Accelerating IEEE 1857 Deblocking Filter on GPU Using CUDA

Xiaoou Sun, Ronggang Wang
Peking University, Shenzhen Graduate School
Peking University School of Electronic and Computer Engineering
Shenzhen, China
oomnilol@163.com, rgwang@pkusz.edu.cn

***Abstract*—In IEEE 1857/AVS decoder, the deblocking filter is one of the most time consuming tasks and it accounts for nearly 30% of the decoder complexity. Due to data dependence in the process of filtering, it is a big challenge to execute deblocking filter in parallel efficiently. In this paper, a parallel algorithm is proposed, which hides the data dependence by dividing a frame into several 8 x 8 Intersection-blocks. The proposed parallel deblocking filter algorithm is implemented on GPU using CUDA. Experimental results of processing 1920 x 1080 video streams show that the proposed CUDA deblocking filter implementaion accelerates traditional CPU implementation by 6~8 times.**

Keywords-IEEE 1857; AVS; Deblocking filter; parallel processing; CUDA; GPU

I. INTRODUCTION

In recent years, people's growing requirement for videos makes video coding become the central technology in many applications. Some of these include digital TV, DVD, Internet streaming video, video conferencing, distance learning, and surveillance and security [1]. In order to address the requirement of different applications, some video coding standards has been created. One of them is Audio Video coding Standard (AVS) [2].

AVS is a compression standard developed by China Audio Video Coding Standard Working Group [3], and is competing with H.264/AVC to replace MPEG-2 [4]. AVS bring us higher compression efficiency than MPEG-2 but cause increase in the complexity of compression algorithms as well, which means the time cost of video decoding becomes higher, especially for high-definition videos. Deblocking filter is introduced to AVS to solve the blocking artifact caused by the block-based prediction and transformation. It can significantly improve the quality of reconstructed image, but also increases the time cost and computation complexity of AVS decoder. In AVS decoder, deblocking filter is one of the most time consuming tasks, it accounts for nearly 30% of the decoder complexity [5]. Traditionally, deblocking filter is executed in sequential way and has a large number of computations. Due to the strong computational locality exhibited by deblocking filter algorithm, only using software resource to accelerate this tool cannot meet the need of high-definition video decoding, making full use of the hardware resources for parallel processing becomes imperative.

Recently, graphics processing units (GPUs) have emerged as co-processing units for central processing units (CPUs) to accelerate various numerical and signal processing applications [6]-[7]. With the popularity of General-purpose computing on GPU, it is into hot spots accelerating the deblocking filter by using GPU. GPU has greater data transfer bandwidth, more computing units, and more powerful floating-point handling capability, which is very suitable for highly parallel computing [8]. NVIDIA CUDA (Compute Unified Device Architecture) is a widely used tool on GPU platform and is mature enough for parallel computation. Therefore, in this article, we propose an efficient parallel CUDA-based optimization algorithm of deblocking filter. It does not change the filter order of boundaries and win more than 8.5 speed-up ratio compared with CPU version.

The rest of this paper is organized as follows. An introduction of CUDA programming model is presented In Section 2. In Section 3, it describes the deblocking filter of AVS standard. We propose a CUDA-based parallel deblocking filter algorithm in Section 4. In Section 5, experimental results are shown and compared to CPU version. Finally, Section 6 draws the conclusion.

II. CUDA PROGRAMMING MODEL

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by NVIDIA [9]. It gives developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. CUDA is widely used in general purpose computation, such as astronomical calculation, computational fluid dynamics simulation, image processing and video codec.

CUDA programming model consists of two parts: Host-side and Co-processor-side (or Device-side). Generally, CPU is regarded as Host and GPU is regarded as Device. In a system, there is only one CPU and one or more GPUs, they work cooperatively and carry out their own tasks respectively. CPU is responsible for those serial execution modules; Initialization before activating a new kernel function and clean-up work of the last kernel function are also finished in it. While kernel function is executed in parallel in GPU. Fig.1 shows the CUDA programming model.

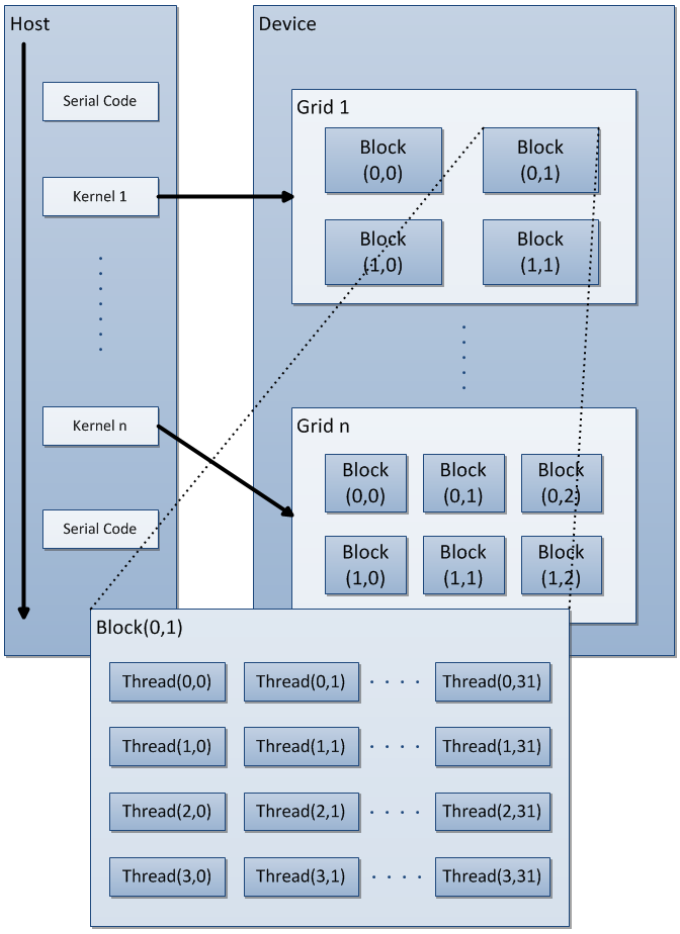

Figure 1. CUDA programming model

Firstly, serial code and initialization of kernel function is executed on Host-side. And then, Host-side activates several kernel functions in serial, for each kernel function, the Device-side starts a Grid, which contains a number of Blocks and each Block also includes a number of Threads. A kernel function is executed in parallel through these Threads. After the work done, Host-side cleans up the used kernel function finally. It involves several important concepts in CUDA programming model:

- Kernel: Kernel is the kernel function executed in GPU;
- Thread: Thread is the smallest unit of computing;
- Block: Block consists of multiple threads. Those threads in the same Block can access the same shared memory, and can be executed synchronously;
- Grid: Gird contains multiple Blocks.

III. AVS DEBLOCKING ALGORITHM

A. *Blocking artifact*

In AVS, a frame or a slice is divided into a great number of processing units named as Macro-block (MB) and each MB contains a fixed size of 16 x 16 pixels. Furthermore, a MB can be split into four 8 x 8 pixels size sub-blocks. Some processes are based on these sub-blocks, such as predictive coding, transformation and quantization. These processes, especially quantization and motion compensation, will cause error and bring loss in subjective and objective quality. Moreover, the lower the bit rate is, the more loss in quality it would be. This leads to discontinuity at the boundary of MBs and sub-blocks, which is called blocking artifact. In order to reduce blocking artifact, the in-loop deblocking filter has been introduced to AVS and applied in AVS decoder.

B. *Boundary strength calculation and Filter boundary judgement*

Before filtering, boundary strength (Bs) of each 8 x 8 pixels sub-block boundary needs to be calculated first [10]. There are three possible values for Bs and it is determined by prediction type, reference frame or motion vectors of the two neighboring sub-blocks next to the boundary. The derivation of Bs is shown in Fig.2. If at least one of the two sub-blocks next to the current boundary belongs to an intra-coding type MB, Bs of this boundary should be set to 2. Otherwise, both of them are inter-coding type, if their reference frames are different or the difference between their motion vectors is more than one pixel, Bs should be set to 1. Otherwise, Bs is set to 0. Bs decides the strength of filter, the bigger the Bs is, the stronger filter would be used. If Bs equal to 0, it means there is no need to filter current boundary.

After calculating Bs of all boundaries, it is necessary to judge whether a boundary is a real boundary or not as accurately as possible. AVS provides two boundary thresholds α and β to distinguish real boundary and fake boundary. α is boundary gradient and β is the gradient near a boundary. Looking up tables to get α and β, the index is calculated by average value of quantization parameters (QP_{av}) and filter offset parameters (OffsetA and OffsetB) of the two neighboring blocks. Generally speaking, pixel grad of pixels next to real boundary is larger than fake boundary. Therefore, whether a boundary is real or not is determined by comparing pixel grad with boundary thresholds α and β. Formula is shown in (1) and positions of pixel p_0, p_1, q_0, q_1 are shown in Fig.3. If value of (1) is true, which means that the current boundary is not real, then deblocking filter should be opened. Otherwise, shut down deblocking filter.

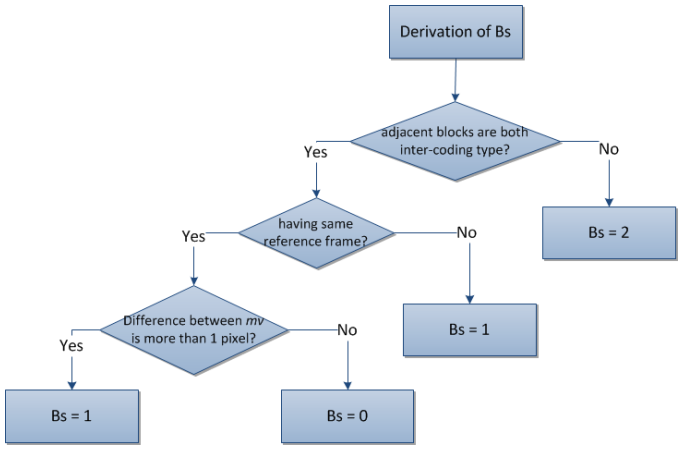

Figure 2. Derivation of boundary strength

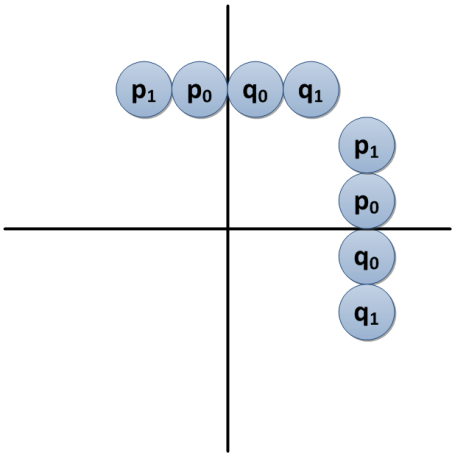

Figure 3. Pixels referred to when judging boundaries

$$Bs != 0 \;\&\&\; Abs(p_0 - q_0) < \alpha \;\&\&\; Abs(p_1 - p_0) < \beta \;\&\&\; Abs(q_1 - q_0) < \beta \quad (1)$$

C. *Filtering based on macroblock and Data dependence*

AVS deblocking filter is performed sequentially and its operating unit is MB. In MB layer, the process of deblocking filter is organized from left to right, from top to bottom. In a MB, firstly, vertical boundaries are filtered from left to right, and then horizontal boundaries are filtered from top to bottom. Take a MB of an image (4:2:0), which is shown in Fig.4, for example. For luminance component, firstly, vertical boundaries BV00, BV10, BV01, BV11 of the four 8 x 8 sub-blocks in current MB are filtered horizontally, and then horizontal boundaries BH00, BH01, BH10, and BH11 are filtered vertically. The length of luminance sub-block boundary is eight pixels, for each pixel, three pixels on each side of current boundary are used as input and no more than two pixels on each side need to be modified. As for chrominance component, there is only one 8 x 8 sub-block and filtering order is BV00, BV10, BH00, BH01, each boundary has only four pixels, two pixels on each side of current boundary are used as input and one pixel on each side need to be modified.

From the description above, it is apparent that there are large quantities of data dependence in the process of AVS deblocking filtering. Those pixels near a cross point of vertical boundary and horizontal boundary may be modified twice in a fixed order or used as input of another boundary. Some of the possible pixels are marked in Fig.4. For example, pixel q_1 should be filtered firstly with BV01 and then with BH11 of current MB.

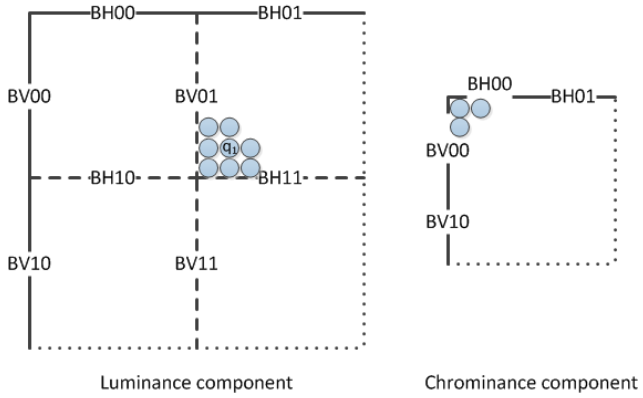

Figure 4. Filtering order in a Macro-block

IV. CUDA-BASED PARALLEL DEBLOCKING FILTER IMPLEMENTATION

As the analysis above, there is strong data dependence around the cross points of vertical and horizontal boundary. In a MB, the vertically filter to the horizontal boundaries must wait the execution results of the vertical boundaries in this MB. Moreover, because of the process of deblocking in MB layer is sequential, the previous MB affects the latter one. As a result, for luminance component, the parallelism only shows in 32 pixels of four vertical boundaries or four horizontal boundaries in the same MB, while chrominance component is much lower than luminance component. Obviously, the current deblocking filter is not suitable for parallel filtering on CUDA platform and its low parallel degree make it impossible to take full advantage of the massively parallel architecture GPU. Accordingly, we propose a new 8x8 block-based method to make deblocking filter suitable for parallel implementation.

A. *8 x 8 Intersection-block partitioning and Intersection-block based deblocking filter algorithm*

MBs strategy deblocking filter has strong data dependence and is not suitable for parallel processing. However, whether the data dependence occurs between adjacent MBs or inside a MB, it is located around the cross point of vertical boundary and horizontal boundary, couple with the situation that pixels of same boundary can be filtered in parallel, a new parallel partitioning mode is proposed as follows: dividing a frame into several 8 x 8 blocks, which is called Intersection-block, instead of 16 x 16 MBs. In each Intersection-block, there are four 4 pixels long boundaries to be filtered. Fig.5 shows the proposed Intersection-block partitioning mode. $IB_{(i,j)}$ (i, j = 1,2,3,…) represents the Intersection-block located in row i and column j. b_1, b_2, b_3, b_4 are boundaries in $IB_{(i,j)}$.

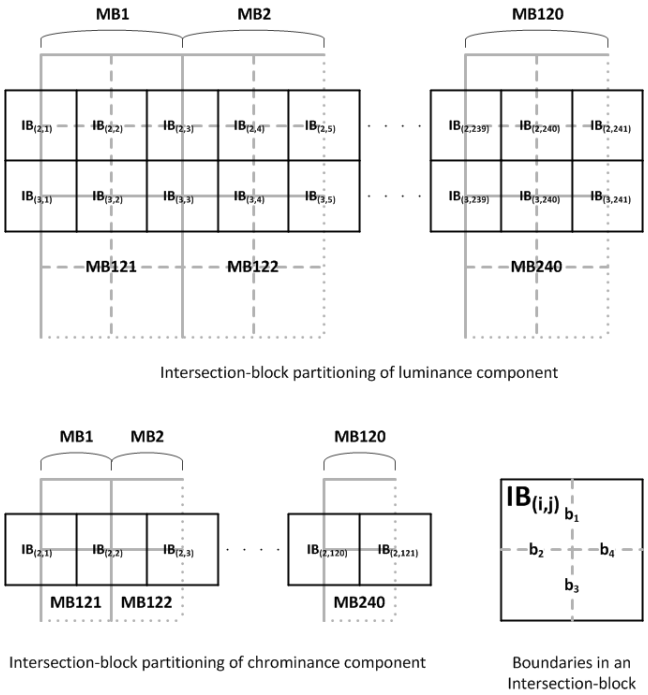

Figure 5. Intersection-block partitioning mode

Intersection-block partitioning mode hide the data dependence inside an 8 x 8 block, in which, boundaries are filtered sequentially. In Intersection-block layer, there is no data dependence between adjacent blocks. Therefore, the process of deblocking filter can be organized in parallel and its operating unit is Intersection-block. Take a frame of 1920 x 1080(4:2:0) video for example, there are 120 * 68 MBs, which means in the proposed partitioning mode, for luminance component, a frame has 241 * 137 Intersection-blocks and its parallel degree achieves 33017.

In order to keep the same quality of videos compared with CPU version, Intersection-block partitioning doesn't change the filtering order. For that reason, in luminance component, there are three possible filtering orders for these four boundaries in an Intersection-block and it is determined by the location of current Intersection-block. Fig.6 describes the filtering orders of three different kinds of Intersection-block. The number means the order of boundaries to be filtered in a sequential way. If an Intersection-block is located in odd-numbered row and odd-numbered column, such as $IB_{(2n+1,2n+1)}$(n=0,1,2,…), boundary b_1 is filtered first, then boundary b_2 and boundary b_3 is filtered sequentially after that, boundary b_4 is filtered last; If an Intersection-block is located in even-numbered row and odd-numbered column, such as $B_{(2n,2n+1)}$(n=0,1,2,…), boundaries are filtered in the following order: b_2, b_1, b_3, b_4. Otherwise, the filter order is b_1, b_3, b_2, b_4.

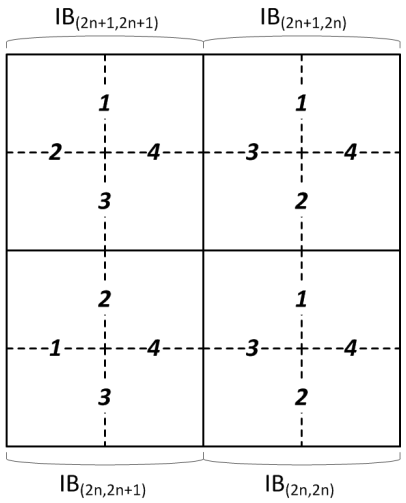

Figure 6. Three different filtering orders of Intersection-block

B. *Implementation on CUDA platform*

Intersection-blocks strategy deblocking filter has no data dependence in Intersection-block layer, so the whole frame can be filtered in parallel. But due to the different filtering process and parallel degree of luminance component and chrominance component, two kernel functions are employed to deal with filter of these two parts respectively. For luminance component, each CUDA Block contains 256 threads, which is one of the classic CUDA Block size. Each thread processes an Intersection-block filtering. Grid size depends on the resolution of video. For chrominance component, CUDA Block size is 128 threads and each thread also processes an Intersection-block filtering. The CUDA kernel functions are shown as follows:

1) luma_deblock<<<luma_dimGrid,256>>>(d_buf_y, d_buf_bs): which starts threads to filter luminance component, where luma_dimGrid is the number of CUDA Block;

2) chroma_deblock<<<chroma_dimGrid,128>>>(d_buf_u, d_buf_v, d_buf_bs): which starts threads to filter chrominance component, where chroma_dimGrid is the number of CUDA Block.

On CUDA platform, those threads in the same CUDA block are synchronized. But as the analysis above, an Intersection-block has three possible filtering orders, which means that there are branches in the process of filtering and leads to inefficient usage of the stream processor (SP). In response to this situation, several index matrixes are used to describe the filtering order and stride of adjacent pixels of current Intersection-block. Through this way, the code of 3 different Intersection-blocks filtering is unified and the performance of CUDA-based debloking filter can be improved. Following are the experiment results in detail.

V. EXPERIMENTAL RESULTS

A. Experimental platform

The following development environments are used to evaluate the performance of the implemented Intersection-block based deblocking filter:

Hardware configuration:

- Intel Core i5-4590 3.3GHz CPU;
- NVIDIA GeForce GTX 760 1019MHz/6008MHz 2GB/256Bit;
- 8GB(4GB * 2) memory;

Software platform:

- Microsoft Windows7 sp1 64bit;
- Microsoft Visual Studio 2008;
- CUDA Toolkit version 6.5.14 for Windows7 64bit;
- AVS1-P2 decoder;

B. Experimental results

In this work, six video sequences with size of 1920 x 1080 (4:2:0) are used to test the implementation. They are Kimono, ParkScene, Beach, TaiShan, Cactus and BasketballDrive. We get the execution time of Intersection-block based deblocking filter on CUDA GPU and compare it with the original CPU version. Table.1 lists the results and Fig.7 shows the improvement of these six videos. From Table.1 and Fig.7, we can see that the proposed parallel deblocking filter is practical and achieves high efficiency.

VI. CONCLUSIONS

In this paper, we propose a CUDA-based parallel algorithm for AVS deblocking filter called Intersection-block-based deblocking filter. By dividing a frame of image into several 8 x 8 Intersection-blocks, we make the filter suitable for parallel execution without changing the filtering order. We implement this proposed algorithm on GPU using CUDA. Experimental results show that our parallel deblocking filter achieves 8.52 times improvement at most on efficiency.

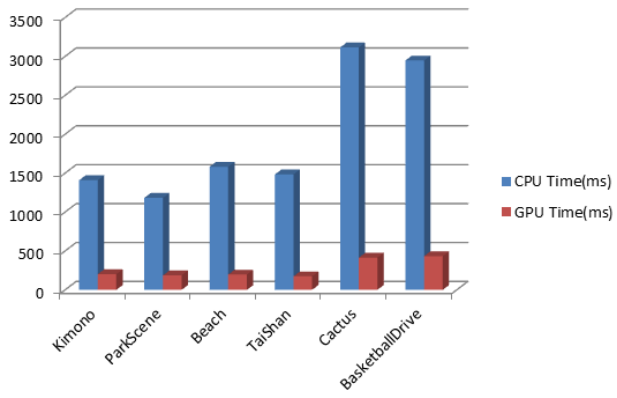

Figure 7. Comparison of execution time

REFERENCES

[1] G. Sullivan, J.-R. Ohm, A. Ortega, E. Delp, A. Vetro, and M. Barni, "dsp Forum—Future of video coding and transmission," IEEE Signal Processing Mag., vol. 23, no. 6, pp. 76–82, Nov. 2006.

[2] L. Fan, S. Ma, and F. Wu, "Overview of AVS video standard," IEEE International Conference on Multimedia and Expo., vol. 1, pp. 423–426, Jun. 2004.

[3] AVS Working Group Website: http://www.avs.org.cn.

[4] T. Wiegand, G. 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, July 2003.

[5] Peter List, Anthony Joch, Jani Lainema, Gisle Bjøntegaard, and Marta Karczewicz, "Adaptive deblocking filter", IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 614-619, July 2003.

[6] G. Shen, G. Gao, S. Li, H. Shum, and Y. Zhang, "Accelerate video decoding with generic GPU," IEEE Trans. Circuits Syst. Video Technol., vol. 15, no. 5, pp. 685–693, May 2005.

[7] O. Fialka and M. Cadik, "FFT and convolution performance in image filtering on GPU," in Proc. Conf. Information Visualization., pp. 609–614, July 2006.

[8] Huifang Deng, Chunhui Deng, and Jingjing Li, "GPU-based real-time decoding technique for high-definition videos," Intelligent Information Hiding and Multimedia Signal Processing., pp. 186-190, July.2012.

[9] NVIDIA CUDA Website: http://www.nvidia.com/cuda.

[10] Lu Yu, Sijia Chen, and Jianpeng Wang, "Overview of AVS-video coding standards," Signal Processing: Image Communication., vol. 24, issue 4, pp. 247-262, April. 2009.

TABLE I. EXECUTION TIME OF DEBLOCKING FILTER

Test Sequences	*Kimono*	*ParkScene*	*Beach*	*Taishan*	*Cactus*	*BasketballDrive*
CPU Time(ms)	1409	1182	1582	1483	3115	2947
CUDA Time(ms)	201	186	196	174	414	433
Speed-up Ratio	7.01	6.35	8.07	8.52	7.52	6.81