Flexible CTU-level Parallel Motion Estimation by CPU and GPU Pipeline for HEVC

Juncheng Ma[1], Falei Luo[2], Shanshe Wang[1], Siwei Ma[1]
[1]*Institute of Digital Media, Peking University, Beijing 100871, China*
{[1]jcma, [1]swma}@pku.edu.cn
{[1]sswang@jdl.ac.cn}
[2]*University of Chinese Academy of Sciences, Beijing, China*
{[2]falei.luo@vipl.ict.ac.cn}

***Abstract*—In the high efficiency video coding (HEVC) encoder, motion estimation (ME) takes up more than 50% encoding time. To reduce the complexity of the ME module in HEVC, this paper proposes a flexible coding tree unit (CTU)-level parallel ME method through CPU and GPU pipeline collaboration. Firstly a highly scalable CTU-level parallel motion search scheme on GPU is provided, in which, the parallel CTU group can be configured to be any size to adapt to the variable sequence resolution and hardware configurations. Then, the motion search range can be adaptively adjusted based on the motion intensity. Therefore, the unnecessary GPU time wasting can be further avoided for slow-moving scenes, while high performance kept for fast-moving scenes. Moreover, the ME information returned from GPU can be used by CPU for fast mode decision. Experimental results show that the proposed method achieves up to 73% complexity reduction than HM10.0 anchor using CPU only with acceptable coding performance loss, providing higher performance than the state-of-the-art scheme.**

***Index Terms*—HEVC, Motion Estimation, GPU, CUDA, Search Range**

I. Introduction

The new published HEVC standard has achieved significant higher coding efficiency compared to the preceding standards, e.g. H.264/AVC [1][2]. However, the encoder complexity increases greatly mainly due to the motion estimation (ME) process for more variable coding block size. Specifically, the ME module takes up more than 50% encoding time of the HEVC encoder (Fig. 1).

ME is a block matching algorithm (BMA) performed at the encoder to find a matched prediction block in the reference frame. Generally, there are two kinds of search methods to get such a predicted block. The first one is full motion search by searching all points in a search window, which is simple but time-consuming. The second one is fast motion search by searching several points in several iterations, so it's faster than the first way and is more common used in software encoder. However, for hardware platform or some heterogeneous computing (e.g. CPU plus GPU computing), the first way is more suitable due to its regularity.

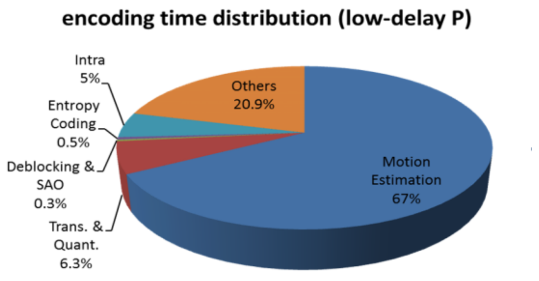


Fig. 1 Encoding time distribution of the HM encoder (LDP)

Recently, due to the rapid development of GPU processing capability, there has been a strong demand of using GPU as a co-processor to assist CPU to deal with data-intensive application [3]. Fortunately, NVIDIA has announced a programming friendly GPU architecture called "Compute Unified Device Architecture" (CUDA) [4] to make massive data parallel processing easier. Therefore, ME paralleling on CPU plus GPU platform has been widely studied. Chen et al. presented a block-level parallel scheme for the variable block size ME in H.264/AVC on CUDA platform [5], achieving 12 times faster than the CPU only scheme. Another ME acceleration method based on diamond search adaption on GPU for H.264/AVC was proposed by Schwalb et al. and got a significant reduction of computation time [6]. Wang et al. designed a parallel variable block size ME algorithm [7], but it can only process pixel-level ME line by line, and may bring in significant coding performance loss.

In this paper, we propose a flexible CTU-level parallel ME algorithm for HEVC by CPU and GPU cooperation. In the proposed scheme, variable-block-size prediction units (PUs) in a CTU are processed in only one CUDA thread block (Fig. 2), which is scheduled by GPU system, so the number of simultaneously processed CTUs can be configured to be any size. Then, the motion search range is adaptively adjusted each time when the GPU computing is launched, according to the motion intensity of the previous coded blocks. This strategy can further reduce ME complexity for the slow-moving scenes. Lastly, the motion vectors (MVs) and corresponding cost – sum of absolute difference (SAD), are returned by GPU and employed on CPU side to help mode decision.

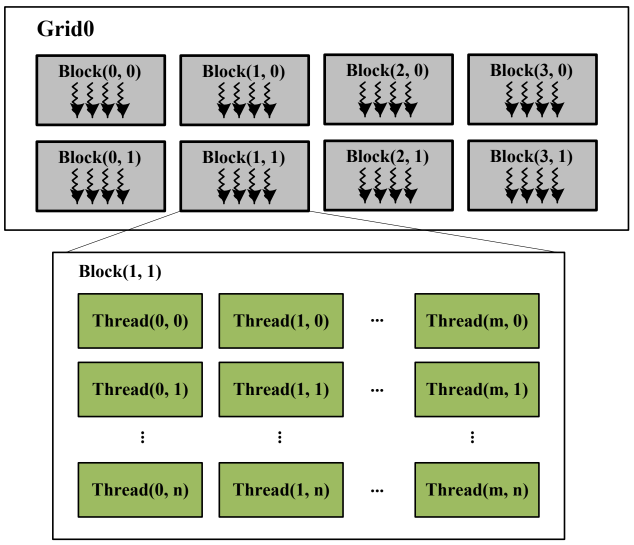


Fig. 2 Example of CUDA thread structure

The rest of the paper is organized as follows. Section 2 describes the ME of HEVC. Section 3 details the proposed ME paralleling method by CPU and GPU pipeline collaboration. Experimental result and analysis are presented in Section 4. Finally Section 5 concludes the paper.

II. OVERVIEW OF MOTION ESTIMATION IN HEVC

In the HEVC encoder, an inter-coded CU can be partitioned into multiple inter PUs, as illustrated in Fig. 3, and each PU has its own reference frame and corresponding motion vectors (MVs). The motion parameters of every PU are achieved by inheriting from spatial neighbour or temporal co-located PUs for "SKIP" or "MERGE" mode, or by invoking motion estimation on reference frames for other inter modes. Then, the mode with minimum rate-distortion (RD) cost is selected as the final coding mode for the CU. The RD cost is modelled as:

$$J_{mode} = SSD + \lambda_{mode} R \tag{1}$$

where *SSD* indicates sum of square difference between the original block and the prediction block, and *R* indicates the bits number used for coding the prediction residual and motion parameters. And λ_{mode} is the Lagrange multiplier that determines the trade-off between rate and distortion.

To get the best motion parameters for a specific inter mode, firstly the motion vector predictor (MVP) is derived from a set of MV candidates. Then for inter modes except for "SKIP" and "MERGE", motion search algorithm is performed at encoder. In HM encoder, integer-pel accuracy search is performed at first starting from the MVP, to select a best position based on SAD criterion (2). And as a second step, fractional-pel refinement is performed to generate the final quarter-pel accuracy MV based on sum of absolute transformed difference (SATD) criterion (3), as shown below:

$$J_{pred,sad} = SAD + \lambda_{pred,sad} R_{pred} \tag{2}$$

$$J_{pred,satd} = SATD + \lambda_{pred,satd} R_{pred} \tag{3}$$

where $\lambda_{pred,sad}$ and $\lambda_{pred,satd}$ is the Lagrange multiplier for SAD and SATD cost respectively, and R_{pred} is the approximate bit rate of MV. In addition, to generate fractional-pel samples, interpolation filtering is performed for the reference picture samples.

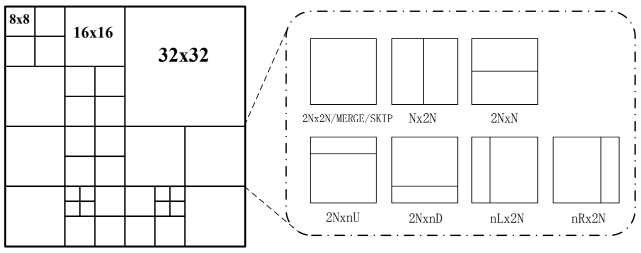


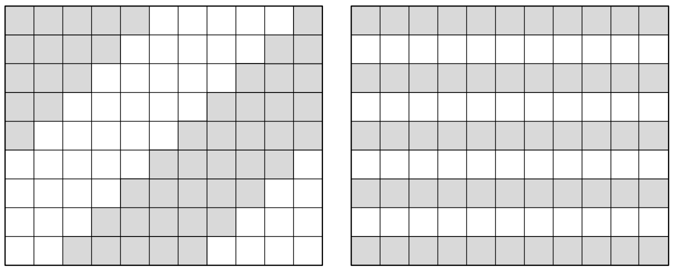
Fig. 3 Inter PU mode for CUs in variable depth

III. PROPOSED PARALLEL MOTION ESTIMATION SCHEME

The general idea of ME paralleling on GPU is dividing ME computing by MVs and PU sizes and allocating it to every running stream processing units, while CPU handling other encoder tasks synchronously. Hence this CPU and GPU pipeline framework can hide the ME coding time, directly causing encoding time saving. However, the highest parallelism degree depends on the hardware configuration, e.g. on-chip memory or cores number, and the previous ME paralleling methods [5-7] cannot expediently increase or decrease the parallel data size for targeted hardware due to the fixed data structure. Moreover, these methods may waste unnecessary GPU time for searching the best MV for slow-moving sequences due to the fixed search range. In our design, these problems can be solved by setting variable-size parallel CTU group (called CTU window below) and adaptive search range.

A. Variable Parallel CTU Window

Our method is implemented in CUDA programming architecture, in which threads are arranged in a block and blocks are arranged in a grid, as shown in Fig. 2. All threads in a block share the same "shared memory", and no matter how many blocks launched in a grid, they are automatically scheduled by GPU system. Based on this feature, we implement ME of one CTU in exactly one block, so the CTU window can be set to any length by launching corresponding number of blocks in a grid, as illustrated in Fig. 4.


a) CTU window length = 5 b) CTU window length = one CTU row
Fig. 4 Examples of CTU window configure

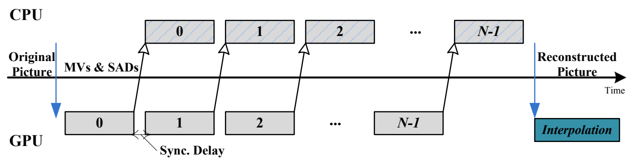


Fig. 5 CTU window pipeline between CPU and GPU

When the CTU window length is determined, every frame in the sequence is coded as a CTU window pipeline, as shown in Fig. 5, where *N* is the number of CTU window in a frame. At first the original picture is transmitted from CPU to GPU, then the parallel ME process of the first CTU window is launched. Next, CPU and GPU are synchronized for the MVs and SADs information of all PUs in CTU window *i*. After that, CPU executes mode decision and other encoder tasks, while GPU deal with CTU window *i*+1 simultaneously. Finally, when the last CTU window is finished by CPU, the whole reconstructed picture is transmitted to GPU for interpolation and saved as a new reference picture.

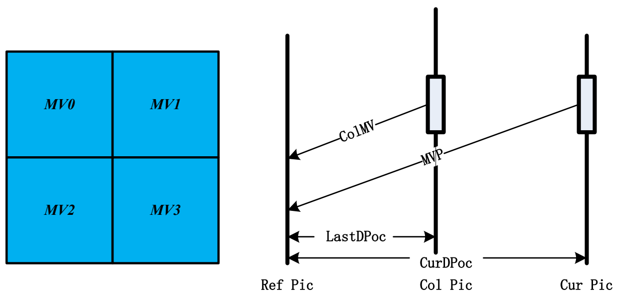
B. MVP Derivation and Motion Search

Before motion search on the GPU side, the MVP is derived at CTU-level from four 16x16 PUs in the temporal CTU by (4) and (5), as shown in Fig. 6:

$$ColMV = \frac{1}{4}\sum_{i=0}^{3} MV_i \qquad (4)$$

$$MVP = ColMV \times CurDPoc \,/\, ColDPoc \qquad (5)$$

where *ColMV* indicates the average MVs of four 16x16 PUs in the referenced CTU. *CurDPoc/ColDPoc* is the picture order count (POC) difference between the current/co-located picture and the reference picture, respectively. In addition, if the co-located picture does not exist, the above CTU is used instead.



a) MVs of four 16x16 PUs in a CTU b) temporal MV

Fig. 6 CTU-level MVP derivation

For integer-pel MV search, the variable-block-size SAD generation and comparison starting from 4x4 block is extended to more sizes from the method detailed in [5] for H.264/AVC. After that, fractional-pel MV refinement is performed around the best integer-pel MV position, just like the HM but concurrently by PU sizes.

C. Adaptive Motion Search Range

Full search scheme is adopted on GPU due to its regularity, and three targeted optimized search ranges (in integer-pel unit) are used: 8, 16 and 32. The smaller the search range, the lower complexity for parallel ME, but lower prediction performance either. The opposite result is got with the search range set larger. However, for slow-moving area such as background in a picture, small search range is enough.

Base on temporal relativity, the co-located blocks is used to prejudge an appropriate search range for the next CTU window to be processed. Each CTU in the CTU window is traversed, and for a certain CTU, define MVC as the MV of the PU containing luma pixel location (15, 15) relative to the top-left luma sample of the co-located CTU, and judge as below:

- Defaultly assign search range 8 for the current CTU window.
- If the partition depth of the co-located CTU is bigger than 0 and |MVC| is bigger than 16, then assign search range 16 for the current CTU window.
- If |MVC| is bigger than 32, then assign search range 32 for the current CTU window and exit the traversal.

D. GPU Returned Information-Assisted Mode Decision

The variable-block-size MVs and SADs information returned by GPU can not only provide motion parameters for ME, but also assist fast mode decision for inter-coded CUs. Here is the modification for the original HM mode decision.

1) Check MERGE/SKIP and PART_2Nx2N mode with early skip detection as before.

2) Define sad2Nx2N as the SAD of the 2Nx2N PU, sad2NxN/sadNx2N as the sum SAD of two 2NxN/Nx2N PUs of the current CU, respectively. If sad2NxN > t_0 * sad2Nx2N or sad2NxN > t_1 * sadNx2N, then skip PART_2NxN mode. The same goes for PART_Nx2N mode. The threshold t_0 and t_1 are empirically both assigned as 1.1.

3) For INTRA_2Nx2N mode, firstly calculate the SAD of the best-SATD intra mode selected from original rough mode decision (RMD) and most probable mode (MPM) procedure, named sadIntra2Nx2N. Then if sadIntra2Nx2N > t_2 * sad2Nx2N, skip Intra_2Nx2N mode. The threshold t_2 is empirically assigned as 0.5.

4) Select the best mode for the current CU based on rate-distortion cost as before.

IV. EXPERIMENTAL RESULTS

To verify the acceleration and compression performance of the proposed ME paralleling method, it has been implemented into HEVC reference software HM10.0. Simulations are conducted on a desktop with AMD Phenom (tm) II X4 830 CPU Processor @ 2.80 GHz plus NVIDIA GeForce GTX 560 Ti GPU. The CUDA driver version of the GPU is 5.5 and the CUDA capability version number is 2.1.

The experiment is performed under the low-delay P configuration of common test condition [8], except that the maximum CU size is set to be 32 and AMP is turned off. Full search with adaptive search range between 8, 16 and 32 is used for our proposed method and EPZS fast search with search range 64 for the anchor. Test sequences with different resolution, *Kimono, ParkScene, Cactus, Johnny, FourPeople,* *KristenAndSara, BQMall, BasketballDrill, BasketballPass and BlowingBubbles*, are examined with 10 seconds time interval. And the CTU window length is configured to be one row of CTU. The experimental results are shown in the Table 1.

TABLE I
PERFORMANCE AND TIME SAVING OF PROPOSED METHOD VS HM10.0

Sequence		Y	U	V	Time Saving
1080P	Kimono	1.5%	2.9%	2.7%	72%
	ParkScene	2.4%	2.3%	3.0%	68%
	Cactus	2.1%	1.7%	1.5%	69%
720P	Johnny	2.4%	3.2%	2.4%	73%
	FourPeople	1.5%	1.0%	2.0%	73%
	KristenAndSara	2.1%	2.9%	1.5%	73%
WVGA	BQMall	2.2%	2.8%	2.4%	70%
	BasketballDrill	2.2%	1.7%	2.0%	70%
WQVGA	BasketballPass	2.4%	2.3%	1.5%	70%
	BlowingBubbles	2.5%	2.4%	3.2%	62%
Average		**2.1%**	**2.3%**	**2.2%**	**70%**

From Table 1, it can be seen that the proposed parallel ME method can provide 70% complexity reduction on average with about 2.1% coding performance loss. To further compare the performance of the proposed algorithm with the anchor, the rate-distortion performance curves of some typical test sequences are shown in Fig. 7. It can be intuitively seen that, the proposed algorithm doesn't cause considerable objective coding quality degradation, compared to the significant 0.7 dB loss in the previous method [7].

We further verify the advantage of using adaptive motion search range by verify how much GPU utilization ratio (or GPU time) saved, as shown in Table 2, where the GPU utilization ratio is calculated as below:

$$Util\_ratio = GPU\_time \,/\, Enc\_time \qquad (6)$$

where *Enc_time* is the total encoding time while *GPU_time* is the total GPU running time. It can be seen from Table 2 that adaptive motion search range can effectively reduce the GPU utilization ratio down to less than 1/5 of fixed 32 search range with little extra performance loss for slow-moving sequences, and down to about 2/5 for fast-moving sequences.

V. CONCLUSIONS

This paper proposes a flexible CTU-level ME paralleling scheme on CPU plus GPU platform. The flexibility of this scheme lies in two aspects. Firstly, we arrange the CPU and GPU pipeline between CTU windows, whose length can be variably configured to apply to different hardware resources. Secondly, we propose a motion search range adjustment method based on the motion intensity of the co-located blocks. Experimental results show that our method can achieve significant coding complexity reduction with much smaller objective coding quality degradation than the state-of-the-art scheme, with extraordinary low GPU utilization ratio.

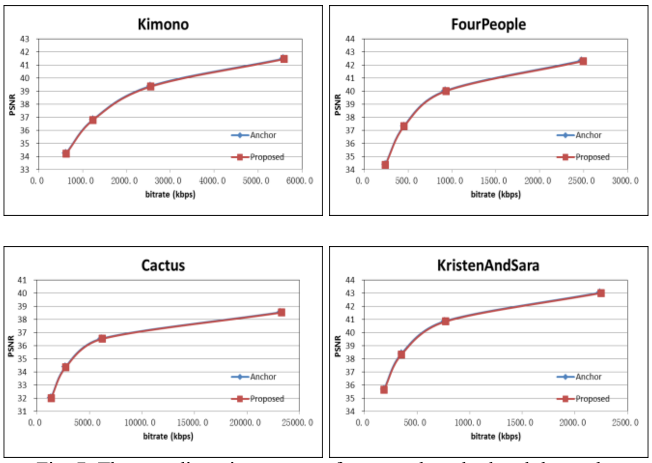


Fig. 7 The rate-distortion curves of proposed method and the anchor

TABLE II
GPU utilization ratio saving of adaptive motion search range

Sequence	Search Range = 32		Adapt.	
	Util_ratio	BD-rate	Util_ratio	BD-rate
FourPeople	6.2%	1.0%	1.2%	1.2%
Johnny	6.3%	1.9%	1.2%	2.1%
KristenAndSara	5.3%	1.7%	0.9%	1.6%
BasketballPass	4.4%	1.5%	1.8%	2.0%
RaceHorses	3.1%	2.3%	2.3%	2.4%

ACKNOWLEDGMENT

This work was supported in part by the National High-tech R&D Program of China (863 Program, 2012AA010805)，National Science Foundation of China (61322106, 61103088).

REFERENCES

[1] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, and T. Wiegand, “High efficiency video coding (HEVC) text specification draft 10,” Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, document JCTVC-L1003, Geneva, Switzerland, Jan. 2013.

[2] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification ITU-T Rec. H.264/ISO/IEC 14996-10 AVC), Mar. 2003.

[3] GPGPU. [Online]. Available: http://www.gpgpu.org/.

[4] NVIDIA, NVIDIA CUDA Compute Unified Device Architecture Programming Guide Version 6.0, 2014.

[5] W. N. Chen and H. M. Huang, “H. 264/AVC motion estimation implmentation on compute unified device architecture (CUDA),” Multimedia and Expo, 2008 IEEE International Conference on. IEEE, 2008.

[6] M. Schwalb, R. Ewerth and B. Freisleben. "Fast motion estimation on graphics hardware for H.264 video encoding," Multimedia, IEEE Transactions on 11.1 (2009): 1-10.

[7] X. W. Wang, L. Song, M. Chen and J. J. Yang, "Paralleling variable block size motion estimation of HEVC on CPU plus GPU platform," Multimedia and Expo Workshops (ICMEW), 2013 IEEE International Conference on. IEEE, 2013.

[8] F. Bossen, “Common HM test conditions and software reference configurations,” ITU-T SG16 Contribution, JCTVC-L1100, Geneva, Switzerland, Jan. 2013.