Parallel In-Loop Filtering in HEVC Encoder on GPU

Yang Wang, Xun Guo, *Member, IEEE*, Xiaopeng Fan, *Senior Member, IEEE*, Yan Lu, Debin Zhao, *Member, IEEE*, and Wen Gao, *Fellow, IEEE*

***Abstract*—In-loop filtering is an important part of high efficiency video coding (HEVC), which consists of deblocking filter and sample adaptive offset (SAO) filter. It can not only improve the compression efficiency of HEVC, but also improve the visual quality of the reconstructed videos significantly. However, the high computational complexity hampers its applications for real-time encoding scenarios. In this paper, we propose a parallel strategy for in-loop filtering in HEVC encoder on graphics processing unit (GPU). In the proposed strategy, the pipeline structure for HEVC encoding by parallel processing deblocking filter and SAO on GPU is described first. Then, the joint optimization for deblocking filter and SAO on GPU is detailed by parallel processing of deblocking filter and parallel processing of SAO separately. The joint optimization can improve the degree of parallelism and ease the computational burden of the CPU. Experimental results demonstrate that the proposed method can achieve about 47% (up to 67%) time saving on average for test sequences.**

***Index Terms*—HEVC encoder, in-loop filter, GPU, deblocking filter, SAO.**

I. Introduction

HIGH Efficiency Video Coding (HEVC) standard developed by the Joint Collaborative Team on Video Coding (JCT-VC) [1] becomes the state-of-the-art video coding standard, which can provide similar perceptual quality with about 50% bitrate saving compared with its predecessor H.264/AVC [2]. This remarkable improvement mainly comes from the following techniques, such as flexible quad-tree coding block partitioning structure, multi-angular intra prediction, advanced motion vector prediction, in-loop filtering, and improved context-adaptive binary arithmetic coding. In-loop filtering, as an important part of HEVC, consists of deblocking filter and Sample Adaptive Offset filter (SAO) [3]. According to [4], the coding gains of deblocking filter are about 1.3%, 2.6%, and 3.3% for luma component on average in All Intra (AI), Random Access (RA), and Low Delay P (LDP) configurations respectively. According to [5], the coding gains of SAO are about 0.7%, 1.7%, 9.2%, and 2.5% for luma component on average in AI, RA, LDP, and Low Delay B (LDB) configurations respectively. In-loop filtering can not only improve the compression efficiency of HEVC, but also improve the visual quality of the reconstructed videos significantly.

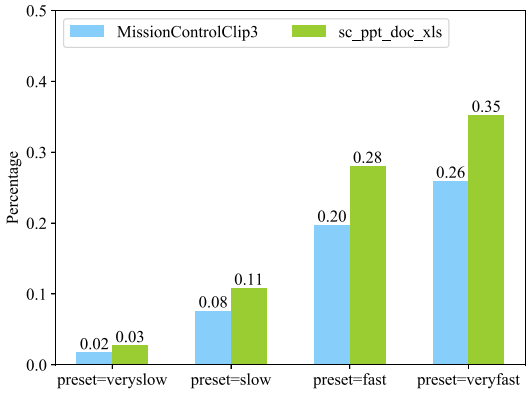

Fig. 1. Complexity percentages of in-loop filtering in HEVC encoder.

However, its high computational complexity hampers its applications for real-time encoding scenarios, especially for low-power devices. For instance, in x265, which is one of the best open-source HEVC encoders, the encoding complexity percentage of in-loop filtering is fairly high. As shown in Fig. 1, the percentage increases 2%-35% with different encoding configurations. With the popularity of consumer electronic devices (e.g., smartphones, tablets, laptops), it is inevitable to reduce the complexity of in-loop filtering for real-time encoding in these devices. Moreover, it is more challenging for consumer electronic devices due to the low-power and limited computing resources. Benefit from the development of GPU in mobile devices, we propose to reduce the complexity of in-loop filtering in HEVC encoder on GPU.

In recent years, there have been some research to reduce the computational complexity of deblocking filter and SAO. They can be classified into three categories. The first category mainly focuses on reducing the complexity by utilizing coding information or texture information [6], [7]. Kang *et al*. [6]

Manuscript received March 13, 2018; revised June 13, 2018 and August 16, 2018; accepted August 17, 2018. Date of publication August 29, 2018; date of current version September 24, 2018. This work was supported in part by the Major State Basic Research Development Program of China (973 Program) under Grant 2015CB351804, and in part by the National Science Foundation of China under Grant 61472101 and Grant 61631017. *(Corresponding author: Xiaopeng Fan.)*

Y. Wang, X. Fan, and D. Zhao are with the Department of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China (e-mail: wangyang.cs@hit.edu.cn; fxp@hit.edu.cn; dbzhao@hit.edu.cn).

X. Guo and Y. Lu are with Microsoft Research Asia, Beijing 100080, China (e-mail: xun.guo@microsoft.com; yanlu@microsoft.com).

W. Gao is with the Department of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: wgao@pku.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCE.2018.2867812

proposed a novel and efficient deblocking algorithm for HEVC, in which the complexity of boundary decision was reduced by using coding information, including depth, transform index, and partitioning size. Joo *et al*. [8] proposed to decide the best edge offset type in SAO according to intra prediction mode. Furthermore, they also proposed a fast parameter estimation algorithm for SAO by using the dominant edge direction [9]. Zhengyong *et al*. [10] utilized the coding unit partitioning information to accelerate encoding process of SAO. Besides, Yang *et al*. [7] accelerated the process of SAO by using the temporal relationship of SAO parameters in current coding tree unit (CTU) and the collocated CTU.

The second category focuses on parallel processing on CPU to reduce the complexity of deblocking filter and SAO [11], [12], which generally contains several threads. Kotra *et al*. [11] implemented three different parallelization deblocking methods on changing the order of horizontal filtering and vertical filtering. Yan *et al*. [12] proposed to process edge decision and boundary strength calculation simultaneously before edge filtering and then parallelize edge filtering with directed acyclic graphs.

The third category focuses on parallel processing on graphics processing unit (GPU) to reduce the complexity of deblocking filter and SAO [13]–[19], which usually contains thousands of threads. GPU is very efficient at manipulating computer graphics and image processing, and the highly parallel structure of GPU makes it more efficient than general-purpose CPU. In [13]–[15], different parallelization methods to accelerate deblocking filter on GPU were proposed. De Souza *et al*. [13] proposed three novel optimization algorithms for maximum parallelism level in deblocking filter of HEVC decoder, including a highly optimized CPU parallel implementation, GPU implementation of deblocking filter, and a hybrid and load-balanced CPU+GPU implementation. Jiang *et al*. [14] presented a novel parallel optimization strategy to improve the parallel performance of deblocking filter. Eldeken *et al*. [15] proposed a parallel-straight processing order that improved concurrency for deblocking multiple horizontal and vertical edges. In [16] and [17], the data dependency of SAO in HEVC encoder was eliminated on GPU. Zhang and Guo [16] proposed a parallel algorithm for SAO in HEVC on GPU, in which parallel algorithms for statistical information collection, calculation of the best offset and minimum distortion, SAO merging were proposed. In our preliminary work [17], we proposed an optimization algorithm for SAO in HEVC encoder on GPU, in which the first step of SAO was processed on GPU implemented with OpenCL. De Souza *et al*. [18] optimized deblocking filter and SAO by using GPU parallelization in HEVC decoder for embedded systems. Wang *et al*. [19] presented a frame-level strategy to improve the parallelism of in-loop filtering in HEVC decoder on GPU. However, the processes of SAO in HEVC encoder and decoder are different. For SAO in HEVC encoder, pixel-wise constraint exists in type classification and offset computation stages, which will cause much more difficulty to jointly optimize deblocking filter and SAO in HEVC encoder. Different from optimizing in-loop filtering in HEVC decoder [18], [19], we propose a parallel strategy for in-loop filtering in HEVC encoder. Specifically, CTU compression (block partition, mode decision, transform and quantization, and reconstruction) is processed on CPU, while in-loop filtering is processed on GPU. By jointly optimizing deblocking filter and SAO on GPU, the complexity of in-loop filtering is reduced.

The motivations of this research are as follows:

First, the encoding complexity percentage of in-loop filtering is fairly high. When in-loop filtering is processed on GPU, it will not only improve the degree of parallelism in HEVC encoder, but also ease the computing burden of the CPU, especially for power-limited or low-computing capacity devices.

Second, the processes of SAO in HEVC encoder and decoder are different. For SAO in HEVC encoder, pixel-wise constraint exists in type classification and offset computation stages, which will lower the parallelism of GPU. Due to this difficulty, few existing algorithms have been proposed to optimize SAO on GPU in HEVC encoder.

Furthermore, to the best of our knowledge, none algorithms have been presented to jointly optimize deblocking filter and SAO in HEVC encoder on GPU.

In this paper, we propose a parallel strategy for in-loop filtering in HEVC encoder on GPU. Different from individually deblocking filter optimization [13]–[15] or SAO optimization [16], [17], we optimize deblocking filter and SAO together on GPU, which reduces the data accessing between CPU and GPU. Different from jointly optimizing deblocking filter and SAO in HEVC decoder [18], [19], we optimize deblocking filter and SAO in HEVC encoder. In the proposed strategy, the pipeline structure for HEVC encoding by parallel processing deblocking filter and SAO on GPU is described first. Then, the joint optimization for deblocking filter and SAO on GPU is detailed by parallel processing of deblocking filter and parallel processing of SAO separately. The joint optimization can improve the degree of parallelism and ease the computational burden of the CPU. Experimental results demonstrate that the proposed method can achieve about 47% (up to 67%) time saving on average for test sequences.

The rest of this paper is organized as follows. Section II briefly reviews the in-loop filtering in HEVC and the related works. Section III introduces the details of joint optimization for deblocking filter and SAO on GPU, including the pipeline structure, the parallel processing of deblocking filter, the parallel processing of SAO, and memory management. Experimental results and analysis are given in Section IV. Finally, Section V concludes the paper.

II. In-Loop Filtering in HEVC Encoder

In HEVC, in-loop filtering consists of deblocking filter and SAO. It is applied to the reconstructed pixels just before writing them into the decoded picture buffer within the inter prediction loop. Similar to that in H.264/AVC, deblocking filter in HEVC aims to reduce the visibility of blocking artifacts caused by block-based coding structure and is applied only to samples located at block boundaries. Whereas, SAO is a newly adopted tool in HEVC, which is intended to improve

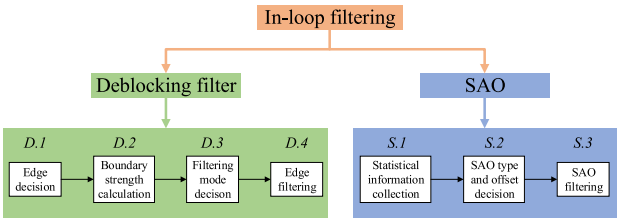

Fig. 2. In-loop filtering in HEVC encoder.

accuracy of reconstruction of the original signal. SAO is performed adaptively for all samples by conditionally adding an offset value to each sample based on values in look-up tables defined by HEVC encoder. In this subsection, we will briefly review the process of in-loop filtering in HEVC encoder, and more details can be found in [1], [4], and [5].

A. Deblocking Filter

Deblocking filter is applied to the edges aligned on 8×8 sample grids. As denoted in Fig. 2, it can be split into four steps, namely edge decision, boundary strength calculation, filtering mode decision, and edge filtering. They are represented by *D*.1, *D*.2, *D*.3, and *D*.4 respectively.

In *D*.1, only the edges on 8×8 grids, either prediction unit or transform unit boundaries, are considered to be filtered.

In *D*.2, the boundary strength is calculated according to the coding information. The boundary strength can take one of three possible values: 0, 1, and 2, which indicate no filtering, weak filtering, and strong filtering respectively.

In *D*.3, if the boundary strength value is 1 or 2, additional conditions are checked to determine whether the deblocking filtering should be applied. The weak or strong filtering mode is decided according to values of samples near the boundaries.

In *D*.4, first vertical edges are filtered (horizontal filtering) then horizontal edges are filtered (vertical filtering).

B. Sample Adaptive Offset

As denoted in Fig. 2, SAO consists of three steps, namely statistical information collection, SAO type and offset decision, and SAO filtering. They are represented by *S*.1, *S*.2, and *S*.3 respectively.

In *S*.1, the number of pixels and the sum of distortion of a certain SAO type are calculated.

In *S*.2, the final best SAO type and offset are selected from candidates of edge offset (EO) and band offset (BO) types and offsets.

In *S*.3, pixels in the CTU are filtered by adding the corresponding best offset values conditionally.

In SAO, each CTU is classified into one of five SAO types: four EOs and one BO. As shown in Fig. 3 (a), EO uses four gradient 3-pixel patterns for classification of the current pixel *C* by considering the edge directional information with other two neighboring pixels N_1 and N_2. As shown in Fig. 3 (b), BO uses pixel intensity for pixel classification. BO classifies all pixels in the CTU into 32 uniform bands (for 8-bit pixels). Four consecutive bands are grouped together and each group is indicated by the starting band position. Only offsets of four consecutive bands and the starting band position are signaled to the decoder. In the following, we will review the related works on optimization for deblocking filter and SAO in HEVC on GPU.

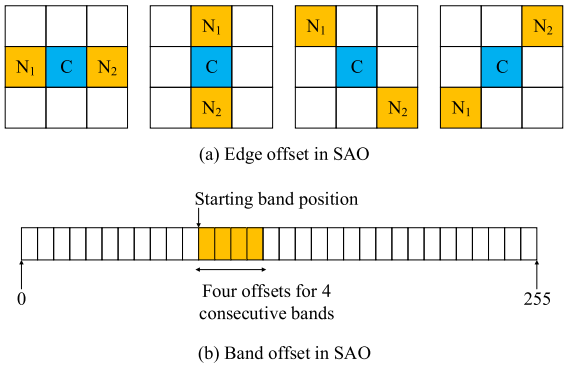

Fig. 3. Edge offset and band offset in SAO.

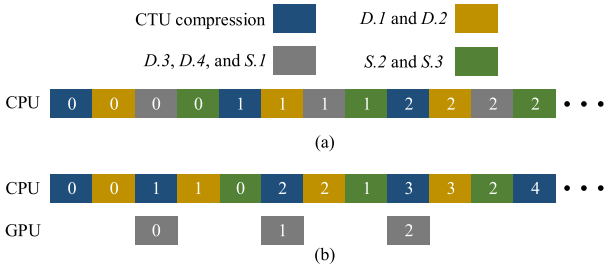

Fig. 4. The pipeline structure by the cooperation between CPU and GPU. The number on the block denotes the row number of CTU row.

III. Proposed Joint Optimization for Deblocking Filter and SAO on GPU

In this section, we will introduce the details of the proposed joint optimization for deblocking filter and SAO in HEVC encoder on GPU. At first, we propose a pipeline structure for HEVC encoding process by parallel processing in-loop filtering on GPU. Then the details of parallel processing designs on deblocking filter and SAO are described in Sections III-B and III-C. Finally, the strategy for memory management is discussed in Section III-D.

A. Overview of the Pipeline Structure

HEVC provides a new encoding method in parallel, namely wavefront parallel processing (WPP). When this method is enabled, a slice is divided into multiple rows of CTUs. Each row is encoded by a thread. The first row is processed in the ordinary way, the second row begins to be processed after only two CTUs in the first row have been processed, the third row begins to be processed after only two CTUs in the second row have been processed, etc. WPP not only improves the degree of parallelism for current frame, but also supports the frame-level encoding in parallel.

As shown in Fig. 2, deblocking filter is divided into *D*.1, *D*.2, *D*.3, and *D*.4, corresponding to edge decision, boundary strength calculation, filtering mode decision, and edge filtering.

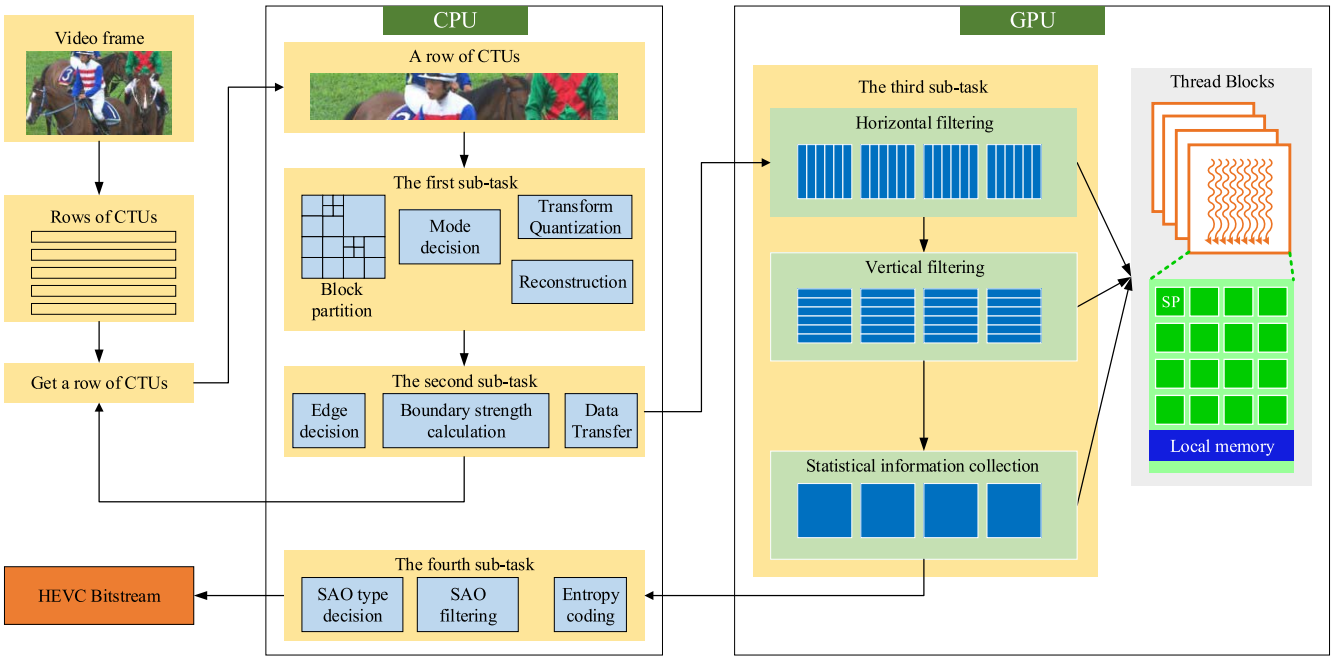

Fig. 5. The details of HEVC encoding process by parallel processing in-loop filtering on GPU.

SAO is divided into *S*.1, *S*.2, and *S*.3, corresponding to statistical information collection, SAO type and offset decision, and SAO filtering. To further improve the degree of parallelism in HEVC encoder, we propose to divide the encoding process for each row of CTUs into four sub-tasks. The first sub-task is CTU compression, consisting of block partition, mode decision, transform and quantization, and reconstruction for CTUs. The second sub-task consists of *D*.1 and *D*.2. The third sub-task consists of *D*.3, *D*.4, and *S*.1. The fourth sub-task consists of *S*.2 and *S*.3. As shown in Fig. 4 (a), when HEVC encoding is processed only on CPU, these four sub-tasks for a row of CTUs are processed sequentially. The proposed pipeline structure for HEVC encoding is shown in Fig. 4 (b) by the cooperation between CPU and GPU. Specially, the third sub-task, consisting of *D*.3, *D*.4, and *S*.1, which are the most time-consuming steps of in-loop filtering, is jointly optimized and processed on GPU, while the others are still processed on CPU. The overall encoding time can be reduced by using this pipeline structure. In order to be compatible with WPP, the proposed HEVC encoding is also processed row by row.

Details of HEVC encoding process by parallel processing in-loop filtering on GPU are shown in Fig. 5. In Fig. 5, a video frame is first divided into different rows of CTUs. Then CTUs in a row are processed one by one. The first sub-task named CTU compression includes block partition, mode decision, transform and quantization, and reconstruction.

The second sub-task includes *D*.1 and *D*.2 in deblocking filter. This sub-task need to transfer data from CPU to GPU when in-loop filtering is processed on GPU.

The third sub-task consists of *D*.3, *D*.4, and *S*.1, which are processed on GPU, as shown in the right part of Fig. 5. First, all vertical edges in the CTUs of a row are filtered. Then all horizontal edges in the CTUs of a row are filtered. Statistical information is collected by using original pixels and reconstructed pixels which have been filtered by *D*.3 and *D*.4. This joint optimization on deblocking filter and SAO can make full use of computing resources. Data dependency exists between deblocking filter and SAO. The deblocked pixels are as input for SAO. Both of two filters share several data, such as the original pixels, coding information. The details of deblocking filter and SAO on GPU are introduced in Sections III-B and III-C respectively.

The fourth sub-task includes *S*.2, *S*.3 in SAO, and entropy coding for a row of CTUs. This sub-task needs the data transferred from GPU.

Compute Unified Device Architecture (CUDA) is a parallel computing platform and application programming interface model. CUDA is selected as the platform for the proposed in-loop filtering on GPU.

B. Parallel Processing of Deblocking Filter

There are four steps in the deblocking filter, namely edge decision, boundary strength calculation, filtering mode decision, and edge filtering, which are represented by *D*.1, *D*.2, *D*.3, and *D*.4 respectively in Fig. 2. In general, *D*.1 and *D*.2 are executed recursively, which are not suitable for parallel processing on GPU. According to [20], *D*.3 and *D*.4 are the most time consuming parts of deblocking filter. We propose to process *D*.3 and *D*.4 on GPU.

For processing *D*.3 and *D*.4 on GPU, the data such as boundary strength values, unfiltered pixels, and quantization parameters of each CTU are need to be transferred from CPU to GPU. On GPU, all vertical edges of the CTUs in a row are filtered concurrently first. Then all horizontal edges of the CTUs in a row are filtered concurrently. Fig. 6 and Fig. 7 depict the thread mapping of the horizontal filtering and the vertical filtering on GPU respectively. All CTUs in a row are

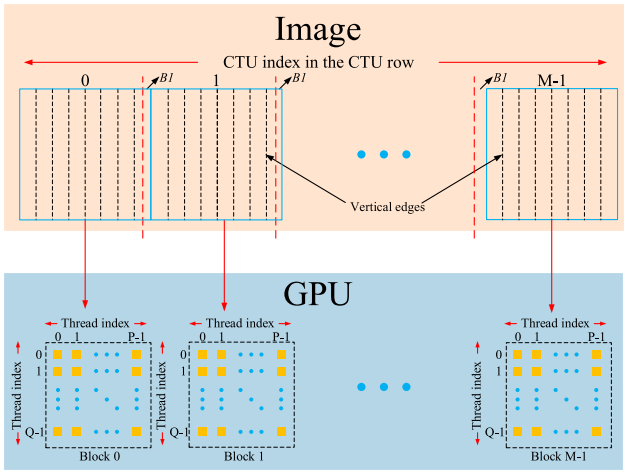

Fig. 6. Thread mapping of horizontal filtering.

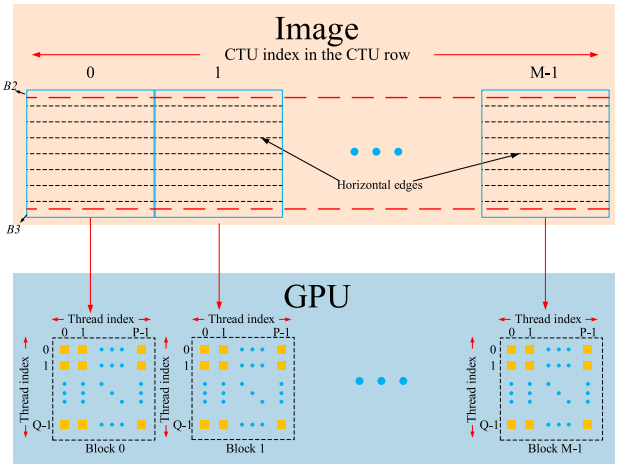

Fig. 7. Thread mapping of vertical filtering.

processed concurrently. Each CTU is processed by threads in a thread block on GPU. Since deblocking filter is applied on 8×8 grids, all edges aligned on 8×8 grids can be processed simultaneously. Horizontal filtering is applied first, and then vertical filtering is applied.

For horizontal filtering, as shown Fig. 6, all vertical edges in a CTU are processed by the threads in the same thread block. We take CTU of 64×64 as an example, since deblocking filter is applied on 8×8 grid, only up to 8 vertical edges may need to be filtered. Regarding 4 pixel lines as a unit, a vertical edge in the CTU needs 16 threads for filtering. In this case, the numbers of threads in horizontal and vertical directions are 8 and 16 respectively. Therefore, the total number of threads for horizontal filtering in a CTU is 128. Note that the rightmost 4 pixel columns in current CTU, denoted by $B1$ in Fig. 6, are processed by threads of next block since the rightmost vertical edge belongs to next CTU.

For vertical filtering, as shown in Fig. 7, all horizontal edges in a CTU are processed by the threads in the same thread block. If we take CTU of 64×64 as an example, similar results as horizontal filtering can also be obtained. Note that the upmost 4 pixel lines, denoted by $B2$ in Fig. 7, in previous row of CTUs are also processed, which is caused by deblocking filter delay. In addition, the bottommost 4 pixel lines, denoted by $B3$ in Fig. 7, are left to be processed by the next row of CTUs.

C. Parallel Processing of SAO

SAO in HEVC encoder consists of three stages, namely statistical information collection, SAO type and offset decision, and SAO filtering, represented by $S.1$, $S.2$, and $S.3$ respectively in Fig. 2. According to [7] and [21], $S.1$ is the most time consuming part of SAO, and its computational complexity occupies about 80%∼90% of SAO encoding process. We propose to process $S.1$ on GPU with $D.3$ and $D.4$ together.

In our preliminary work [17], we proposed a GPU-based SAO optimization algorithm in HEVC encoder. It reduced the processing time of SAO in HEVC encoder significantly by implementing $S.1$ on GPU. In this work, we also implement the $S.1$ on GPU using CUDA instead of OpenCL with different data access and memory management.

Two main issues need to be considered when optimizing $S.1$ of SAO on GPU. First, the number of total pixels belonging to each SAO type need to be counted, and offset values of each pixel have to be collected to compute the overall distortion. This loop structure cannot be straightly applied for parallel processing on GPU. Second, when collecting statistical information for different SAO types, different number of pixels are utilized for computing. GPU has to perform logical judgement pixel by pixel to select the correct ones for computation. This logical judgement degrades performance of parallelism on GPU extremely. It also leads to more data access for one pixel line.

Similar to deblocking filter optimization on GPU, all CTUs in a row are also processed simultaneously in SAO optimization. The thread mapping of SAO optimization on GPU is shown in Fig. 8. All CTUs in the row are denoted by the index from 0 to $M-1$. Since each CTU is processed by one block, the number of thread blocks on GPU is the same as the number of CTUs in the row, which is also M.

As shown in Fig. 8, each block contains P threads, where P is equal to the height of CTU. The total number of threads used for one row of CTUs is $N = P * M$, where N denotes the total number of threads used for one row of CTUs.

To solve the first issue, we utilize memory and threads of GPU sufficiently. Specially, all the pixels in the same CTU are processed by threads within a thread block. Each pixel line within a CTU is assigned to a thread for counting the number of total pixels for each SAO type and distortion computing. By this means, the whole CTU can be processed simultaneously by one block of threads. In addition, all threads in a block can share the same local memory, which reduces the delay of data accessing.

After all blocks of threads are finished, the statistical results of the CTUs including 4 EOs and 1 BO are transferred back to CPU and are used to determine the optimal one by rate distortion optimization. To save data exchange overhead, all original pixels and reconstructed pixels of one row of CTUs are preloaded into GPU memory before the process of deblocking

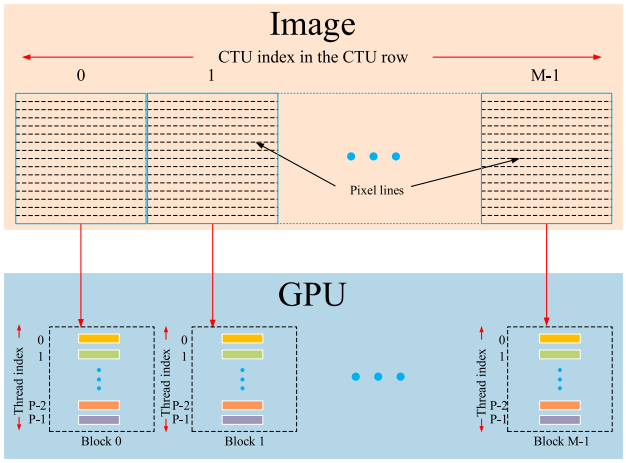

Fig. 8. Thread mapping of SAO.

TABLE I
PIXEL EXCLUDED IN COLLECTING INFORMATION FOR SAO

SAO Type	Right columns	Bottom rows
BO	3	4
EO_0 (horizontal)	3	5
EO_1 (vertical)	4	4
EO_2 (135 degree)	4	5
EO_3 (45 degree)	4	5

filter. For the second issue, as there is a constrain condition on pixels when collecting statistical information. Different SAO types use different number of pixels for SAO type classification and offset computing. Specifically, when deblocking filter is enabled, pixels in the right 4 columns and in the bottom 5 rows of the CTU are excluded in collecting statistical information of luma component. However, when deblocking filter is disabled, the excluded pixels depend on SAO types, which are shown in Table I. Please note that the excluded pixels will still be added on corresponding offsets, although they are not used for offset computation.

This constrain is a big burden for parallel processing. GPU has to perform logical judgement pixel by pixel to select the correct ones for computation, which also leads to more data access for one pixel line. This reduces the speed of computation dramatically. To avoid the flaw, we remove this constrain, which means all the pixels within a CTU are used for statistical information collection for all SAO types no matter the deblocking filter is enabled or not. This change is non-normative that will not change the syntax of SAO. There are two advantages for this change. First, judgement on GPU is removed so that the computational complexity is reduced. Second, more pixels are used to calculate SAO offsets, which increases the accuracy.

One possible mismatch may occur when deblocking filter is enabled and CTU level processing is used. In such case, several pixel lines in the bottom of a CTU will not be deblocked until its lower CTU in next row is in deblocking filter. This is to avoid adding additional line-buffers in hardware to store the un-deblocked pixels in the upper CTU for the use of the lower CTU. In our algorithm, these un-deblocked pixel lines will be included in the offset computation together with other deblocked pixels. No coding loss is observed in our experiments.

TABLE II
SEQUENCES USED IN THE EXPERIMENT

Class	Sequence	Resolution	fps	format	frames
Class A	*Traffic*	2560x1600	30	YUV420	150
	PeopleOnstreet		30		150
Class B	*ParkScene*	1920x1080	24	YUV420	240
	Cactus		50		500
	BasketballDrive		60		600
	BQTerrace		60		600
Class UHD	*CampfireParty*	3840x2160	30	YUV420	300
	Fountains		30		300
	Runners		30		300
	RushHour		30		300
	TrafficFlow		30		300
Class S	*MissionControlClip3*	1920x1080	60	YUV444	600
	sc_cad_waveform		20		200
	sc_console		60		600
	sc_desktop		60		600

D. Memory Management

For the in-loop filtering optimization algorithm on GPU, apart from improvement on the algorithm itself, it is necessary to optimize utilization of memories on GPU. Memories on CUDA can be classified into three categories, namely global memory, private memory and local memory, among which local memory plays an important role in improving the efficiency of the proposed parallel strategy.

In the stage of deblocking filter, local memory is used to for storing intermediate results of $D.1$, $D.2$, and $D.3$. Pixels in the same CTU are stored in local memory, and then are filtered when horizontal filtering and vertical filtering applied. In the stage of SAO, similar strategy is adopted to efficiently utilize local memory. When collect statistical information for each CTU, pixels in the same CTU are stored in local memory, which is used for counting number of pixels belonging to each SAO type and calculating corresponding offset. Then final results for statistical information collection are copied from local memory to global memory.

IV. EXPERIMENTAL RESULTS

In this section, extensive experiments are conducted in order to evaluate the performance of the proposed parallel strategy for in-loop filtering in HEVC encoder on GPU. Section IV-A describes the experimental settings. The complexity reduction is evaluated and compared in Section IV-B. The compression performance is provided in Section IV-C.

TABLE III
PROCESSING TIME (IN MILLISECONDS) OF IN-LOOP FILTERING FOR DIFFERENT METHODS

Class	Sequence	ILFC (x265)	ILFCG (proposed)		ILFG (proposed)	
		Time (ms)	Time (ms)	ATS	Time (ms)	ATS
Class A	*Traffic*	26.6	16.5	38.1%	14.9	44.1%
	PeopleOnStree	28.6	20.3	29.1%	15.6	45.5%
Class B	*ParkScene*	19.6	15.3	21.8%	13.6	30.4%
	Cactus	42.0	30.0	28.6%	28.0	33.4%
	BaskeballDrive	44.2	32.5	26.6%	28.8	34.8%
	BQTerrace	101.7	71.3	29.9%	67.6	33.5%
Class UHD	*CampfireParty*	214.7	133.1	38.0%	103.2	51.9%
	Fountains	131.6	81.8	37.8%	67.0	49.1%
	Runners	59.5	39.6	33.5%	33.6	43.5%
	RushHour	40.8	37.6	26.0%	31.8	37.5%
	TrafficFlow	58.1	32.6	43.8%	29.5	49.2%
Class S	*MissionControlClip3*	91.7	36.7	59.9%	35.7	61.1%
	sc_cad_waveform	91.7	34.5	62.4%	34.2	62.7%
	sc_console	88.6	37.4	57.7%	34.9	60.7%
	sc_desktop	114.1	37.4	67.2%	37.2	67.4%
Average		77.6	43.8	40.0%	38.4	47.0%

A. Experimental Settings

x265 is used as the HEVC video encoder, which is one of the best open source HEVC encoders [22]. It has been well optimized on coding structure and encoding modules, e.g., motion estimation by using fast algorithms and SSE instruction set. The experimental results are obtained under HEVC common test conditions [23] by applying the default parameter of x265.

Video sequences with higher resolution (Classes A and B) in [23] are used, since they are the most computationally demanding. An additional set of Ultra HD 4K sequences [24] is also evaluated (Class UHD). Besides, screen content video, is emerging as a popular video type and typified by the computer and mobile display content such as, remote desktop, video conference, distance education, cloud gaming and computing. As there is no sampling noise for screen content videos, in-loop filtering improves the visual quality significantly. A set of screen content sequences [25] is also included in the experiment (Class S). The information of video sequences used for experiments is shown in Table II. The computer with 3.4 GHz Octa-core processors and 64GB memory is used for simulation, and the number of streaming processors on GPU is 2560.

B. Complexity Reduction

First, we evaluate the processing time of the proposed strategy. Average time saving (*ATS*) is used to evaluate the complexity reduction. *ATS* denotes the average time saving of four quantization parameters (22, 27, 32, and 37), which is defined as follows:

$$ATS = \frac{T_{x265} - T_{proposed}}{T_{x265}} \times 100\% \quad (1)$$

where T_{x265} and $T_{proposed}$ denote the processing time per frame of in-loop filtering of x265 and the proposed method respectively.

To evaluate the performance, we compare three methods. The first method is the baseline. It processes in-loop filtering on CPU, so we call it ILFC. The second method ILFCG processes in-loop filtering partially on CPU and partially on GPU. Specifically, ILFCG processes deblocking on CPU and SAO on GPU. The third method ILFG processes in-loop filtering on GPU. Table III shows the processing time of these methods respectively.

As shown in Table III, the processing time of ILFG is much less than that of ILFC. For Class A, the processing time of in-loop filter on CPU is about 27.6ms on average, and on the contrary on GPU, it is only about 15.25ms on average. 45% time saving can be achieved. Similarly, for Class B, the processing time of in-loop filter on CPU is about 51.9ms, and on GPU it is only about 34.5ms. 33% time saving can be achieved. For high resolution sequences, namely Class UHD, the processing time of in-loop filter on CPU is about 102.9ms, and on GPU it is only about 53ms. 46.2% time saving can be achieved. For screen content sequences, the processing time of in-loop filter on CPU is about 96.5ms, and on GPU it is only about 26.2ms. 63% time saving can be achieved. Experimental results demonstrate that the proposed ILFG strategy can achieve about 47% (up to 67%) time saving on average for test sequences. Note that the experiment result of ILFC is achieved when SAO on CPU has been optimized by SIMD, which can accelerate the process of SAO significantly.

When deblocking filter is implemented on CPU and SAO is implemented on GPU (ILFCG), the encoding complexity of in-loop filter can also be reduced, it can achieve about 40%

TABLE IV
COMPARISON WITH THE STATE-OF-THE-ART METHODS

Class	Joo [9]	Wang [19]	ILFG
Class A	12.3%	18.6%	44.8%
Class B	12.3%	15.4%	33.0%
Class UHD	28.4%	15.8%	46.2%
Class S	40.3%	3.6%	63.0%
Average	23.3%	13.3%	47.0%

TABLE V
THE COMPRESSION PERFORMANCE OF THE PROPOSED METHOD

Class	BD-Y	BD-U	BD-V
Class A	-0.10%	0.05%	0.10%
Class B	-0.23%	0.15%	0.03%
Class UHD	-0.22%	0.18%	0.04%
Class S	-0.38%	0.03%	0.00%
Average	-0.23%	0.10%	0.04%

time saving on average for test sequences. The proposed joint optimization of deblocking filter and SAO on GPU (ILFG) outperforms ILFCG with about 7.0% average time saving.

It can be observed that the time saving for sequences with higher resolution is much bigger than that for sequences with lower resolution. This is due to that for sequences with higher resolution, there are more CTUs in a row. The computing time on CPU increases linearly; however, it is nearly unchanged on GPU due to its parallel structure.

In addition, the time saving for screen content sequences is much bigger than that for natural sequences, as the sampling frequency for chroma components is the same as luma component in screen content sequences.

In order to further evaluate the performance of ILFG, ILFG is compared with [9] and [19]. Reference [9] optimizes SAO in HEVC encoder. Reference [19] optimizes the in-loop filtering in HEVC decoder. As the processes of SAO in encoder and decoder are different, only deblocking filter in [19] is used for comparison. As shown in Table IV, the results demonstrate the superiority of the proposed method. The proposed method can achieve better performance than deblocking filter or SAO separately optimized and it can save data accessing time between CPU and GPU.

C. Compression Performance

As mentioned in Section III-C, when collecting the statistical information of SAO in the proposed strategy, all the pixels within a CTU are used for all SAO types no matter the deblocking filter is enabled or not, which is different from the original process: different SAO types use different number of pixels for SAO type classification and offset computing. Bjϕntegaard distortion rate (BD-rate) [26] is used to evaluate the compression performance, in which the negative number indicates bitrate saving and the positive number indicates bitrate increasing for the same quality.

As shown in Table V, small coding gain can be achieved for luma component for all test sequences, which can reach -0.23% on average. This is due to that when collecting statistical information for SAO, we use more pixels for counting the number of total pixels for each SAO type and distortion computing. The statistical information is more accurate, which makes the compression efficiency better.

V. CONCLUSION

In this paper, we propose a parallel strategy for in-loop filtering in HEVC encoder on GPU. The pipeline structure of HEVC encoding by the cooperation between CPU and GPU is described first. Then, the parallel implementation on GPU for deblocking filter is provided. After that, the parallel implementation on GPU for SAO is provided, in which the constraint against parallelization in SAO is eliminated. The joint optimization on deblocking filter and SAO can improve the degree of parallelism and ease the computational burden of CPU. The proposed method can achieve about 47% time reduction on average for all test sequences.

REFERENCES

[1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[3] C. Rosewarne, B. Bross, M. Naccari, K. Sharman, and G. J. Sullivan, *High Efficiency Video Coding (HEVC) Test Model 16 (HM 16) Update 5 of Encoder Description*, document JCTVC-W1002, JCTVC, Feb. 2016.

[4] A. Norkin *et al.*, "HEVC deblocking filter," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1746–1754, Dec. 2012.

[5] C.-M. Fu *et al.*, "Sample adaptive offset in the HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1755–1764, Dec. 2012.

[6] R. Kang, W. Zhou, X. Huang, and B. Dong, "An efficient deblocking filter algorithm for HEVC," in *Proc. IEEE China Summit Int. Conf. Signal Inf. Process. (ChinaSIP)*, Xi'an, China, Jul. 2014, pp. 379–383.

[7] K. Yang, S. Wan, Y. Gong, Y. Yang, and Y. Feng, "Fast sample adaptive offset for H.265/HEVC based on temporal dependency," in *Proc. Asia–Pac. Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA)*, Dec. 2016, pp. 1–4.

[8] J. Joo, Y. Choi, and K. Lee, "Fast sample adaptive offset encoding algorithm for HEVC based on intra prediction mode," in *Proc. IEEE 3rd Int. Conf. Consum. Electron. Berlin (ICCE-Berlin)*, Berlin, Germany, Sep. 2013, pp. 50–53.

[9] J. Joo and Y. Choi, "Dominant edge direction based fast parameter estimation algorithm for sample adaptive offset in HEVC," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Paris, France, Oct. 2014, pp. 3749–3752.

[10] Z. Zhengyong, C. Zhiyun, and P. Peng, "A fast SAO algorithm based on coding unit partition for HEVC," in *Proc. 6th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Beijing, China, Sep. 2015, pp. 392–395.

[11] A. M. Kotra, M. Raulet, and O. Deforges, "Comparison of different parallel implementations for deblocking filter of HEVC," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Vancouver, BC, Canada, May 2013, pp. 2721–2725.

[12] C. Yan, Y. Zhang, F. Dai, and L. Li, "Efficient parallel framework for HEVC deblocking filter on many-core platform," in *Proc. Data Compression Conf.*, Mar. 2013, p. 530.

[13] D. F. de Souza, N. Roma, and L. Sousa, "Cooperative CPU+GPU deblocking filter parallelization for high performance HEVC video codecs," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Florence, Italy, May 2014, pp. 4993–4997.

[14] W. Jiang *et al.*, "A novel parallel deblocking filtering strategy for HEVC/H.265 based on GPU," *Concurrency Comput. Pract. Exp.*, vol. 28, no. 16, pp. 4264–4276, 2016.

[15] A. F. Eldeken, R. M. Dansereau, M. M. Fouad, and G. I. Salama, "High throughput parallel scheme for HEVC deblocking filter," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Quebec City, QC, Canada, Sep. 2015, pp. 1538–1542.
[16] W. Zhang and C. Guo, "Design and implementation of parallel algorithms for sample adaptive offset in HEVC based on GPU," in *Proc. 6th Int. Conf. Inf. Sci. Technol. (ICIST)*, Dalian, China, May 2016, pp. 181–187.
[17] Y. Wang, X. Guo, Y. Lu, X. Fan, and D. Zhao, "GPU-based optimization for sample adaptive offset in HEVC," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Phoenix, AZ, USA, Sep. 2016, pp. 829–833.
[18] D. F. de Souza, A. Ilic, N. Roma, and L. Sousa, "HEVC in-loop filters GPU parallelization in embedded systems," in *Proc. Int. Conf. Embedded Comput. Syst. Archit. Model. Simulat. (SAMOS)*, Jul. 2015, pp. 123–130.
[19] B. Wang *et al.*, "GPU parallelization of HEVC in-loop filters," *Int. J. Parallel Program.*, vol. 45, no. 6, pp. 1515–1535, Dec. 2017. [Online]. Available: https://doi.org/10.1007/s10766-017-0488-z
[20] Z. Yang, W. Gao, Y. Liu, and D. Zhao, "Deeply pipelined DSP solution to deblocking filter for H.264/AVC," *IEEE Trans. Consum. Electron.*, vol. 52, no. 4, pp. 1267–1274, Nov. 2006.
[21] Y. Choi and J. Joo, "Exploration of practical HEVC/H.265 sample adaptive offset encoding policies," *IEEE Signal Process. Lett.*, vol. 22, no. 4, pp. 465–468, Apr. 2015.
[22] *The x265 Website*. [Online]. Available: https://bitbucket.org/multicoreware/x265/wiki/Home
[23] F. Bossen, *Common Test Conditions and Software Reference Configurations*, document JCTVC-L1100, JCTVC, Jan. 2013.
[24] L. Song, X. Tang, W. Zhang, X. Yang, and P. Xia, "The SJTU 4K video sequence dataset," in *Proc. 5th Int. Workshop Qual. Multimedia Exp. (QoMEX)*, Klagenfurt, Austria, Jul. 2013, pp. 34–35.
[25] H. Yu, R. Cohen, K. Rapaka, and J. Xu, *Common Conditions for Screen Content Coding Tests*, document JCTVC-R1015, JCTVC, Jun. 2014.
[26] G. Bjøntegaard, *Improvements of the BD-PSNR Model*, document VCEG-AI11, ITU-T Video Coding Experts Group, Geneva, Switzerland, and Heinrich-Hertz-Inst., Berlin, Germany, Jul. 2008.

Yang Wang received the B.S. and M.S. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 2012 and 2014, respectively, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology.

From 2013 to 2014, he was with the School of Electronics Engineering and Computer Science, Peking University, Beijing, as a Research Assistant. From 2014 to 2016, he was with the Media Computing Group, Microsoft Research Asia, Beijing, as an Intern. His current research interests are in image processing, video coding, and deep learning.

Xun Guo (M'10) received the Ph.D. degree in computer science from the Harbin Institute of Technology, China, in 2007.

He joined Microsoft Research Asia in 2012, where he is currently a Lead Researcher of Media Computing Group. From 2007 to 2012, he was with MediaTek Inc., as a Group Manager, leading the development of core technologies on video coding and processing for HEVC. He holds over 20 granted U.S. patents. He has published over 50 papers and technical contributions in the area of video coding and multimedia. His current research interests include image and video coding, multimedia communication, and computer vision.

Xiaopeng Fan (S'07–M'09–SM'17) received the B.S. and M.S. degrees from the Harbin Institute of Technology (HIT), Harbin, China, in 2001 and 2003, respectively, and the Ph.D. degree from the Hong Kong University of Science and Technology (HKUST), Hong Kong, in 2009.

He joined HIT in 2009, where he is currently a Professor. From 2003 to 2005, he was with Intel Corporation, China, as a Software Engineer. From 2011 and 2012, he was with Microsoft Research Asia as a Visiting Researcher. From 2015 to 2016, he was with HKUST as a Research Assistant Professor. He has authored one book and over 100 articles in refereed journals and conference proceedings. His current research interests include video coding and transmission, image processing, and computer vision. He was a recipient of the Outstanding Contributions Award to the Development of IEEE Standard 1857 by IEEE in 2013. He served as the Program Chair of PCM2017, the Chair of IEEE SGC2015, and the Co-Chair of MCSN2015. He has been an Associate Editor of IEEE 1857 standard since 2012.

Yan Lu received the Ph.D. degree in computer science from the Harbin Institute of Technology, China.

He joined Microsoft Research Asia in 2004, where he is currently the Principal Research Manager of Media Computing Group, leading the development of core technologies around intelligent video analytics, computer vision, natural user interface, video coding, and communications. From 2001 to 2004, he was a Team Lead of video coding group in the JDL Lab, Institute of Computing Technology, China. From 1999 to 2000, he was with the City University of Hong Kong as a Research Assistant. He holds over 30 granted U.S. patents. He has also published over 100 papers in refereed journals and conference proceedings in the areas of image and video coding, multimedia, and computer vision.

Debin Zhao (M'11) received the B.S., M.S., and Ph.D. degrees in computer science from the Harbin Institute of Technology, China, in 1985, 1988, and 1998, respectively, where he is currently a Professor with the Department of Computer Science. He has published over 200 technical articles in refereed journals and conference proceedings in the areas of image and video coding, video processing, video streaming and transmission, and computer vision.

Wen Gao (M'92–SM'05–F'09) received the Ph.D. degree in electronics engineering from the University of Tokyo, Tokyo, Japan, in 1991. He was a Professor of computer science with the Harbin Institute of Technology, Harbin, China, from 1991 to 1995, and a Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He is currently a Professor of computer science with Peking University, Beijing. He has authored five books and over 600 technical articles in refereed journals and conference proceedings in image processing, video coding and communication, pattern recognition, multimedia information retrieval, multimodal interface, and bioinformatics.

He is a member of the Chinese Academy of Engineering.