PARALLELING VARIABLE BLOCK SIZE MOTION ESTIMATION OF HEVC ON CPU PLUS GPU PLATFORM

Xiangwen Wang[1], Li Song[2], Min Chen[1] and Junjie Yang[1]

[1]Institute of Electronic and Information Engineering, Shanghai University of Electric Power
[2]Institute of Image Communication and Network Engineering, Shanghai Jiao Tong University
wxw21st@gmail.com, song_li@sjtu.edu.cn, chenm003@163.com, iamyjj@163.com

ABSTRACT

The emerging HEVC standard supports up to 12 variable block sizes ranging from 4x8/8x4 to 64x64 to conduct motion estimation (ME) and motion compensation (MC). This feature contributes considerable coding gain compared with 7 variable block sizes in H.264/AVC at the cost of huge computational complexity. In the test model HM, ME with variable block sizes (VBSME) may be called up to 425 times for the mode decision procedure of one CTU (Coding Tree Unit). Obviously, VBSME becomes the bottleneck for real time encoding. In this paper, we focus on parallel realization architecture design of VBSME in HEVC. Firstly, an efficient parallel encoder framework is proposed for CPU plus GPU platform. With the framework, VBSME, fractional-pixel image interpolation and border padding processes run on GPU without burden on the host CPU. Secondly, for workload balance between CPU and GPU, a fast Prediction Unit partition mode decision algorithm is also proposed. Lastly, the parallel realization strategy of VBSME on GPU is improved for ME compression performance improvement. Experimental results based on the NVIDIA's C2050 GPU show that the speed of the VBSME strategy on GPU is about 113 times faster than the one on CPU.

Index Terms— HEVC, Motion Estimation, GPU

1. INTRODUCTION

With the increasing demands of HD video and the emerging beyond-HD video, huge traffic is becoming a severe challenge for communication networks. The emerging video coding standard HEVC aims at providing a double compression performance improvement than H.264 mainly targeting at HD and beyond-HD resolution video.

This work was supported by National 863 project (2012AA-011703), National Key Technology R&D Program (2013BAH53F-04), NSFC (61221001, 61202369, 61271221), the 111 Project (B07022) the Shanghai Key Laboratory of Digital Media Processing and Transmissions and Shanghai Technology Innovation Project (10110502200,11510500900).

The ongoing HEVC coding standard inherits the well known hybrid ME/MC followed by transform and entropy coding framework adopted by H.261 since 1994 [1, 2]. In recent 20 years, the hybrid framework has been extended by dozens of new coding options. The inter frame prediction coding contributes substantial compression gains because of the high temporal correlation in video sequences. To improve the inter frame prediction precision, the block size for ME and MC has been extended from 8x8 to a series of combinations of block sizes. 8x8 and 16x16 block sizes are permitted for ME and MC in MPEG-2. In H.264/AVC, 7 variable block sizes ranging from 4x4 to 16x16 are employed. HEVC supports 12 variable block sizes ranging from 4x8 and 8x4 to 64x64. By enabling variable block sizes for ME and MC, the compression performance can be improved significantly, meanwhile, the encoding computational complexity increases dramatically. Since it is difficult and very complex to decide the most efficient PU combination for one CTU. The huge computationally intensive nature may prevent the HEVC from prevalent video communication implementations based on computation-limited platform.

Nowadays, personal computers are typically equipped with powerful but cost-effective Graphics Processing Unit (GPU). To accelerate the complex VBSME process, several works have targeted to accelerate VBSME on the powerful GPU with Compute Unified Device Architecture (CUDA) platform. Chen et *al*. presented an efficient block-level parallel algorithm for VBSME for H.264/AVC on CUDA platform in [3]. Pieters et *al.* described a flexible architecture and a task scheduling mechanism on CPU plus GPU system [4]. Chi-Wang Ho et *al*. rearranged the ME sequence of the 4x4 blocks in the wavefront format to overcome the dependency problem in the MV selection process [5].

In HEVC, the block sizes for ME and MC have been extended from 7 variable block sizes in H.264/AVC to 12. The basic coding unit has also been changed. So the VBSME mechanism as well as ME and MC block size selection are different from that of H.264/AVC. Therefore, the solutions proposed by the references can not be used directly for HEVC. Additionally, to the best of the authors'

knowledge, no integral framework for parallel accelerating the VBSME for CPU plus GPU platform has been published. In this paper, we propose an efficient parallel encoding framework for such CPU plus GPU platform. This framework has the following excellent features: 1) The VBSME and fractional-pixel images interpolation tasks are allocated to the GPU and they run in parallel with mode decision, MC and entropy coding; 2) The MVs gotten by VBSME are flexibly applied to PU partition mode decision; 3) The MVs dependency is considered in ME process on GPU.

The following sections of the paper are organized as follows. Section 2 gives an overview of VBSME as well as mode decision process in HEVC. The parallel encoding framework on CPU plus GPU platform is provided in section 3. Section 4 designs the parallel realization of VBSME on CUDA platform. Some experimental results are presented in Section 5. Finally, Section 6 concludes this paper.

2. OVERVIEW OF VBSME IN HEVC

In the HEVC, one frame is divided into a series of non overlapped Coding Tree Unit (CTU). The size of the luma block in a CTU is 64x64. The basic encoding unit is called Coding Unit (CU). It is always square and it may take a size from 8x8 luma samples up to the size of the CTU. The Prediction Unit (PU) is the basic unit for carrying the information related to the prediction process. It is not restricted to being square in shape. Each CU may contain one or more PUs, each of which may be as large as the CU or as small as 8x4 or 4x8 in luma block size. From the latest working draft 9 [1], we can get the allowed CU depth and PU partition structure as shown in Fig1. AMP partition modes are not supported by the main profile. There are 4 CU sizes from 64x64 to 8x8. The total number of allowed PU size is 12 (from 64x64 to 4x8/8x4), consequently, there may be up to 425 (5+4x5+16x5+64x5 = 425) times ME for one CTU to find the best combination of CU sizes and PU partition modes. Obviously, VBSME becomes the bottleneck for real time encoding. In the following, we give a brief description of the mode decision procedure which incorporates MV searching for blocks of different sizes, CU depth decision and PU partition mode decision.

The mode decision procedure can be divided into two stages: 1) ME and 2) CU depth and PU partition mode decision. In the first stage, the best MVs of each permitted PU are found. The criterion for selecting the best MV is the cost function:

$$J_{pred,SAD} = SA(T)D + \lambda_{pred} * R_{pred} \quad (1)$$

where R_{pred} is the approximate rate of MVs and is obtained by a pre-calculated table. *SAD* (Sum of Absolute Difference) is used for integer pixel ME while *SATD* (Sum of absolute Hadamard Transformed Difference) is used for fractional pixel. $\lambda_{pred} = \sqrt{\lambda_{\text{mode}}}$ and $\lambda_{\text{mode}} = \alpha * W_k * 2^{((QP-12)/3,0)}$, in which α is adjusted according to whether the current frame is used for reference. W_k represents weighting factor dependent on encoding configuration and *QP*. *QP* is the quantization parameter.

In the second stage, the RD costs of different CU sizes and PU partitions are evaluated and the combination which has the smallest RD cost is selected. The RD cost is expressed as:

$$J_{mode} = SSD + \lambda_{mode} * R_{mode,} \quad (2)$$

SSD is Sum of Square Difference between the original block and its reconstruction. R_{mode} denotes the rate required for representing the PU. It includes the bits required for signaling the coding mode and the associated side information, e.g. MV, reference indices as well as the bits required for the transform coefficient levels of the residual signal.

To calculate the J_{mode} for each PU, reconstruction and entropy coding of all syntaxes are necessary. If all 254 PU size combinations are checked, the complexity is beyond the computational capability of common computers for real applications.

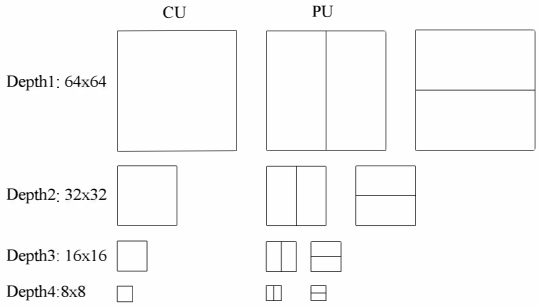


Fig. 1. CU size and PU partition structure

3. CPU PLUS GPU PARALLEL ENCODING FRAMEWORK

The framework of the encoder based on CPU plus GPU platform is diagramed in Fig. 2. The HEVC encoder is mainly divided into six components: VBSME, Mode Decision, MC, Interpolation, Encode & Reconstruction and Entropy coding. The VBSME and Interpolation components, i.e. the right part in Fig. 2, run on GPU, the others run on CPU. There are two loops in the parallel encoding framework, the CTU line loop and the frame loop as shown in the right part of Fig. 2. The input sequence is coded frame by frame in the frame loop. One frame is divided into a series of non-overlapped CTU lines which are coded line by line in the CTU line loop. The Mode Decision, VBSME, MC and Encode & Reconstruction components process all the data of one CTU line in one pass, while Entropy coding and Interpolation components process all the data of one frame in one pass.

The components on GPU and the components on CPU run in parallel by two *Synchronization* components: Sync to

Interpolate and Sync to Last CTU Line ME. The CPU launches the Interpolation task to the GPU after the CTU line loop finishes one frame and copies the reconstruction image to the GPU global memory. When the CPU executes Entropy coding, the GPU runs Interpolation and boundary padding tasks for fractional-pixel images. When CPU launches a new CTU line ME task to the GPU, the fractional-pixel interpolation and boundary padding should have been finished. It is guaranteed by Sync to Last Interpolate component. Before the Mode Decision for the CTUs in one CTU line begins, the MV information should have been ready. This is guaranteed by the Sync to Last CTU Line ME component. When CPU executes Mode decision, MC and Encode & Reconstruction components for CTUs in one CTU line, the GPU run VBSME for the next CTU line. Using the two Sync components, fractional-pixel interpolation followed by boundary padding and entropy coding can run in parallel. Meanwhile, the VBSME for the next CTU line and mode decision followed by MC and Encoding & Reconstruction is capable of running in parallel.

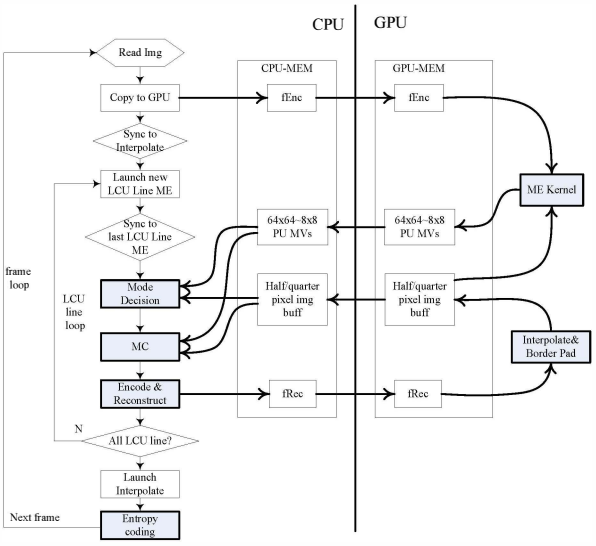


Fig. 2. Parallel encoding framework on CPU plus GPU platform. The thick lines with arrows represent the dataflow directions.

As described in Section 2, to calculate the RD cost for mode decision, PU reconstruction and entropy coding of all syntaxes are necessary. Mode decision becomes the bottleneck in our parallel encoding framework. For workload balance between CPU and GPU, we propose a fast PU partition scheme to speed up mode decision. Before mode decision for one CTU, the MVs for PUs with 10 kinds of block sizes are available. The motion information can be flexibly employed for PU partition decision [6]. In the following, we describe the scheme as shown in Fig. 3. It is composed by the following steps.

Step 1. The SKIP detection and the CBF_fast detection algorithms employed in HM are also adopted. If none of the SKIP and CBF_fast conditions is true, proceed to step 2. Otherwise, set CU depth to 4 and jump to step 5.

Step 2. If the max difference between four MVs of four PART_NxN PUs and MV of the PART_2Nx2N is not larger than 6, which means one and a half pixel distance between these two PU MVs, the CU is coded as PART_2Nx2N. Jump to step 5. Otherwise, set CU depth to 4 and proceed to step 3.

Step 3. Do motion compensation for PART-2Nx2N PU with the MV and calculate the texture pattern of the residual block. To calculate the texture pattern, one 2Nx2N block is divided into four NxN blocks. Let S_{00}, S_{01}, S_{10} and S_{11} denote the average intensity of each NxN block starting from left top and in raster scanning order. Two edge feature parameters: vertical edge parameter V and horizontal edge parameter H are introduced.

$$V = \left\lfloor \frac{\left|(S_{00} - S_{01}) + (S_{10} - S_{11})\right|}{N \times \left(QP\_step / 8\right)} \right\rfloor \qquad H = \left\lfloor \frac{\left|(S_{00} - S_{10}) + (S_{01} - S_{11})\right|}{N \times \left(QP\_step / 8\right)} \right\rfloor \tag{3}$$

$\lfloor \bullet \rfloor$ represents the floor function. QP_step is the quantization step and N represents the block size of the PART_NxN. According to H and V, CU partition of the 2Nx2N block is decided as follows:

If H==V and H!=0, which means no obvious edge, PART_2Nx2N is selected, and set CU depth to 4 and jump to step 5.

else if H==V and H==0, which means Diagonal edges exist, PART_NxN is selected;

else if H>V, which means Horizontal Dominant edges exist, PART_Nx2N is selected;

else, PART_2NxN is selected.

Step 4. Calculate RD cost J_{mode} as equation (4). Compare and select the smallest one.

Step 5. Check if the current CU depth is 4 and all the four NxN blocks have been processed. If the condition is true, proceed to step 6. Otherwise, jump to step 1.

Step 6. Mode decision for the CTU ends.

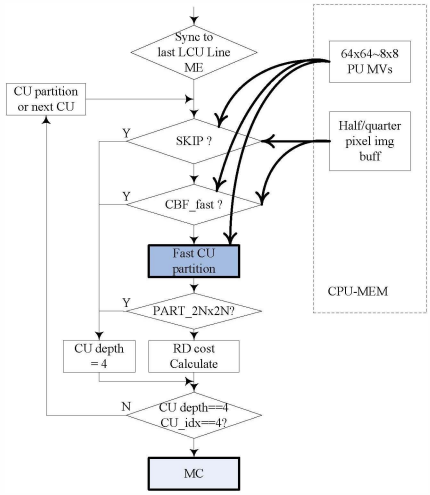


Fig. 3. The schematic of the fast PU partition algorithm

4. PARALLEL REALIZATION OF VBSME ON CUDA

The architecture proposed in [4] simply employs SAD as the criterion for MV selection. The MV dependency between neighboring block is neglected. It may induce RD performance degradation severely compared with the reference encoder. In our parallel encoding framework, the GPU executes the VBSME task for one CTU line in parallel. Therefore, the MVs of the top CTU line are ready when conducting the VBSME for the current CTU. In this section, the architecture proposed in [4] is improved by taking the MV dependency for MV selection into account. The flowchart of the improved VBSME on CUDA is shown in Fig. 4. The procedures of the improved architecture are composed of the following steps.

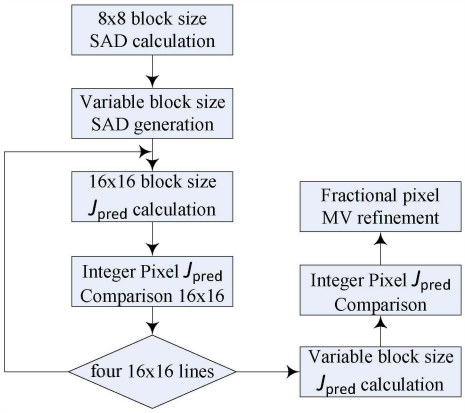


Fig. 4. Flowchart of the improved VBSME

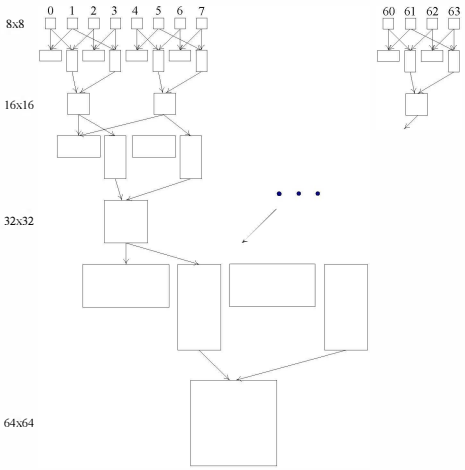


Fig. 5. Variable block size SAD Generation

Step 1. 8x8 block size SADs are calculated for all MV candidates;

Step 2. Variable block size SADs are generated by summing up the corresponding 8x8 block SADs as diagramed in Fig.5;

Step 3. MVs for 16x16 block size are selected using J_{pred} as the criterion. In this step, we divide one CTU line into four 16x16 block lines. The SAD comparison in MV selection for each 16x16 line is done sequentially. The J_{pred} is specified by the following formula.

$$J_{pred} = SAD + 2 * DMV = SAD + 2*(MV\_C - PMV) \quad (4)$$

where *MV_C* represents one of the MV candidates. *PMV* is the predicted MV which is calculated as shown in Fig. 6. As the MVs of the top CTU line are now available, the MV0, MV1 and MV2 of the top 16x16 PUs as well as the MV3 and MV4 of the last frame PUs are employed for MV prediction.

Step 4. The J_{pred} for all the other variable block sizes are calculated as the procedure for 16x16 size. In our architecture, the MVs gotten by 16x16 PU size in step 3 are employed for all variable block size MV predictions.

Step 5. Integer pixel J_{pred} comparison to get IMVs for all variable block sizes.

Step 6. Fractional pixel MV refinement.

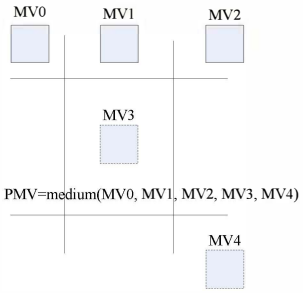


Fig. 6. MVP prediction schematic

5. EXPERIMENTAL RESULTS

The workstation Z620 produced by Hewlett-Packard is selected as the CPU plus GPU platform. It contains one NVIDIA Tesla C2050 GPU. The Operating System is 64bit Windows7. There are 448 CUDA cores at clock rate 1.15G Hz. The CUDA driver version of the GPU is 5.0 and the CUDA Capability version number is 2.0. Some preliminary experimental results of the parallel realization of VBSME on CUDA and the parallel encoder are presented.

Firstly, we evaluate the acceleration performance of the improved VBSME on GPU.

The improved VBSME runs on one CPU core and GPU, in parallel and serially, respectively, for acceleration performance comparison. The search range is 64x64 with the full search strategy for IMV and 24 fractional-pixel positions around the IMV. Two test sequences with different resolutions *BasketballDrive_1920x1080_50* and *Traffic_2560x1600_crop* are tested. The results are tabulated in Table I. For the 2560x1600 resolution sequence, the GPU version of VBSME achieves 23.77 fps, while the single CPU thread version only process 0.21 fps. The speed-up ratio is about 113 times. The same acceleration performance is obtained for the 1920x1080 resolution sequence. The ME time is picture independent due to full search.

The compression performance as well as the speed acceleration performance of our parallel encoder is compared with the anchor HM9.0 encoder [8]. The parallel encoding framework is transplanted into the latest x265 encoder, a first open source encoder implementation of HEVC [9]. We call the x265 encoder with the parallel framework as the proposed encoder. The main configurations are: 1) the first 50 frames are encoded, one I frame followed by 49 P frames; 2) AMP is off; 3) SAO is off; 4) PCM mode is disabled; 5) TU size is set as the PU size except TU size is 32x32 for CU size larger than 32x32; 6) search range is 64x64; 7) EPZS fast search is enabled for the HM encoder while full search for the proposed encoder; 8) Rate control is disabled, six QPs 22, 24, 26, 28, 31 and 34 are tested; 9) input sequences are *BasketballDrive _1920x1080_50.yuv* and *Cactus_1920x 1080_50.yuv*.

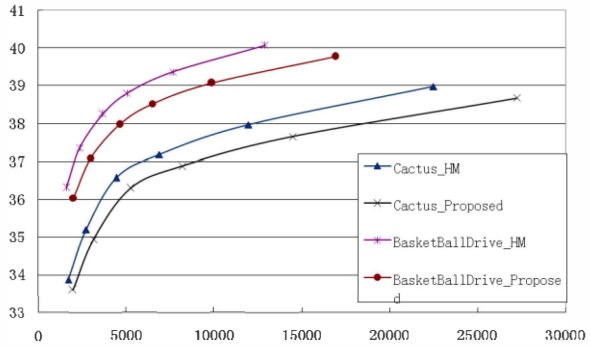
Table I. Speedup gains on different video sequences

sequence	CPU(fps)	GPU(fps)	Speedup ratio
Traffic_2560x 1600_crop	0.21	23.77	113.2
ParkScreen_1 920x1080_24	0.69	77.76	112.7

The RD curves by the HM encoder and the proposed encoder are shown in Fig. 7. From these RD curves, we see that the RD performance degradation of the proposed encoder with fast CU partition decision algorithm is about 0.7 dB degradation which is calculated followed by BD-PSNR[7]. The encoding speed of the HM encoder is about 0.03 fps while the average speed of the proposed encoder is 14.5 fps. The acceleration by the proposed mainly from the program re-designed by the x265 encoder. We also conduct the acceleration comparison between two x265 encoders, one without GPU acceleration and another with GPU acceleration using the proposed framework. There is about 71 times faster by the GPU parallel acceleration. The RD performance degradation is mainly induced by the fast CU partition mode decision. With the fast PU partition mode decision, instead of calculating four RD costs, only one RD cost is calculated for the CU size decision. Despite of the RD performance degradation, the real-time encoding speed makes the parallel encoder promising for practical applications.

6. CONCLUSION

In this paper, we design an efficient parallel framework of the HEVC encoder on CPU plus GPU platform. In the framework, a novel synchronization mechanism between CPU and GPU is proposed, a fast PU partition scheme is proposed and the procedure of the VBSME on GPU is improved. In the parallel framework encoder, motion estimation, fractional-pixel image interpolation and border padding processes run on GPU without occupying CPU cycles. Significant computational complexity is saved by the fast PU partition scheme. Experimental results show that the speed of the proposed VBSME strategy on GPU is about 113 times faster than the one running on CPU platform. The RD performance degradation induced mainly by the fast CU partition scheme is about 0.7 dB.



7. REFERENCES

[1] B. Bross, et al., "High efficiency video coding (HEVC) text specification draft 9," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-K1003, October 2012.

[2] K. Kim, et al., HM9: HEVC Test Model 9 encoder Description, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-K1002, October 2012.

[3] W.-N. Chen and H.M. Hang, "H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA)", International Conference on Multimedia and Expo (ICME 08), pp. 697-700, 2008.

[4] B. Pieters, et al., Motion estimation for H.264/AVC on multiple GPUs using Nvidia CUDA. In Applications of Digital Image Processing XXII, volume 7443, page 74430X, September 2009.

[5] Chi-Wang Ho, O.C. Au, Gary Chan, Shu-Kei Yip, Hoi-Ming Wong: Motion estimation for h.264/avc using programmable graphics hardware. In: IEEE International Conference on Multimedia and Expo. (2006) 2049-2052

[6] X. W. Wang, J. Sun, Y. Q. Liu and R. J. Li, "Fast Mode Decision for H.264 video Encoder Based on MB Motion Characteristic,", 2007 IEEE International Conference on Multimedia and Expo(ICME 07), pp. 372-375, 2007.

[7] G. Bjøntegaard, Calculation of Average PSNR Differences Between RD-Curves (VCEG-M33), VCEG Meeting (ITU-T SG16 Q.6), Apr. 2001.

[8] https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware.

[9] x265 project, http://code.google.com/p/x265/