Multiple Layer Parallel Motion Estimation on GPU for High Efficiency Video Coding (HEVC)

Falei Luo*, Siwei Ma†, Juncheng Ma†, Honggang Qi‡, Li Su‡ and Wen Gao*†
Email: falei.luo@vipl.ict.ac.cn {swma,jcma}@pku.edu.cn {hgqi,suli}@ucas.ac.cn wgao@pku.edu.cn
*Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
†National Engineering Laboratory for Video Technology, Peking University, Beijing, China
‡School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing, China

***Abstract*—This paper provides a multiple-layer parallel motion estimation (ME) scheme implemented on GPU for High Efficiency Video Coding (HEVC). The scheme is hierarchically structured, including four layers: coding tree unit (CTU), prediction unit (PU), motion vector (MV) selection and instruction optimization. In PU-layer, costs of various PU sizes were obtained through a SAD (sum of absolute differences) look-up table instead of progressive cost merging. And during MV selection, GPU's comparison instruction was used to avoid branches. At the same time, concurrent CTUs processing and SIMD (Single Instruction, Multiple Data) optimization also improve the performance significantly. Experimental results show that the proposed scheme can take full advantage of GPU and achieves over 90 times speedup compared with the HM10.0 using fast ME.**

***Keywords*—HEVC, GPU, CUDA, ME, parallelization**

I. Introduction

The state-of-the-art HEVC shows superior compression efficiency compared to the preceding coding standards e.g. H.264/AVC [1], but the encoder complexity also increases significantly and makes it too complicated for real time applications. Researchers have been trying to optimize the reference test model HM by applying SIMD instructions [2] or conducting low complexity rate distortion optimization [3], but achieves limited time saving. In the encoder side, ME usually takes up more than half of the encoding time [3], and an efficient way to decrease the time consumption of ME is to utilize graphic processing unit (GPU).

GPU was firstly designed for graphics rendering and then becomes powerful in computations due to its structure with a high degree of parallelism. Nowadays most GPU supports general computations rather than only provides graphics processing APIs. For example, CUDA gives the developers a transition-easy way to code for NVIDIA's new GPUs [4]. At the same time, Open Computing Language (OpenCL) has been extensively used for accomplishing general computation tasks.

The feasibility of parallelization in ME inspired many researches in implementation on GPU. An early implementation of full search ME algorithm on CUDA was proposed by Chen which achieved about 10 times speedup [5]. Later many optimized implementations were published [6][7]. In [7], ME was optimized by Lee with concurrent parallel reduction and 90 times speedup was achieved for H.264/AVC. Wang proposed a full search ME approach on GPU with fast mode decision for HEVC [8], but significant loss was observed.

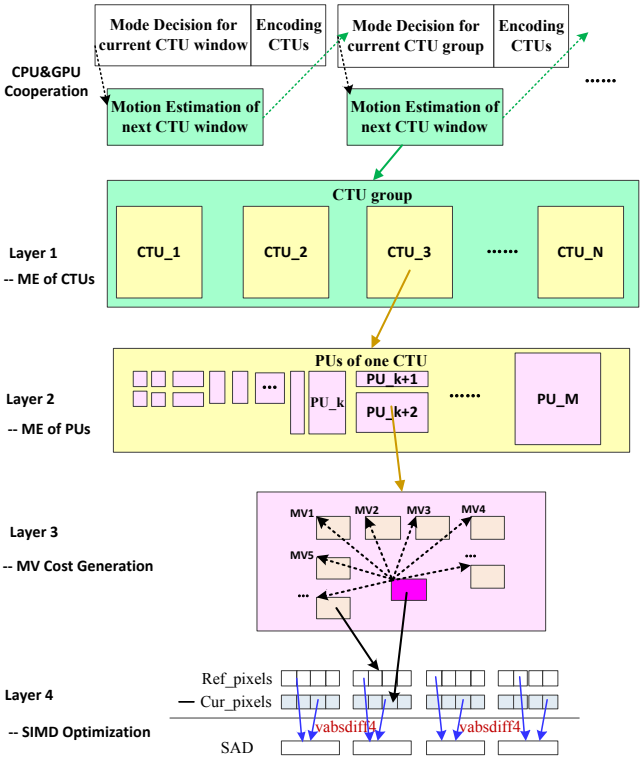

Fig. 1. Parallelization layers of the proposed implementation

In this paper, a CPU and GPU cooperative encoder framework is provided. In the encoder, ME is implemented on GPU and a novel way for fast PU cost generation instead of previous progressive cost merging for various PU sizes [8][9][10] is applied to parallelize the motion estimation of all PUs in one CTU. With this scheme, ME of variable prediction units is able to be conducted concurrently, thus a four-layer parallel motion estimation algorithm is realized, as shown in Fig. 1. Experimental results show that the proposed method can achieve over 90 times speedup with ignorable efficiency loss compared to the HM10.0 using fast ME.

The rest of this paper is organized as follows. Section II introduces the proposed four layer parallelization framework of the ME method. Section III talks about the implementation of ME on GPU, followed by experimental results in section IV. Finally section V summarizes the paper.

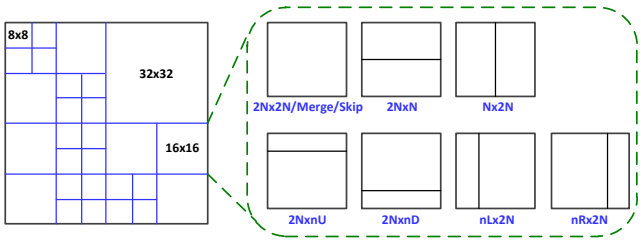

Fig. 2. An example of CTU partition and inter prediction modes

II. Encoder Framework

The proposed encoder framework with multiple layer parallel on GPU is presented in Fig. 1. GPU can process large amounts of data efficiently, but lacks the ability to run programs with complex logic like mode decision. So GPU just finish the motion estimation of all possible prediction units in each CTU, while CPU takes the work of mode decision, transformation, quantization, entropy coding and loop filtering.

In this framework, CTUs are divided into groups to reduce the synchronization points. The group size ranges from one to frame width in CTU depending on the encoder configuration. A larger group size means motion vector prediction (MVP) on the left side could not be used because the neighboring PUs are processed at the same time, while a smaller size leads to more synchronization overheads between CPU and GPU and slows down the coding process. After a lot of testings, we set the group size as a tradeoff between the parallelism and the overheads.

The parallelization of ME on GPU can be classified into four layers. The layer is the parallelization of CTU. By disabling motion vector prediction (MVP) from the left-CTU, motion estimation of CTUs in a row can be conducted concurrently. Another important layer is PUs' concurrency, which was achieved by setting all PUs' MVP as values in the top CTUs and adopting a novel cost merging scheme. The parallelization of searching multiple MVs is a common scene on GPU's ME implementation, and brought significant speedup. The bottom layer is SIMD video instruction optimization to further accelerates the ME process. For example, the instruction *vabsdiff4* accomplishes the SAD calculation of 4 pairs of bytes and brings in much time saving.

Similar to the ME process on CPU, the proposed scheme consists of integer-pixel motion estimation (IME) and fractional-pixel motion estimation (FME). Since each sub-pixel could be used for tens or hundreds of times during the full search, an early interpolation is conducted before the FME of all CTU groups.

III. Proposed ME Implementation on GPU

The recursive quad-tree structure and asymmetric mode partition (AMP) in HEVC is shown in Fig. 2. It brings in more partition modes in inter prediction, thus merging smaller CUs for larger ones in a hierarchical way as [8] turns programming unfriendly. Such method also increases the dependence between PUs. A novel way of addressing this issue and realizing parallelization in PU-layer is conducted by using a mode indexing table and a SAD look-up table as showed in

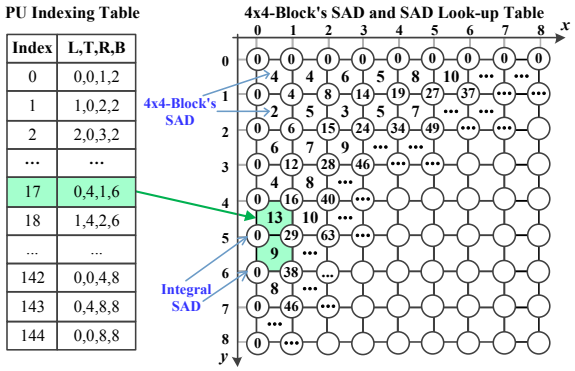

Fig. 3. Indexing of prediction unit and its SAD

Fig. 3, where each item in the mode table is constituted by a quaternion of *(L, T, R, B)*, standing for the position and size of one certain PU. With this scheme, an efficient multiple layer parallel motion estimation is achieved on GPU.

A. Parallelization in CTU-layer

In this parallel coding framework, CTUs are processed group by group successively. On the GPU side, a group of CTUs is being processed concurrently. On CPU side, mode decision and other processes of the encoder for a CTU group are conducted after the ME of all PUs is accomplishes.

The number of CTUs being processed each time affects the processing speed and estimation accuracy. On the one hand, usually more CTUs being processed concurrently leads to higher utilization of GPU and less average time delay. And the fastest circumstance is, only one synchronization point appears in the ME with one reference frame. On the other hand, processing fewer CTUs each time brings in higher coding efficiency since motion vector prediction can introduce significantly coding gain, but requires motion information results of former CTUs. When the parallelization in PU-layer is enabled, the neighboring PUs in the same CTU are also processed concurrently, meaning no MVP could be obtained from the current CTU. Thus in the proposed algorithm, only MVPs from neighboring CTUs could be used. As a tradeoff between speed and coding efficiency, the CTU group size is set as the CTU width of the frames in the proposed encoder implementation. In this way, only MVPs from the above CTUs were used and simplified process logic on GPU. During the following full motion estimation, the average of two above 16x8 PUs' best IMVs is adopted as the searching center for all PUs in a CTU. Thus coding efficiency loss of the implementation was below 3%.

B. Parallelization in PU-layer

The quad-tree coding unit structure of HEVC makes it difficult to merge costs from small PUs. However, a novel scheme was introduced in our implementation by using the mode indexing table, as shown in Fig. 3. When the size of PU ranges from 4x4 to 32x32, there are 10 different sizes and 169 different PUs in total (assuming that the asymmetric mode partition). By expanding this table, more possible PU modes in Fig. 2 could be searched concurrently.

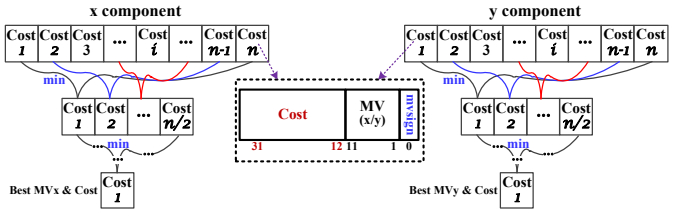

Fig. 4. MV selection without branching

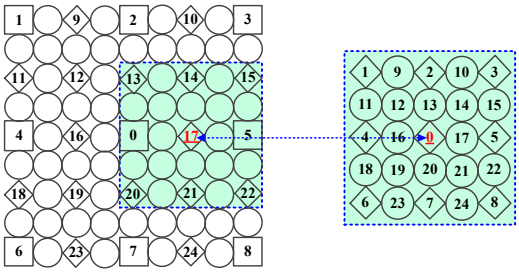

Fig. 5. Searching points in fractional-pixel motion estimation

As shown in Fig. 3, each item of the mode index table consists of 4 numbers. For instance, the index 17-mode is recorded as (0,4,1,6) where 1 unit stands for 4 pixels in width or height. So the index 17 means a PU from (0,16) to (4,24) in a CTU, whose size is 4x8 in pixel, as marked in the right side of Fig. 3. For decreasing the branches and balancing workload of threads in a warp, the arrangement of the indexing table obeys the following rules, (1) PU with smaller size is arranged in lower position, (2) Neighboring PUs are put together.

The PU-layer parallelization was realized by using a SAD look-up table. To get the distortion of each PU in parallel for any MV, firstly the SAD of every 4×4 block $D_{4\times4}$ is calculated with formula (1) and is stored in an 8×8 table i.e. the right side of Fig. 3. Then all SADs of the 4×4 blocks are summed up from the left-top point $(0, 0)$ to the current point (M, N) as (2), thus an SAD look-up table consisting of values in the circles is generated. Lastly, distortion of each PU D_{PU} in position *(L, T, R, B)* could be obtained in constant time with formula (3) with its position.

$$D_{4\times4}(i,j) = \sum_{k=4i}^{4i+3}\sum_{l=4j}^{4j+3} |ref_{k,l} - org_{k,l}| \quad (1)$$

$$S(M,N) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} D_{4\times4}(i,j) \quad (2)$$

$$D_{PU}(L,T,R,B) = S(R,B) + S(L,T) - S(R,T) - S(L,B) \quad (3)$$

C. Parallelization in MV-layer

During motion estimation, the distortion of different MVs can be calculated concurrently on GPU. But reducing the number of MVs is a bottleneck of GPU based encoder implementation since GPUs are not good at processing branching instructions which were definitely needed in traditional ME. In the proposed implementation, the MV reduction is realized by combining the MV component with its cost. As shown in Fig. 4, when the costs of different MVs for one PU were gained, we arrange the distortion in the higher position of one 32-bit word and the MV component in the lower. Then, through *min* operation of GPU, MVs with less cost would be reserved through a series of comparisons.

Depending on the characteristics of IME and FME, the MV-level parallel processing is done as follows:

1) Integer-pixel Motion Estimation (IME): The IME of each CTU is conducted in a dependent thread block of CUDA. Firstly at most 64 group of threads are allocated to compute the SADs simultaneously for 64 MVs concurrently. In each thread group, 8 threads are used to calculate the SAD of a specific 4×4 CU column so 512 threads are active. Later the SAD look-up table for fast cost generation will be achieved by summing up the values in each table. After that, costs, consist of the PU's SAD and 2 times of MV difference (MVD) as an estimation of rate-distortion cost, of different integer motion vectors (IMVs) are compared to select a best IMV for each mode.

2) Fractional-pixel Motion Estimation (FME): FME is conducted just after the IME of one CTU is completed. In this process, the FME of each PU is conducted independently because different PUs may have unique best IMVs. Similar to FME in HM, the searching process consists of half-pixel MV refinement and quarter-pixel MV refinement. There are 25 points to be searched in each step, among which 9 have already been checked in former steps. So only the left 16 points, labeled from 9 to 24 as illustrated in Fig. 5, need to be checked in each refinement step.

When FME of all PUs in one CTU was finished, the best MVs of all the CTUs are transferred back to system memory for mode decision on CPU.

TABLE I. PERFORMANCE SUMMARY WITH SR=[-16,16] COMPARED TO HM10.0

	WQVGA	WVGA	720p	1080p
Encoder Speedup	2.64	2.82	3.46	2.99
Motion Estimation Speedup	40.68	62.14	91.55	152.21
Average Loss (BD-rate)	1.99%	2.42%	1.21%	2.31%

IV. EXPERIMENTAL RESULTS

The proposed multi-layer parallel motion estimation was implemented on HM10.0, and no further optimization on CPU was conducted. The experiments were conducted on a desktop with Intel Core i5 760, 4G RAM and GTX 480. The GPU has a CUDA capacity version of 2.0, which was not powerful. The operating system is Windows 7 and the installed CUDA toolkit has a version of 5.0.

During the tests, the largest CTU size was set to 32x32 and AMP was turned off due to the limitation of our GPU. and 100 frames are tested for sequences from class B to E, using low delay P configuration. As provided by Table I, the implemented encoder can achieve about 3 times speedup and over 60% encoding time (ET) saving for all sequences. When the searching range (SR) is set to [-32,32], 32 times speedup with ignorable loss can be achieved for 720p sequences, while speedup can be over 170 times and BD-rate loss becomes 1.78% when the SR is set to [-8,8], as shown in Fig. IV. More details of our encoder's performance are attached in Table II.

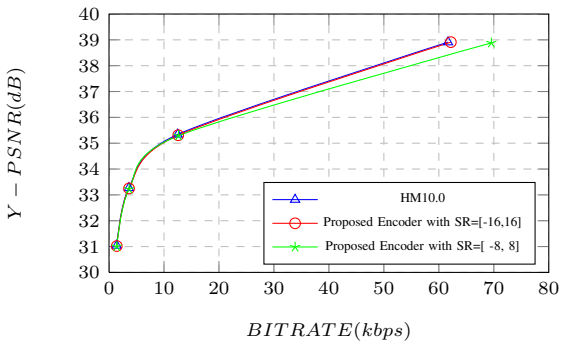

Fig. 6. Performance curve of encoding BQTerrace

TABLE II. CODING PERFORMANCE UNDER DIFFERENT SRS COMPARED TO HM10.0

SR	Sequnce	Y	U	V	ΔET
[−16, 16]	*Kimono*	1.59%	1.26%	1.98%	-69.84%
	ParkScene	1.78%	1.59%	1.64%	-64.43%
	Cactus	1.54%	0.84%	1.44%	-66.86%
	BasketballDrive	4.63%	5.44%	4.74%	-69.29%
	BQTerrace	2.00%	1.85%	1.98%	-62.42%
	FourPeople	1.03%	0.92%	0.54%	-71.38%
	Johnny	1.43%	0.90%	0.69%	-71.03%
	KristenAndSara	1.18%	0.82%	1.08%	-70.96%
	Class C	2.42%	2.01%	2.05%	-64.53%
	Class D	1.99%	1.59%	1.52%	-62.08%
[−32, 32]	*Kimono*	1.20%	1.13%	1.64%	-69.74%
	ParkScene	1.41%	1.12%	1.07%	-64.31%
	Cactus	1.34%	0.58%	0.97%	-66.78%
	BasketballDrive	3.22%	3.35%	2.81%	-69.31%
	BQTerrace	1.68%	1.55%	1.54%	-62.30%
	FourPeople	0.93%	0.83%	0.74%	-72.29%
	Johnny	1.25%	1.18%	1.36%	-71.14%
	KristenAndSara	1.29%	0.99%	1.31%	-71.10%
	Class C	1.80%	1.35%	1.47%	-64.45%
	Class D	1.79%	0.92%	1.63%	-61.77%
[−64, 64]	*Kimono*	1.03%	0.72%	1.23%	-69.80%
	ParkScene	1.40%	1.09%	0.93%	-64.23%
	Cactus	1.23%	0.22%	0.69%	-66.58%
	BasketballDrive	2.43%	2.48%	1.67%	-69.27%
	BQTerrace	1.70%	1.09%	1.17%	-62.16%
	FourPeople	0.94%	0.43%	0.54%	-70.95%
	Johnny	1.43%	1.78%	1.45%	-70.77%
	KristenAndSara	1.28%	0.91%	0.97%	-70.42%
	Class C	1.38%	0.88%	1.00%	-64.25%
	Class D	1.63%	0.76%	1.19%	-61.28%

The performance of the proposed techniques for corresponding layers was checked with 720p sequences and the results were given in Table III where ΔT(ME) is the time reduction of ME on GPU. Parallelization in PU layer was always enabled as the main innovation compared to other optimization methods. When parallelization in CTU-layer were adopted, about 10 times speedup can be achieved. More acceleration can be achieved through non-branching MV reduction and SIMD instruction optimization.

Compared to former implementations like [8] [9] and [10], where progressive cost merging methods were used, higher utilization of GPU and less dependency between PUs can be achieved by using an SAD lookup table. And the proposed fast cost generation method realized parallel processing for all possible PUs. Furthermore, this proposed scheme is programming friendly and can adapt to AMP well by just adding more items into the mode indexing table.

TABLE III. PERFORMANCE OF PROPOSED PARALLEL TECHNIQUES

GPU Optimization Technics	ΔT(ME)	ME Speedup
PU-layer parallelization	—	—
CTU-layer parallelization	-91.29%	11.48
Non-branching MV reduction	-5.43%	12.14

V. CONCLUSION

In this paper, a four-layer parallelized motion estimation on GPU platform is proposed. The PU-layer concurrency is more flexible and adaptive to various PU partitions in HEVC than previous parallel ME methods. With more exact executing paths on GPU, the proposed method makes more use of GPU and achieves higher acceleration. About 100 times speedup can be reached for high definition sequences. As a whole, the proposed scheme can save over 60% of the encoding time with acceptable loss.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation of China (61322106, 61421062), National Basic Research Program of China (973 Program, 2015CB351800), and in part by Supervisor Award Funding for Excellent Doctoral Dissertation of Beijing (20128000103). This work was also granted by Cooperative Medianet Innovation Center.

REFERENCES

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

[2] K. Chen, Y. Duan, L. Yan, J. Sun, and Z. Guo, "Efficient SIMD optimization of HEVC encoder over x86 processors," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference(APSIPA ASC)*, Hollywood, CA, December 2012, pp. 1–4.

[3] S. Ma, S. Wang, S. Wang, L. Zhao, Q. Yu, and W. Gao, "Low Complexity Rate Distortion Optimization for HEVC," in *Data Compression Conference*, vol. 73, no. 1. Snowbird, UT: IEEE, March 2013, pp. 73–82.

[4] NVIDIA Corporation, "NVIDIA CUDA C programming guide," 2014.

[5] W.-N. Chen and H.-M. Hang, "H.264/AVC motion estimation implementation on compute unified device architecture (CUDA)," in *IEEE International Conference on Multimedia and Expo*. Hannover, Germany: IEEE, June 2008, pp. 697–700.

[6] R. Rodriguez, J. Martínez, G. Fernández-Escribano, J. Claver, and J. Sánchez, "Accelerating H.264 inter prediction in a GPU by using CUDA," in *Digest of Technical Papers International Conference on Consumer Electronics (ICCE)*. Putrajaya, Malaysia: IEEE, 2010, pp. 463–464.

[7] D.-K. Lee and S.-J. Oh, "Variable block size motion estimation implementation on Compute Unified Device Architecture (CUDA)," in *IEEE International Conference on Consumer Electronics (ICCE)*. Shenzhen, China: IEEE, April 2013, pp. 633–634.

[8] X. Wang, L. Song, M. Chen, and J. Yang, "Paralleling variable block size motion estimation of HEVC on CPU plus GPU platform," in *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*. San Jose, CA: IEEE, July 2013, pp. 1–5.

[9] F. Wang, D. Zhou, and S. Goto, "OpenCL based high-quality HEVC motion estimation on GPU," in *IEEE International Conference on Image Processing (ICIP)*, Paris, France, October 2014, pp. 2–6.

[10] S. Radicke, J. Hahn, C. Grecos, and Q. Wang, "A highly-parallel approach on motion estimation for high efficiency video coding (HEVC)," in *IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, January 2014, pp. 187–188.