# PARALLELING VARIABLE BLOCK SIZE MOTION ESTIMATION OF HEVC ON MULTI-CORE CPU PLUS GPU PLATFORM

*Xiang-wen Wang, Li Song, Min Chen, Zheng-yi Luo And Jun-jie Yang*

Shanghai University of electric power and Shanghai Jiaotong Univerity

## 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 significant coding gain compared with 7 variable block sizes in H.264/AVC at the cost of huge computational complexity. In the test model HM, motion estimation with variable block sizes (VBSME) may be called up to 254 times for mode decision of one coding tree unit (CTU). Obviously, VBSME becomes the bottle-neck for real time encoding. In this paper, we focus on parallel architecture design of VBSME. An novel parallel encoder framework is proposed for multi-core CPU plus GPU platform. A two stages motion estimation (ME) is proposed for parallel acceleration with little compress performance degradation. With the parallel framework, ME, fractional-pixel image interpolation and border padding processes run on GPU without burden on the host CPU. Experimental results show that the speed of the HEVC encoder with the proposed strategies is about 28.6 fps for 1080P videos.

***Index Terms*—**HEVC, Motion Estimation, GPU

## 1. INTRODUCTION

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. 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 multi-core CPU and powerful Graphics Processing Unit (GPU). A novel wavefront parallel strategy for multi-core CPU acceleration is proposed in [6]. 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].

Almost all the parallel acceleration strategies for VBSME proposed by previous references upon GPU platform don't consider the application scenarios with multi-core CPU encoder realization. In this paper, we propose an novel parallel encoding framework for multi-core CPU plus GPU platform. This framework has the following excellent features: 1) The encoder is parallel accelerated by multi-core CPU with multi-threads and by many-core GPU. The VBSME as well as fractional-pixel images interpolation and border padding tasks are allocated to the GPU and they run in parallel with mode decision, encoding, reconstruction and entropy coding; 2) Several best candidate MVs are found by GPU for each PU in different size in one CTU. The best MV for final encoding is decision by the Mode decision on CPU. This compensates the RD performance degradation induced by conventional parallel acceleration ME schemes on GPU.

The following sections of the paper are organized as follows. Section 2 gives an overview of VBSME and mode decision in HEVC. The parallel encoding framework on multi-core CPU plus GPU platform is provided in section 3. Section 4 proposes the two stages ME algorithm. The parallel VBSME on CUDA platform is described in Section 5. Some experimental results are presented in Section 6. Finally, Section 7 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 maximum 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, 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, and one CU can be split into three kind of PUs except for 8x8 CU size. The total number of allowed PU size combinations for one CTU is 254 (3+4x3+16x3+64x3 = 254). It means there may be up to 254 times of motion estimation for one CTU to find the best combination of 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.

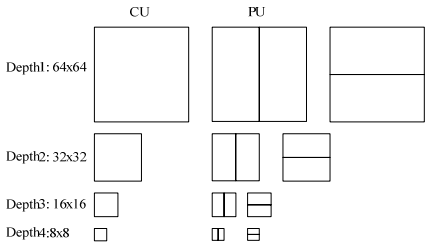


**Fig. 1.** CU size and PU partition structure

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 rate-distortion (RD) 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.

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. $\lambda_{pred}$ and $\lambda_{mode}$ are the lagrangian parameters.

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-time applications.

## 3. MULTI-CORE CPU PLUS GPU PARALLEL ENCODING FRAMEWORK

### 3.1 parallel encoding framework for Multi-core CPU plus GPU platform

As described in Section II, the VBSME is the most time-consuming part in the encoder. We specially divide one encoder into six models: VBSME, Mode Decision, Encoding & Reconstruction, Deblocking, fraction-pixel Interpolation and Entropy coding. In our framework, VBSME, deblocking and fraction-pixel Interpolation are parallel accelerated by the GPU. The others run on the multi-core CPU. The Mode Decision and the Encoding & Reconstruction run on multi-core CPU with multi-threads, while the Entropy coding is executed by the CPU with single thread. The proposed parallel encoding framework is diagramed in Fig. 2.

The first paragraph in each section should not be indented, but all following paragraphs within the section should be indented as these paragraphs demonstrate. There is two-level parallelization in the framework. The first level is the parallelization between CPU and GPU and the second is the multi-threads parallelization on multi-core CPU. Three *Synchronization* components guarantee the parallel execution: the WPP Sync, the Interpolate Sync and the ME Sync. The WPP Sync component takes charge of the multi-threads synchronization on multi-core CPU. The multi-thread on multi-core CPU parallelization is based on the wavefront Parallel Processing (WPP) strategy as proposed in [6]. In the wavefront parallelization, each frame is partitioned into LTU rows. All LTUs in the same LTU row will be processed by the same thread. LTUs from different CTU rows can be processed concurrently, only if its neighboring CTUs in its top CTU row have been encoded and reconstructed. This condition is guaranteed by the WPP Sync component.

There are two synchronization components for CPU and GPU parallelization: the Interpolate Sync and the ME Sync. the Interpolate Sync is inserted between the component Org Img copy to GPU and the component Launch ME. The CPU master thread (the ID of the master thread is 0 within Open MP) launches the Deblocking and the Interpolation asynchronous tasks sequentially to the GPU after all CPU threads finish reconstruction of 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. Before CPU master thread launches new VBSME for the current frame, the Interpolate Sync component guarantees that the boundary-extended fraction-pixel images of the last reconstruction frame are ready. The

ME Sync component is only employed by the master thread on CPU. This component takes charge of the synchronization between the VBSME kernel on the GPU and the components of Mode decision of each thread on the multi-core CPU.

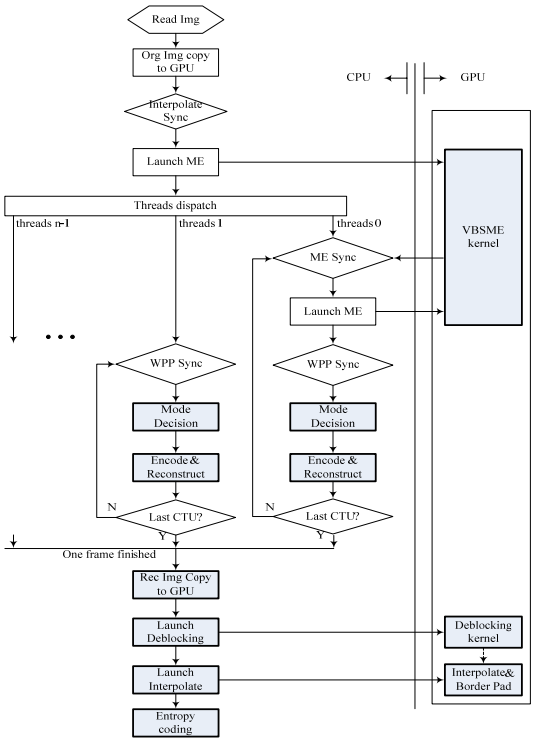


**Fig. 2.** The proposed parallel encoding framework

### 3.2 span wavefront VBSME strategy

With the current CUDA programming paradigm, the overhead of initializing the GPU for one kernel is heavy and the results calculated by GPU kernel is available for CPU only after all the threads on GPU have finished. Because of the heavy overhead of initializing the GPU, it would be inefficient to start a VBSME kernel for each CTU. At the same time, scheduling VBSME for all CTUs of one frame on the GPU means that the all host threads are waiting for encoding the first CTU. Precious CPU resources are not used. Upon these observations, we propose a span wavefront VBSME strategy to arrange the VBSME task on GPU accompany with WPP multi-core CPU acceleration.

An example of the proposed span wavefront sequence is illustrated in Fig. 3. There are altogether 30x17 CTUs in the image. We suppose 8 host threads in WPP multi-core CPU acceleration. The first stage of the span wavefront sequence processes the top left triangle part of CTUs in blue color. There are 64 CTUs in the first stage. The second stage processes the top middle parts of CTUs in red color. There are 8x16 CTUs. With the proposed task scheme, the VBSME kernel for one frame is launched several times which is far smaller than the number of CTUs in one image. Meanwhile, the MVs for the CTUs located in top left part of the image, belonging to the first stage of the span wavefront, are ready, so that the host threads can begin to encode these CTUs without waiting until to the end of the VBSME for the whole frame.

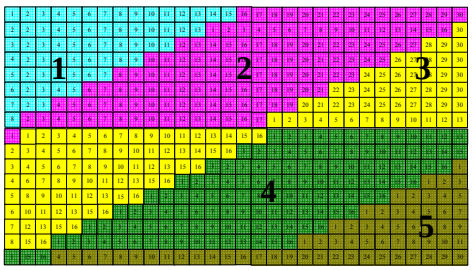


**Fig. 3.** Span wavefront sequence for one image

## 4. TWO-STAGE ME ALGORITHM

In the most VBSME architectures for parallel realization on GPU, only SAD is employed as the criterion for MV selection. To compensate the compression performance degradation induced by neglecting of MV dependency for coding, we propose a two-stage ME algorithm. In the first stage, ME is conducted employing SAD as the criterion for MV selection. In this process, a constant number, denoted as C, of the best MV candidates are saved. In the second stage, MV refinement is employed to select the best MV among the C best candidates using the $J_{pred,SAD}$ in equation (3) as the criterion. The number C becomes crucial in the algorithm. Based on a series of experiments, we found that the RD performance degradation is only about 1.3% when C equals 5. It achieves a good balance between computational complexity and RD performance when C is 3. Based on the two stage ME algorithm, we deploy the first ME stage onto the GPU and the second stage onto CPU. The following section describes parallel realization of the VBSME with full search pattern on GPU.

## 5. PARALLEL REALIZATION OF VBSME ON GPU

The only information needed for full search ME with fixed search range is the coordination of the block. The number of CTUs for one VBSME kernel and the coordinations of each block within these CTUs can be easily calculated in advance on CPU and saved in a table. The number of CTUs are used to decide the number of thread blocks. Before launching a new VBSME task, the table should be copied to GPU global memory. Each thread retrieves the block coordination from the global memory. The following are the setting options in our realization architecture.

1) The maximum CTU size is 64x64, the smallest block size for ME is restricted to 8x8. So one CTU is divided into 64 8x8 blocks;

2) The ME search range is set as 32x32. There are 1024 candidate MVs for each block. The thread block dim is (16,

16). Each candidate SAD for 8x8 blocks is computed by one thread.

3) The SADs of block sizes 8x16, 16x8, 16x16, … , 64x64 are calculated by the combinations of 8x8 block SADs as shown in Fig. 5.

4) All 1024 SADs of one block are compared and three candidates with the least SADs are chosen as the Integer-pixel Motion Vectors (IMVs). The comparisons of SADs are done in one thread block using reduction comparison mechanism as proposed in [7].

5) After three best IMVs for each block size PU have been gotten, 24 fractional pixels adjacent to each IMV are examined, as shown in Fig. 4. The gray square in the middle, white square and the white circles are the IMV, examined half pixels and examined quarter pixels, respectively. Three candidates with the least SADs are chosen as the final MV candidates. The three MV candidates as well as the corresponding SADs will be copied back to CPU for final best MV decision.

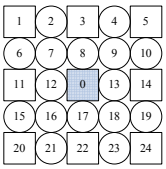


**Fig. 4**. 24 factional pixels candidates MVs

## 6. EXPERIMENTAL RESULTS

We re-write the HEVC encoder with C language, called as x265 encoder [9]. In the x265 encoder, several encoding options such as PCM, SAO, et al are omitted. The RD optimized mode decision is not employed and the cost function as defined in equation (2) is simply replaced with equation (1). The fast diamond ME algorithm with is employed. There is about 17% compression performance degradation induced by these simplifying. The x265 encoder with the proposed parallel strategy is called as the proposed encoder. The x265 with WPP multi-thread acceleration is called as the WPP x265 encoder. The compression performance as well as the acceleration performance of the proposed encoder is compared with the anchor HM10.0 encoder [8], the x265 encoder and the WPP x265 encoder. The main configurations for all encoders are: 1) first 50 frames are encoded, one I frame followed by 49 P frames; 2) AMP is off; 3) SAO is off; 4) intra prediction mode and PCM prediction mode for inter frames are 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 32x32; 7) EPZS fast search is enabled for the HM encoder while full search for the rest 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*. 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.

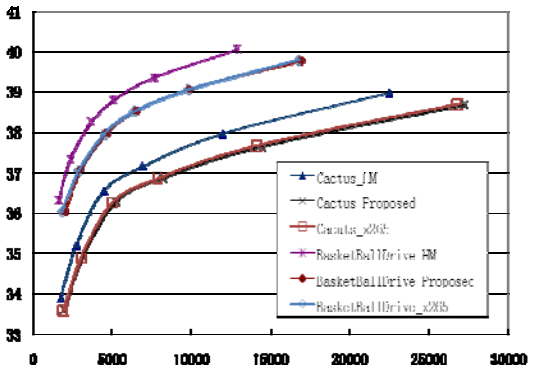


**Fig. 5.** RD curves by different encoders

The RD curves by these encoders are shown in Fig. 7. From these RD curves, we see that the RD performance degradation of the proposed encoder is about 0.73 dB degradation compared with the anchor encoder. The RD performance of the proposed encoder with two-stage ME algorithm is almost same with the x265 encoder. The average encoding speeds for different encoders are tabulated in Table I. The encoding speed of the HM encoder is about 0.03 fps while the average speed of the proposed encoder reaches about 28.6 fps. there is about 953 times speed up. The encoding speed of the WPP x265 encoder is about 8.3 fps with 8 threads. Compared with the WPP x265 encoder, there is about 3.4 times faster by the proposed encoder. Despite of the RD performance degradation, the real-time encoding speed makes the parallel encoder promising for practical applications.

**Table I**. Speed of different encoders

| sequence | The anchor encoder | The x265 encoder | The WPP x265 encoder | The proposed encoder |
|---|---|---|---|---|
| Cactus_1920x 1080_50.yuv | 0.03 | 2.4 | 8.3 | 28.6 |

## 7. CONCLUSION

In this paper, we design an efficient parallel framework for the HEVC encoder on multi-core CPU plus many-core GPU platform. The parallel framework has two excellent features. The first is that the VBSME task is allocated to the GPU and it synchronize flexibly with multi-core CPU WPP parallel encoding process. The second is that the ME process are divided into two stages which is suit for many-core GPU parallel acceleration without significant compression performance degradation. Experimental results show that encoding speed of the proposed parallel video encoder reaches 28.6 fps with about 0.73 dB RD performance degradation compared with the HM encoder.

## 8. 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),, 2008.

[4] B. Pieters, et al., “Motion estimation for H.264/AVC on multiple GPUs using Nvidia CUDA”. *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”. *IEEE International Conference on Multimedia and Expo.* 2049-2052. 2006.

[6] Z. Zhao and P. Liang. “Data Partition for Wavefront Parallelization of H.264 Video Encoder”. *ISCAS*. University of California, Riverside, 2006.

[7] Rob Farber. “CUDA application design and development”. *Elsevier*, Waltham, MA 02451, USA, 1011

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

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