Parallel Deblocking Filter Based on Modified Order of Accessing the Coding Tree Units for HEVC on Multicore Processor

Haiwei Lei[1], Wenyi Liu[1] and Anhong Wang[2]
[1] Key Laboratory of Instrumentation Science & Dynamic Measurement, Ministry of Education, North University of China
Taiyuan, 030051- China
[e-mail: lhw0312@163.com; liuwenyi@nuc.edu.cn]
[2] School of Electronic Information Engineering, Taiyuan University of Science and Technology
Taiyuan, 030024 - China
[e-mail: wah_ty@163.com]
*Corresponding author: Wenyi Liu; Anhong Wang

Received April 11, 2016; revised October 18, 2016; revised December 12, 2016; accepted January 15, 2017; published March 31, 2017

Abstract

The deblocking filter (DF) reduces blocking artifacts in encoded video sequences, and thereby significantly improves the subjective and objective quality of videos. Statistics show that the DF accounts for 5–18% of the total decoding time in high-efficiency video coding. Therefore, speeding up the DF will improve codec performance, especially for the decoder. In view of the rapid development of multicore technology, we propose a parallel DF scheme based on a modified order of accessing the coding tree units (CTUs) by analyzing the data dependencies between adjacent CTUs. This enables the DF to run in parallel, providing accelerated performance and more flexibility in the degree of parallelism, as well as finer parallel granularity. We additionally solve the problems of variable privatization and thread synchronization in the parallelization of the DF. Finally, the DF module is parallelized based on the HM16.1 reference software using OpenMP technology. The acceleration performance is experimentally tested under various numbers of cores, and the results show that the proposed scheme is very effective at speeding up the DF.

Keywords: Deblocking filter, parallel programming, multicore processor, high-efficiency video coding (HEVC)

This work is supported in part by the National Natural Science Foundation of China (No. 61272262 and 61210006), the Program for New Century Excellent Talent in Universities (NCET-12-1037), the International Cooperative Program of Shanxi Province (No. 2015081015).

https://doi.org/10.3837/tiis.2017.03.024 **ISSN : 1976-7277**

1. Introduction

High-efficiency video coding (HEVC) is a relatively new video coding standard developed by the Joint Collaborative Team on Video Coding. It achieves a higher compression efficiency and better video quality than the previous standards, and is better for encoding high-definition (HD) and ultra-HD video. Compared with H.264, at the same quality of coding, HEVC reduces the bitrate by approximately 50%, thus doubling the coding efficiency [1]. The high performance of HEVC entails high computational complexity in the codec [2]. This poses a significant challenge to the processor in performing real-time encoding and decoding.

Similar to previous video coding standards, HEVC is based on a hybrid coding scheme that uses block-based prediction and transform coding [3]. As shown in **Fig. 1**, an image frame is partitioned into multiple coding tree units (CTUs). Each CTU is recursively partitioned into coding units (CUs) using a quadtree structure. Moreover, each CU is partitioned into prediction units (PUs) and transform units (TUs) in different ways. Each of the final blocks produced by partitioning is independently encoded. Consequently, videos encoded using the HEVC standard may suffer from blocking artifacts, ringing, color deviations, and image blurring.

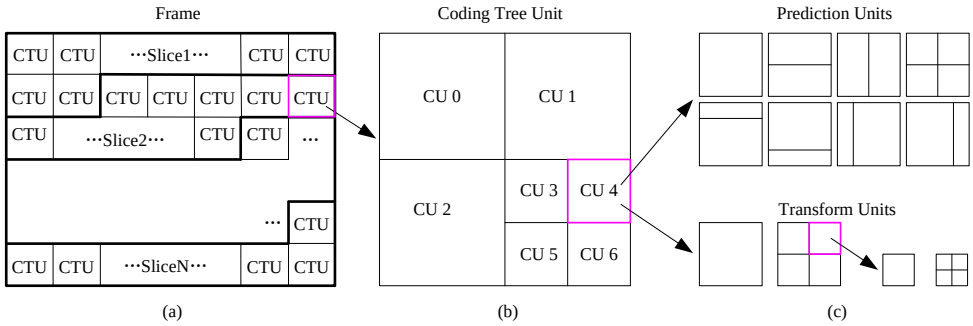

Fig. 1. Block-based coding scheme: (a) CTUs; (b) CUs; and (c) subdivision of CUs (PUs and TUs)

Blocking artifacts are discontinuities caused by encoding at the block boundary of the reconstructed picture, and seriously affect the subjective quality of videos. These artifacts are primarily a result of the independence between the transformation, quantization, and encoding processes of each block. That is, adjacent blocks use different intra-prediction modes and different quantization parameters, or the prediction values of adjacent blocks are derived from different positions of different reference pictures in motion-compensated prediction.

To reduce the impact of blocking artifacts on video quality, various filtering techniques have been developed. In HEVC, the in-loop deblocking filter (DF) is adopted to reduce the presence of blocking artifacts in the reconstructed pictures. To further improve the quality of subsequent encoding, the filtered pictures can be used as references in time-domain prediction. This technology can adaptively select filtering parameters with different intensities according to different video content and encoding parameters. The DF then smooths the reconstructed picture based on these parameters. DF technology reduces the average bitrate by 1.3–3.3%, and the maximum bitrate by more than 6% for specific sequences [4]. Furthermore, the DF significantly improves the visual quality of the pictures.

The DF is applied to both the encoder and decoder sides. Statistical data show that it accounts for 5–18% of the total decoding time [28]; hence, speeding up the DF will improve codec performance, especially for the decoder. With the development of multicore processor technology, the use of parallel processing to improve the encoding and decoding speeds in multicore environments has become a popular area of research. The DF module is relatively independent of the other HEVC modules, and is therefore suitable for parallel processing. Consequently, parallel operation is a good choice for speeding up the DF.

In recent years, there have been numerous studies on the use of parallel technology to speed up all aspects of the video encoding and decoding process. In [5], a parallel implementation of H.264 decoding was introduced using an embedded multicore processor. In [6-8], the encoding process of H.264 was optimized using a graphics processor, whereas [9] reported an open platform for the design of parallel H.264 video encoders. In [10-12], the parallel optimization of motion estimation for H.264 and HEVC was proposed, and [13] described an efficient parallel HEVC intra prediction scheme. The authors of [14] proposed a highly parallel framework for the CU partitioning tree decisions in HEVC.

Various parallel optimization schemes for the DF have also been reported. For example, [15-20] focused on the parallelization of the H.264 DF. For the HEVC DF, a two-step filtering scheme has been proposed [21] in which horizontal filtering is first performed across all vertical boundaries for all CTUs in the processing frame. Vertical filtering is then performed across all horizontal boundaries. This scheme is easily capable of performing horizontal and vertical filtering in parallel. A parallel DF in a directed acyclic graph-based order was proposed and shown to exhibit good performance on a Tile64 experimental platform [22]. A heterogeneous platform consisting of CPUs and GPUs has been used to speed up the DF process [23], allowing the advantages of streaming multiprocessors in GPUs to be fully implemented. In [24], three different parallel DF implementations were compared. The first was a separate filtering method in which horizontal and vertical filtering were successively performed in parallel for one frame. The other two implementations were combined filtering methods in which concurrent threads had to be synchronized to complete the filtering. In addition, hardware architecture implementations of the DF have been reported [25-29].

Despite the contributions of the above-mentioned parallel schemes based on multicore platforms, several limitations remain. For example, there is the high communication overhead among threads, coarse parallel granularity, and the lack of adaptation to different resolution sequences and processors with different numbers of cores. In this paper, by analyzing the data dependencies between adjacent CTUs, we develop a parallel scheme based on a modified order of accessing the CTUs so as to speed up the DF process. In our approach, the filtering task is divided into several subtasks of equal size based on the number of threads. Multiple threads then simultaneously perform horizontal filtering in accordance with the order of rows. After synchronization, they perform vertical filtering according to the order of the columns. In this way, multiple threads can simultaneously perform filtering on different CTUs without data access conflicts. Thus, the DF module is speeded up, and the entire codec performance is improved.

The contributions of this paper are as follows. First, we propose a conflict-free parallel DF scheme based on the results of a data-dependency analysis, which significantly speeds up the process of the DF. This scheme has two advantages. The flexible degree of parallelism (DOP) makes it adaptive to different resolution sequences and processors with different capacities, and finer parallel granularity allows for better load balancing in the execution of threads. Second, the privatization of variables and synchronization of threads, which are typical problems related to parallel programming, are solved, enabling the DF to run correctly in

parallel. Third, after removing a number of bugs caused by parallelization, the proposed parallel scheme is ultimately implemented using the OpenMP technology. The acceleration performance is then tested under different numbers of cores.

The remainder of this paper is organized as follows. In Section 2, we review the process of the DF. Section 3 then introduces the proposed parallel DF scheme. In Section 4, the problems of variable privatization and thread synchronization are solved. Experimental results are presented in Section 5, and our conclusions are given in Section 6.

2. HEVC Deblocking Filter

Each step of the DF is involved in the process of parallelization. Therefore, this section presents a review of the four steps of the DF.

2.1 Determining the Block Boundaries

The DF reduces blocking artifacts by smoothing samples on each side of the boundaries, where the block boundaries come from the PUs and TUs, rather than the inherent boundaries of the picture. Moreover, unlike H.264, in which the DF is applied to all 4×4 block boundaries of a picture, the HEVC DF addresses the boundaries of the PUs and TUs using an 8×8 block, as shown in **Fig. 2**. Only the boundaries (V_1,V_2,V_3 and H_1) of the 8×8 grid are subjected to filtering; P and Q are 4×4 sample blocks on each side of the boundary that represent the samples to be filtered at a given time.

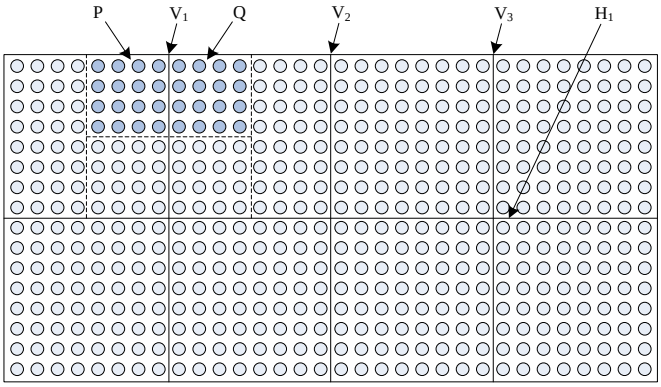

Fig. 2. Block boundaries for the DF

2.2 Calculating the Boundary Strength

The boundary strength (BS) is calculated to determine whether the block boundary is to be filtered and the filtering parameters according to the encoding parameters of blocks P and Q. The BS is calculated for all boundaries that need to be filtered; the BS takes values of 0, 1, or 2. The BS calculation process is depicted in **Fig. 3**.

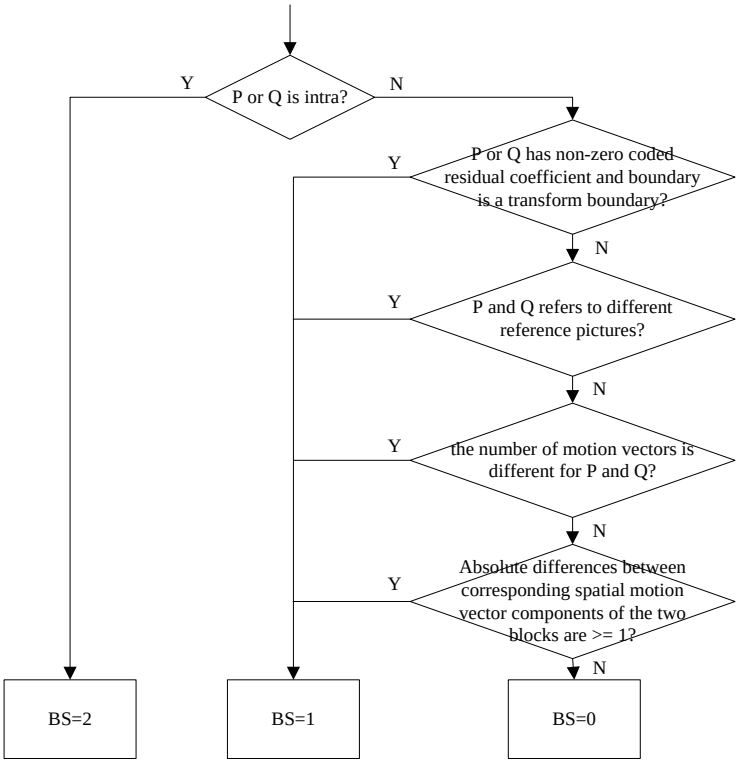

Fig. 3. Flowchart for calculating the BS

2.3 Filtering Decision

The filtering decision ultimately determines whether the boundary must be filtered and the filtering strength. A filtering region containing a vertical boundary is shown in **Fig. 4**, where $p_{x,y}$ and $q_{x,y}$ are sample values on each side of the block boundary.

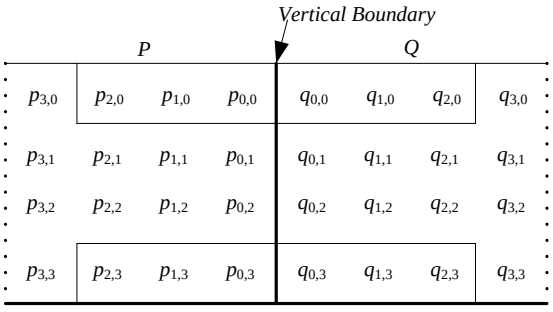

Fig. 4. A filtering region containing a vertical boundary

As specified in HEVC, if the BS is greater than zero and Inequality (1) is satisfied, the DF is applied; otherwise, the DF is not applied.

$$\left|p_{2,0}-2\,p_{1,0}+p_{0,0}\right|+\left|q_{2,0}-2q_{1,0}+q_{0,0}\right|+\left|p_{2,3}-2\,p_{1,3}+p_{0,3}\right|+\left|q_{2,3}-2q_{1,3}+q_{0,3}\right|<\beta \quad (1)$$

where β is a decision threshold relating to the quantization parameters of blocks P and Q (QP_P, QP_Q) as well as the slice-level compensation value (*slice_beta_offset_div2*).

HEVC has two DF modes. Therefore, before filtering, either normal or strong filtering must be selected according to the following expressions. If Inequalities (2)–(7) hold, strong filtering is applied; otherwise, normal filtering is applied.

$$\left|p_{2,0}-2\,p_{1,0}+p_{0,0}\right|+\left|q_{2,0}-2q_{1,0}+q_{0,0}\right|<\beta/8 \quad (2)$$

$$\left|p_{2,3}-2\,p_{1,3}+p_{0,3}\right|+\left|q_{2,3}-2q_{1,3}+q_{0,3}\right|<\beta/8 \quad (3)$$

$$\left|p_{3,0}-p_{0,0}\right|+\left|q_{0,0}-q_{3,0}\right|<\beta/8 \quad (4)$$

$$\left|p_{3,3}-p_{0,3}\right|+\left|q_{0,3}-q_{3,3}\right|<\beta/8 \quad (5)$$

$$\left|p_{0,0}-q_{0,0}\right|<2.5t_C \quad (6)$$

$$\left|p_{0,3}-q_{0,3}\right|<2.5t_C \quad (7)$$

where t_C is a decision threshold related to BS, QP_P, QP_Q, and the slice-level compensation value (*slice_tc_offset_div2*).

The filtering decision process is summarized in **Fig. 5**.

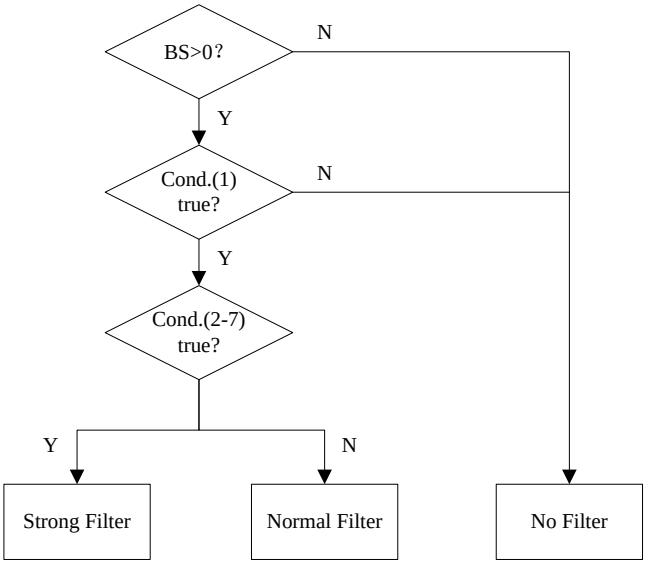

Fig. 5. Flowchart of the filtering decision

2.4 Filtering Operation

The filtering operation is the final step of the DF process, and is responsible for smoothing the

samples on either side of the block boundary. The number of samples to be modified will vary according to the filtering condition. Under normal filtering, if Inequality (8) holds, the two samples closest to the block boundary (p_0 and p_1) will be modified in block P. Otherwise, only the nearest sample (p_0) in block P will be modified. A similar process is applied to block Q in accordance with Inequality (9). When strong filtering is applied, three samples on either side of the block boundary will be modified (p_0, p_1, p_2 in block P and q_0, q_1, q_2 in block Q). Details can be found in [4-5].

$$\left|p_{2,0}-2p_{1,0}+p_{0,0}\right|+\left|p_{2,3}-2p_{1,3}+p_{0,3}\right|<3/16\beta \tag{8}$$

$$\left|q_{2,0}-2q_{1,0}+q_{0,0}\right|+\left|q_{2,3}-2q_{1,3}+q_{0,3}\right|<3/16\beta \tag{9}$$

3. Proposed Parallel Scheme for Deblocking Filter

In this section, a parallel DF scheme based on a modified order of accessing the CTUs is proposed. The modified order is determined by analyzing the data dependencies between adjacent CTUs.

3.1 Data Dependency Analysis

Data dependencies are not conducive to the parallelization of a program, as they cause data access conflicts. Hence, to parallelize the DF, we must first analyze the potential data dependencies.

Two important data items are often accessed during the DF process: the tentatively named TComDataCU and PicYuvRec. The former saves partition information about the CTUs (the partition of PUs and TUs), prediction mode information, and so on, which is mainly accessed when calculating the BS and making a filtering decision. The latter stores the reconstructed picture, which can be regarded as the image given by the original picture through transformation, quantization, inverse quantization, and inverse transformation in the encoding process. It can also be regarded as a decoded and unfiltered picture from the stream file in the decoding process. This data item is mainly accessed in the filtering operation stage.

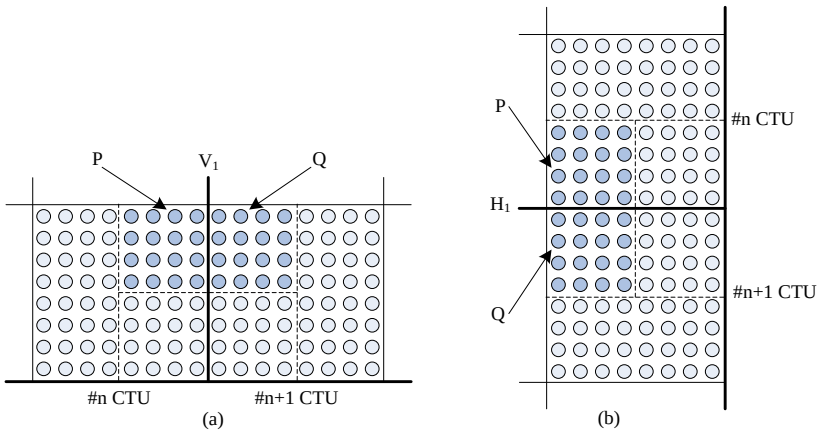

Fig. 6. Special cases for: (a) horizontal filtering; (b) vertical filtering

Data dependency occurs in special cases of horizontal filtering and vertical filtering, namely, when the boundary to be filtered is located between adjacent CTUs, as shown in **Fig. 6**. In this case, the BS value of V_1 (or H_1) is calculated by reading the prediction modes of blocks P and Q according to the specification shown in **Fig. 3**. In other words, both $TComDataCU_{n+1}$ of CTU #n+1 and $TComDataCU_n$ of CTU #n must be accessed. The same situation occurs in the filtering decision stage according to the specification shown in **Fig. 5**. Thus, when horizontal filtering is performed for CTU #n+1, access to the current CTU is required, and access to the left CTU may be needed. Similarly, when vertical filtering occurs, access to the upper CTU may be needed, in addition to the necessary access to the current CTU. This scenario indicates that some data dependency will persist in the DF process, even though the method proposed in [21] is used.

Data access conflicts may appear when the DF module runs in parallel, as multiple threads will access different CTUs simultaneously. Suppose that, at a specific time, $thread_{n+1}$ calculates the BS for V_1 in CTU #n+1 and $thread_n$ calculates the BS for a certain boundary in CTU #n. This will cause a data access conflict, because $thread_{n+1}$ must access $TComDataCU_n$ of CTU #n while it is occupied by $thread_n$.

3.2 Proposed Parallel Scheme

The above analysis causes us to believe that data access conflicts in the parallel DF can be avoided by modifying the order in which CTUs are accessed. That is, for a single frame, horizontal filtering is first performed with the CTUs accessed according to the row order. Vertical filtering is then conducted with the CTUs accessed according to the column order. This access order is more parallel-friendly than that in the method proposed in [21].

Based on the above access order, we propose a parallel DF scheme as follows. The filtering task is divided into several subtasks of equal size in CTU based on the number of threads. Multiple threads are then simultaneously started to perform horizontal filtering from different locations of the frame in accordance with the row order. After synchronization, the threads perform vertical filtering according to the column order. **Fig. 7** illustrates this procedure in a quad-core case. First, four threads perform horizontal filtering in accordance with the order of the rows. Then, they perform vertical filtering according to the order of the columns. In this case, no data access conflicts occur among the four concurrent threads; therefore, all four cores can be effectively used.

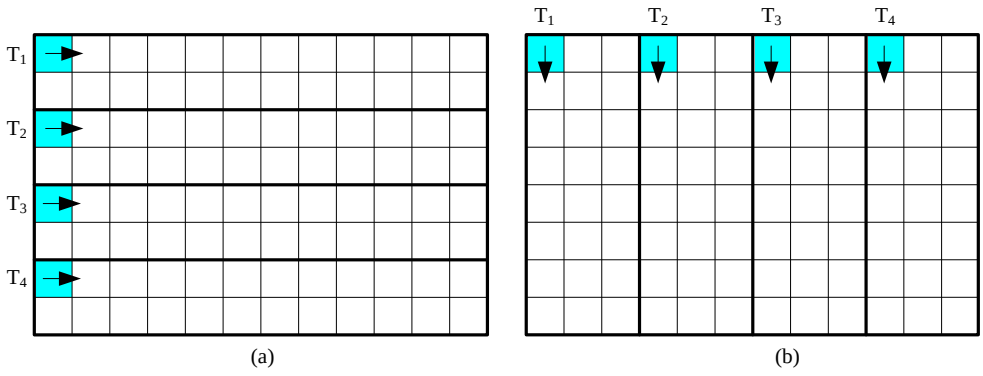

Fig. 7. Proposed parallel DF in a quad-core case: (a) parallel horizontal filtering; (b) parallel vertical filtering

In our scheme, the number of threads may not be the same as the number of rows. **Fig. 8(a)** shows the case in which the number of threads is less than the number of rows, whereas in **Fig. 8(b)** the number of threads is greater than the number of rows. This thread allocation strategy has two advantages. The first is the flexible DOP, which enables adaptation to different resolution sequences and processors with different numbers of cores (i.e., dual-core, quad-core, or octa-core). The second advantage is the finer parallel granularity, namely, the filtering of a single CTU is treated as an atomic task and assigned to a thread. This achieves better load balancing than the scheme proposed in [24], where a whole row is assigned to a thread.

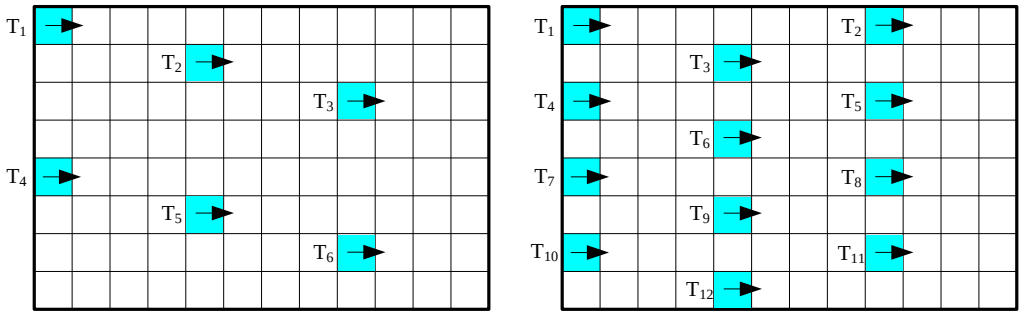

Fig. 8. Parallel horizontal filtering with different numbers of threads: (a) less than the number of rows, (b) greater than the number of rows

4. Problems Associated with the Parallel Deblocking Filter

In addition to data dependencies, several typical problems related to parallel programming need to be solved to ensure the parallel DF runs correctly. This section discusses two common issues, namely the privatization of variables and the synchronization of threads in the parallelization of the DF.

4.1 Variable Privatization

In the DF module, there are two shared variables that need to be privatized. One stores the boundary information of the CTU, and the other stores the BS value. The vertical and horizontal boundary information is stored in a two-dimensional array of length 256 (when the size of the CTU is 64×64). The value type is Boolean; initially, all elements in the array are zero (false). After boundary identification, elements for which a boundary exists are assigned a value of one (true). An example is depicted in **Fig. 9**.

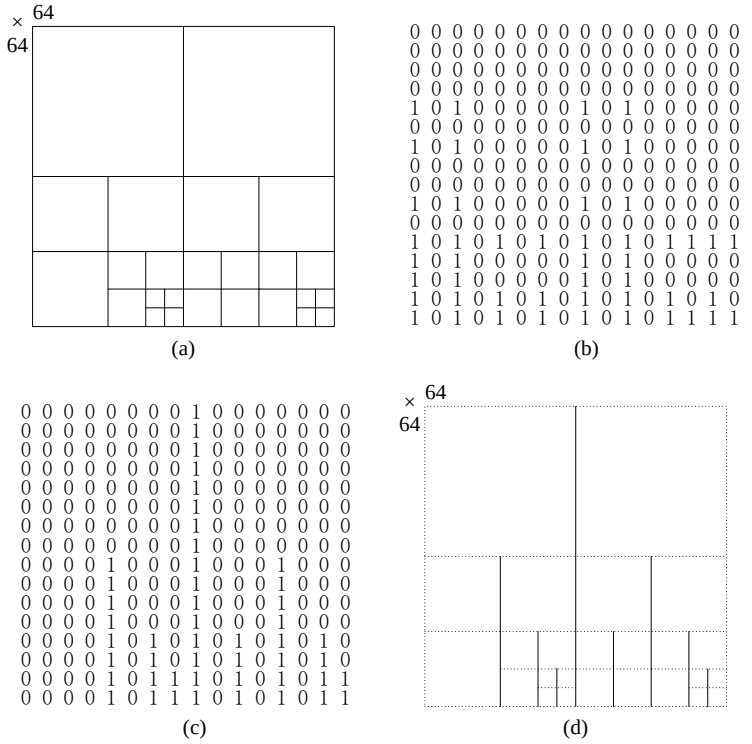

Fig. 9. Illustration of the array used to store the vertical boundary information: (a) CTU and its boundary information, where the solid line represents the partition boundaries of the PUs and TUs; (b) an array that stores vertical boundary information with elements in the Z scanning order; (c) an array with elements in raster scan order converted by the Z scanning order; and (d) the corresponding vertical boundaries of the array

When the DF is applied in non-parallel mode, there is a single main thread. Therefore, the CTUs in the frame can only be handled one at a time. However, when the parallel DF is applied, multiple CTUs are processed by multiple threads simultaneously. It is impossible to store the boundary information of multiple CTUs using only one array. Therefore, a thread-private array must be maintained to save the boundary information of the CTUs for each thread. By default, the variables are shared by all threads; thus, the "threadprivate" directive is added in the parallel DF version to make the array thread-private. Similarly, another array exists to store the BS. This array must also be privatized.

4.2 Thread Synchronization

A second problem to be solved in the parallelization of the DF is thread synchronization. For each frame, vertical filtering begins after the completion of horizontal filtering. Thus, the parallelization of the filtering is divided into two stages: parallel horizontal filtering and parallel vertical filtering. Multiple concurrent threads must be synchronized once the parallel horizontal filtering has been completed. These threads are then concurrently started for the vertical filtering, as shown in **Fig. 10**. In our solution, an implicit barrier method is adopted to synchronize these threads.

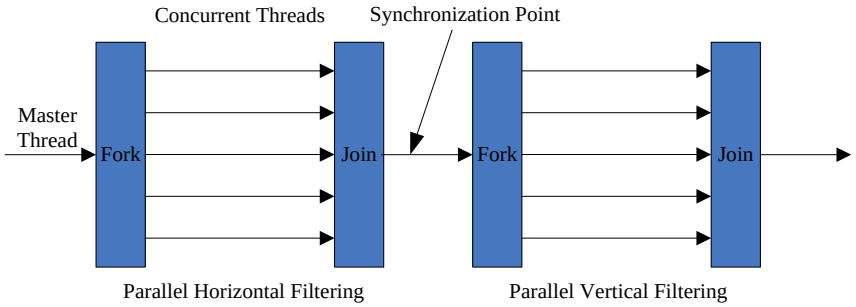

Fig. 10. Illustration of thread synchronization

5. Experiments and Performance Evaluation

This section describes three experiments. First, the filtering effect of the proposed parallel DF is compared with that of the non-parallel (original) DF to show that the proposed scheme does not adversely affect the filtering. Second, the acceleration performance of the proposed parallel DF is tested for various numbers of cores. In each case, the speedup ratio is calculated and the ability to adapt to different resolution sequences and core numbers is verified implicitly. Third, the acceleration performance of the proposed scheme is compared with that of SeparateFiltering [24] to demonstrate the potential of our scheme, especially for load balancing.

5.1 Experimental Setting

The DF module was parallelized using the OpenMP [30]. After fixing various bugs caused by parallelization, the evaluations were conducted using HM16.1 (the reference software for HEVC) [31] and a server with an Intel® Core™ Xeon E-5 2680 CPU at 2.8 GHz. The test sequences recommended by JCT-VC were classified into six class (Class A–F) according to the picture size and their applications, and six of them were selected (one taken from each of six class) for the tests. To facilitate the comparison of statistics, only the first ten frames of the sequences were tested under all intra (AI) configurations [32].

5.2 Experimental Results

We evaluated the performance of the parallel DF from two aspects: (1) filtering effect, and (2) speedup ratio.

The filtering effect was determined indirectly by investigating the peak signal-to-noise ratio (PSNR) and the structural similarity (SSIM) of the encoded sequence. These two metrics are used to indicate the objective and subjective video quality, respectively. Specifically, the same sequence was encoded twice, one through non-parallel filtering, and another through parallel filtering. The PSNR and the SSIM of the two coded sequences were collected. **Table 1** lists the PSNR and SSIM results about the six sequences. It is evident that both the PSNR and SSIM perform the same, which implies that the parallelization of the DF does not affect the filtering performance.

Table 1. Comparison of encoding performance

Class	Sequences	Resolution	Non-parallel		Parallel	
			PSNR	SSIM	PSNR	SSIM
A	Traffic	2560×1600	37.71	0.94	37.71	0.94
B	BasketballDrive	1920×1080	39.01	0.91	39.01	0.91
C	BasketballDrill	832×480	36.37	0.89	36.37	0.89
D	BasketballPass	416×240	36.50	0.91	36.50	0.91
E	FourPeople	1280×720	39.47	0.96	39.47	0.96
F	ChinaSpeed	1024×768	38.38	0.96	38.38	0.96

The speedup ratio was measured using the filtering time, namely, the execution time of the filtering process. Similar to the previous evaluation, the same sequence went through the processes of non-parallel filtering and parallel filtering. The non-parallel DF was executed firstly and the filtering time for Class A–F were plotted in **Fig. 11**, in which the horizontal and vertical axes stand for the frame number and filtering time, respectively. As expected, the filtering time increases with the sequence resolution. Moreover, the filtering time of each frame is approximately equal due to the use of the AI configuration. To ensure the accuracy of evaluation, the filtering time of the ten frames was averaged and the mean will be used for calculating the speedup ratio.

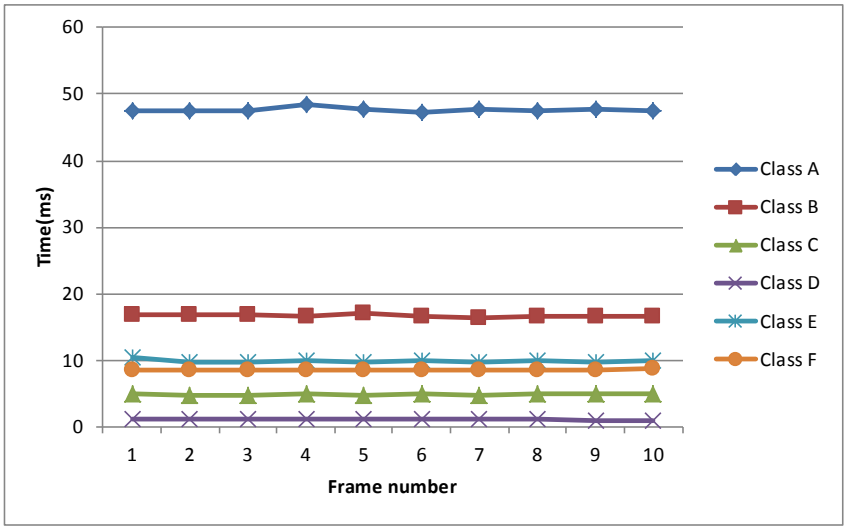

Fig. 11. Filtering time of the first ten frames of different test sequences

Then, the parallel DF was conducted with 2, 4, 6, 8, 10 cores respectively. The execution time was recorded and the speedup ratio can be calculated accordingly, i.e., the ratio of execution times of non-parallel and parallel DF. The speedup ratios for different numbers of cores are shown in **Fig. 12**. As shown in this figure, the DF performance improved significantly while parallelizing, and the speedup ratio increases as the growth of number of cores. However, the speedup is not as linear as expected (the trend is clearly shown in **Fig. 13**), which is consistent with Amdahl's law [33].

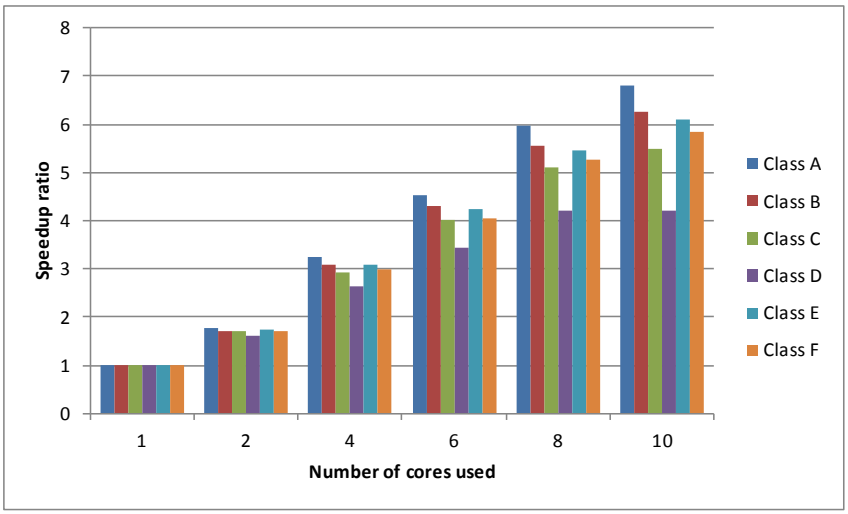

Fig. 12. Speedup ratio obtained after parallelization for different numbers of cores

It can also be observed that, for a given number of cores, the speedup obtained for sequences of different resolution varies slightly. In particular, the speedup of the Class D sequence shows no improvement when going from eight to ten cores, because it is not worthwhile for too many threads to handle such a low-resolution sequence. In fact, when the program runs in parallel, the total time spent includes both the program runtime and the time spent starting and recycling additional threads.

We also evaluated our parallel scheme by comparing its acceleration performance with that of the SeparateFiltering scheme proposed in [24]. Both schemes adopt a two-step filtering method. Test sequences of 1080p and 1600p were used as inputs, and the comparative experimental results are shown in **Fig. 13**. It can be seen that our parallel scheme achieves better acceleration performance, especially with more than six cores. This improvement is because our scheme effectively prevents data access conflicts and achieves better load balancing.

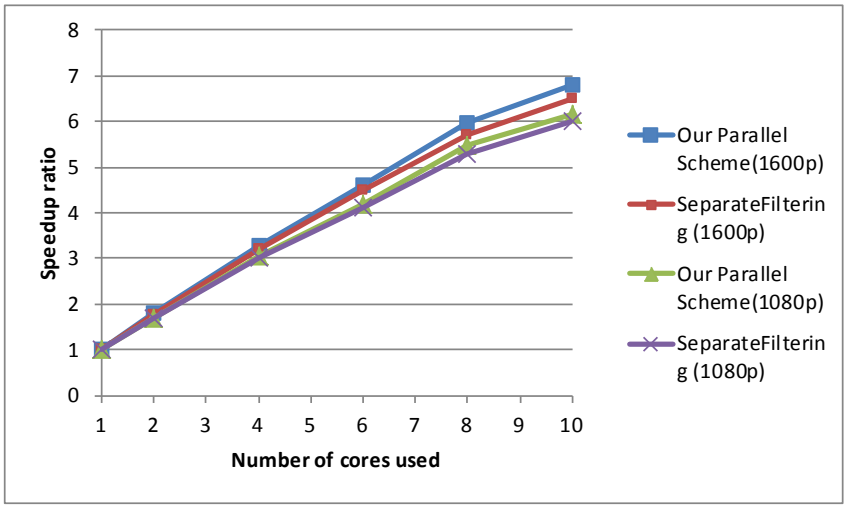

Fig. 13. Comparison of our parallel scheme and the SeparateFiltering scheme

6. Conclusion

In this paper, we proposed a parallel scheme to speed up the HEVC DF on a multicore processor. This scheme is based on a modified order of accessing the CTUs, and enables the DF to run rapidly in parallel without data access conflicts among multiple concurrent threads, as well as offering more flexibility in DOP and finer parallel granularity. After various bugs had been corrected and the problems of variable privatization and thread synchronization had been solved, the proposed scheme was implemented using OpenMP technology and tested on a multicore processor. Experimental results showed that the proposed scheme significantly speeds up the DF compared with the non-parallel DF without any loss of filtering effect and outperforms a similar parallel scheme.

References

[1] J. R. Ohm, G. J. Sullivan, H. Schwarz, T. Thiow Keng, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards-Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669-1684, 2012. Article(CrossRefLink).

[2] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC Complexity and Implementation Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1685-1696, 2012. Article (CrossRef Link).

[3] G. J. Sullivan, J. R. Ohm, H. Woo-Jin, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649-1668, 2012. Article(CrossRefLink).

[4] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, *et al.*, "HEVC Deblocking Filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1746-1754, 2012. Article(CrossRefLink).

[5] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, et al., *High Performance Embedded Architectures and Compilers*, Springer-Verlag Berlin Heidelberg, New York, 2009. Article(CrossRefLink).

[6] W. Nan, W. Mei, S. Huayou, R. Ju, and Z. Chunyuan, "A Parallel H.264 Encoder with CUDA: Mapping and Evaluation," in *Proc. of 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)* , pp. 276-283, 2012. Article(CrossRefLink).

[7] R. Husemann, V. Roesler, J. V. Lima, and M. Gobbi, "Evaluation of CUDA GPU architecture as H.264 intra coding acceleration engine," in *Proc. of the 19th Brazilian symposium on Multimedia and the web*, pp. 177-180, 2013. Article(CrossRefLink).

[8] E. Baaklini, H. Sbeity, and S. Niar, "H.264 parallel optimization on graphics processors," in *Proc. of 5th International Conferences on Advances in Multimedia*, pp. 109-114, April 21-26, 2013. Article(CrossRefLink)

[9] A. Rodrigues, N. Roma, and L. Sousa, "p264: Open platform for designing parallel H.264/AVC video encoders on multi-core systems," in *Proc. of 20th ACM Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 81-86, June 2- 4, 2010. Article(CrossRefLink).

[10] C. Yan, Y. Zhang, J. Xu, F. Dai, J. Zhang, Q. Dai, *et al.*, "Efficient Parallel Framework for HEVC Motion Estimation on Many-Core Processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 12, pp. 2077-2089, 2014. Article(CrossRefLink).

[11] W. Zhu, J. Pan, H. Guo, and W. Sun, "Parallel optimization of motion estimation for video coding on cell BE processors," in *Proc. of 2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)* , pp. 1-6, 2014. Article(CrossRefLink).

[12] B. Wang, M. Alvarez-Mesa, C. C. Chi, and B. Juurlink, "Parallel H.264/AVC Motion Compensation for GPUs Using OpenCL," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 3, pp. 525-531, 2015. Article(CrossRefLink).

[13] C. Yan, Y. Zhang, F. Dai, J. Zhang, L. Li, and Q. Dai, "Efficient parallel HEVC intra-prediction on many-core processor," *Electronics Letters*, vol. 50, no.11, pp. 805-806, 2014. Article(CrossRefLink).
[14] C. Yan, Y. Zhang, J. Xu, F. Dai, L. Li, Q. Dai, *et al.*, "A Highly Parallel Framework for HEVC Coding Unit Partitioning Tree Decision on Many-core Processors," *IEEE Signal Processing Letters*, vol. 21, no. 5, pp. 573-576, 2014. Article(CrossRefLink).
[15] S. Vijay, C. Chakrabarti, and L. J. Karam, "Parallel deblocking filter for H.264 AVC/SVC," in *Proc. of 2010 IEEE Workshop on Signal Processing Systems (SIPS)*, pp. 116-121, 2010. Article(CrossRefLink).
[16] M. Kthiri, P. Kadionik, L. H, x00E, vi, H. Loukil, *et al.*, "A parallel hardware architecture of deblocking filter in H264/AVC," in *Proc. of 2010 9th International Symposium on Electronics and Telecommunications (ISETC)*, pp. 341-344, 2010. Article(CrossRefLink).
[17] B. Pieters, C. F. J. Hollemeersch, J. D. Cock, P. Lambert, W. D. Neve, and R. V. d. Walle, "Parallel Deblocking Filtering in MPEG-4 AVC/H.264 on Massively Parallel Architectures," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 1, pp. 96-100, 2011. Article(CrossRefLink).
[18] Y. Zhang, C. Yan, F. Dai, and Y. Ma, "Efficient Parallel Framework for H.264/AVC Deblocking Filter on Many-Core Platform," *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 510-524, 2012. Article(CrossRefLink).
[19] B. Pieters, C. Hollemeersch, J. D. Cock, W. D. Neve, P. Lambert, and R. V. d. Walle, "Parallel deblocking filtering in H.264/AVC using multiple CPUs and GPUs," in *Proc. of the 20th ACM international conference on Multimedia*, pp.1013-1016, 2012. Article(CrossRefLink).
[20] D. P. Prasad, S. Sonachalam, M. K. Kunchamwar, and N. R. Gunupudi, "Parallel processing architecture for H.264 deblocking filter on multi-core platforms," *Image Processing: Algorithms and Systems X; and Parallel Processing for Imaging Applications II*, pp. 829512-829512-10, 2012. Article(CrossRefLink).
[21] M. Ikeda, J. Tanaka, and T. Suzuki, "Parallel deblocking filter," JCTVC-D263, JCT-VC, Daegu, Kr, 2011.
[22] C. Yan, Y. Zhang, F. Dai, J. Zhang, L. Li, and Q. Dai, "Parallel deblocking filter for HEVC on many-core processor," *Electronics Letters*, vol. 50, no. 5, pp. 367-368, 2014. Article(CrossRefLink).
[23] D. F. d. Souza, N. Roma, and L. Sousa, "Cooperative CPU+GPU deblocking filter parallelization for high performance HEVC video codecs," in *Proc. of 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4993-4997, 2014. Article(CrossRefLink).
[24] A. M. Kotra, M. Raulet, and O. Deforges, "Comparison of different parallel implementations for deblocking filter of HEVC," in *Proc. of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2721-2725 2013. Article(CrossRefLink).
[25] W. Shen, Q. Shang, S. Shen, Y. Fan, and X. Zeng, "A high-throughput VLSI architecture for deblocking filter in HEVC," in *Proc. of 2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 673-676 , 2013. Article(CrossRefLink).
[26] E. Ozcan, Y. Adibelli, and I. Hamzaoglu, "A high performance deblocking filter hardware for High Efficiency Video Coding," in *Proc. of 2013 23rd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1-4, 2013. Article(CrossRefLink).
[27] W. Cheng, Y. Fan, Y. Lu, Y. Jin, and X. Zeng, "A high-throughput HEVC deblocking filter VLSI architecture for 8kx4k application," in *Proc. of 2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 605-608, 2015. Article(CrossRefLink).
[28] C. M. Diniz, M. Shafique, F. V. Dalcin, S. Bampi, and J. Henkel, "A deblocking filter hardware architecture for the high efficiency video coding standard," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1509-1514, 2015. Article(CrossRefLink).
[29] I. Hautala, J. Boutellier, J. Hannuksela, O. Silv, and x00E, "Programmable Low-Power Multicore Coprocessor Architecture for HEVC/H.265 In-Loop Filtering," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 7, pp. 1217-1230, 2015. Article(CrossRefLink).

[30] OpenMP. OpenMP Specifications [Online]. Available: http://openmp.org/wp/openmp-specifications/
[31] JCT-VC. Subversion repository for the HEVC test model version HM-16.1 [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.1/
[32] F. Bossen, "Common Test Conditions and software reference configurations," JCTVC-J1100, JCT-VC, Stockholm, SE, 2012.
[33] Wikipedia. Amdahl's law [Online]. Available: https://en.wikipedia.org/wiki/Amdahl%27s_law

Haiwei Lei received the B.S. degree in computer software and the M.S. degree in electronic science and technology from the North University of China, Shanxi, China, in 2002 and 2008, respectively, where he is currently working toward the Ph.D. degree. Since 2003, he has been engage in teaching and research work in the Department of Computer Science and Technology, North University of China. His research interests include video signal processing, 3-D video coding, compressed sensing, and 3-D measurement.

Wenli Liu received the B.S. degree in safety engineering, the M.S. degree in computer software, and the Ph.D. degree in instrument science and technology from the North University of China, Shanxi, China, in 1992, 1998, and 2009, respectively. Since 2009, he has been a Professor. He served as a Visiting Scientist with the College of Engineering, Design and Physical Sciences at Brunel University, UK, in 2013. His research interests include digital signal processing, compression coding, storage testing, and embedded systems.

Anhong Wang was born in Shanxi Province, P. R. China in 1972. She received B.E. degree and M.E. degree respectively in 1994 and 2002, both in Electronic Information Engineering of Taiyuan University of Science and Technology (TYUST). She received the Ph. D. degree in the Institute of Information Science, Beijing Jiaotong University in 2009. She became an associate professor with TYUST in 2005 and became a full professor in 2009. She is now the director of Institute of Digital Media and Communication, TYUST. Her research interest includes image video coding and transmission, compressed sensing, and secret image sharing. She has published more than 90 papers in international journals and conferences. Now she is leading several research projects, including two National Science Foundations of China.