11] Peesapati R, Das S, Baldev S and Ahamed S R 2017 Design of streaming deblocking filter for HEVC decoder Transactions on Consumer Electronics. TCE(2017) 5 1-9
[12] Radicke S 2014 A Multi-Threaded Full-feature HEVC Encoder Based on Wavefront Parallel Processing International Conference on Signal Processing and Multimedia Applications. ICSPMA(2014) 16 90