OPTIMIZATION OF MOTION COMPENSATION BASED ON GPU AND CPU FOR VVC DECODING

Xu Han[1], *Shanshe Wang*[2], *Siwei Ma*[2], *Wen Gao*[2]

[1]School of Electronic Information and Electrical Engineering, Shanghai Jiaotong University
[2]Institute of Digital Media, Peking University

ABSTRACT

To achieve higher compression efficiency, the new developing video coding standard Versatile Video Coding(VVC) introduced a large amount of new coding technologies, which increases the computational complexity of the decoder significantly. Among these technologies, the inter prediction methods, including affine motion compensation and decoder side motion vector refinement(DMVR), make inter prediction become the most time consuming module and bring new challenges for real-time decoding. In this paper, we proposed an efficient GPU-based motion compensation scheme to speedup the decoding. Through re-partition of coding unit(CU) according to the data dependency and different thread organization methods for different situation, the computational resources of GPU are utilized efficiently. Experiments on NVIDIA GeForce RTX 2080Ti GPU showed the motion compensation can be done in 5ms for Ultra HD 4K, which means the decoding speed is accelerated by 16 times compared to the VVC reference software on CPU.

Index Terms— VVC, GPU, video decoding, motion compensation

1. INTRODUCTION

Versatile Video Coding standard is the new generation video coding standard which is being developed by JVET. Its aim is to further improve coding efficiency for UHD. Many new coding tools are carefully designed to achieve coding performance improvement, e.g. new coding tree unit structure, AMVR, HMVP and so on. However, they also bring in more decoding complexity. It is stated in [1] that the decoding time of VVC decoder is over 1.6 times than that of HEVC.

To achieve real-time decoding, many computational platforms have been attempted, including CPU, GPU, FPGA[2] and DSP[3]. Among these platforms, CPU and GPU are the most common platforms.

For CPU-based decoder, besides basic programming optimization, an important strategy is how to achieve as much parallelism as possible. Single instruction, Multiple Data(SIMD) instructions are proposed to accelerate the decoding process in data-level parallelism, which reduced the instruction numbers and improved the execution efficiency. However, due the pixel dependency and block size, SIMD instructions sometimes do not show satisfying speed. Chi et al.[4] shows speedup status of every modules in HEVC with different SIMD instruction set. VTM, the reference software of VVC, also adopted SIMD instructions to accelerate interpolation and other process. Multiple threading technology is another indispensable technology to optimization decoder, which speeds up the decoder by task-level parallelism. The main drawback of multiple threading is its performance is limited by the reference relationship among frames[5]. OpenHEVC is a representative open-source HEVC decoder using both methods, which is used in many works as benchmark, such as [4, 6, 7].

Compared to CPU, GPU usually has lower clock frequency but more computing units which can performing parallel computations on a very large scale. There are two key points in GPU based decoding. First is how to schedule a large number of threads to make as many threads as possible executing simultaneously so that the computing resources of GPU can be fully utilized. Another one is how to utilize the characteristic of the memory hierarchy to minimize the memory access time. De et al.[7] integrates the whole decoding pipeline except entropy decoding into a GPU decoder based on OpenHEVC, which achieve 145fps for 4K videos with the NVIDIA GeForce GTX TITAN X GPU. To strike a balance between several constraints, such as parallelism degree, amount of shared memory and global memory bandwidth, they split PU into 8x8 pixel block to perform motion compensation. Jiang et al.[8] also proposed to split PU into several blocks smaller than 32x32 to save the shared memory. Compared with HEVC, the data dependency between pixels in VVC are more complex. The old resource allocation schemes and the organization of threads are not suitable for the new motion compensation process. In this paper, we proposed an efficient GPU-based motion compensation scheme with the re-partition for different coding unit and suitable thread organization. Compared with the performance on CPU, the decoding speed is accelerated by 16 times.

The rest of paper is organized as follows: Section 2 provides the decoding complexity analysis in VVC. The pro-

posed GPU-based motion compensation scheme is presented in Section 3. The experimental results are shown in Section 4 and finally Section 5 concludes the paper.

2. MOTION COMPENSATION IN VVC

Motion compensation is the most time consuming module in VVC. Fig.1 shows the decoding time distribution of main modules and the detailed distribution of inter prediction in VTM6.1. MC in the figure is the traditional motion compensation process. Affine motion compensation(Affine) and decoder-side motion vector refinement(DMVR) are the two new technologies in inter prediction module improved the coding efficiency significantly. MV is the derivation process of MV. In order to illustrate the complexity of each module, we disabled the SIMD optimization in decoder, which changes the decoding time of inter prediction and ALF greatly. As we can see, DMVR and Affine account for a large part of the complexity in inter prediction. Besides, Fig.2 has compared the difference speedup ratio of applying AVX2 instructions to HEVC and VVC decoder, using data from [4] and [9]. As illustrated, applying SIMD optimization to new technologies in VVC cannot achieves as high speedup ratio as in HEVC.

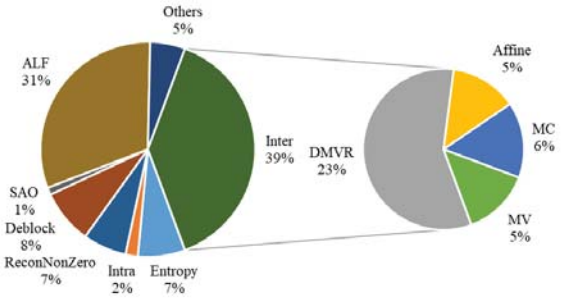

Fig. 1. Decoding time of main modules in VVC decoder

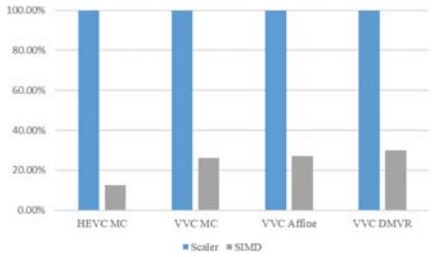

Fig. 2. SIMD optimizations impact of motion compensation in HEVC and VVC

3. GPU-BASED MOTION COMPENSATION

3.1. Overall Architecture

Because of the special architecture, GPU cannot handle a process efficiently if too many prediction methods integrated. First, complicated processes cause the compiler to assign more registers to a thread block, which decreases the number of thread blocks that can executing simultaneously. Secondly, a general process needs a general data structure that contains all of the prediction information, which will increases the communication cost between CPU and GPU. Finally, for motion compensation, different prediction technologies require different computing resource. A integrated process creates great challenges for resource allocation. Therefore, in the proposed GPU implementation, all CUs are classed into 4 categories according to the prediction technologies they used, which are DMVR CU, affine CU, triangle CU and common CU.

Fig.3 shows the 4 step of the proposed GPU-based motion compensation. Firstly, the entropy decoding is performed and the prediction info of 4 types of CU will be initialized, including the motion info, the block partition info and the tools on-off flags. Secondly, the prediction info will be transferred to GPU successively. To overlap the transfer with computation, all memory copy instructions are asynchronous and performed in an individual stream named copy stream. The motion compensation of a type of CU begins as soon as the transfer of the corresponding info finished, which reduces the waiting time for prediction info. Thirdly, after the motion compensation, the results will be copied back to CPU for reconstruction. Fourthly, the filtering result will be transferred to GPU during the entropy decoding of the next frame. To avoid unnecessary communication, the reference pixels transmission starts after the slice header parsed, at which the reference relationship is determined. Because the rest of the entropy decoding process takes more time than transferring a single frame, the transfer time is hided completely.

The motion compensation process of luma component and chroma component are separated into two kernels, which is mainly because of their different block size. After the separating, we can allocate different computing resources to different components. Moreover, some technologies only applied on luma component can be removed from chroma kernels to further simplify the chroma kernels. As the chroma kernels require less resources and not depend on luma kernel result (except DMVR CU), the two kernels can be executed in two streams concurrently, which increases the efficiency of GPU.

3.2. Motion Compensation of different types of CU

Memory access latency is one of the main factor that affects the efficiency of kernels. An uncached global memory access may cause 400 to 600 clock cycles of latency. There are two methods to reduce the memory latency. One is cre-

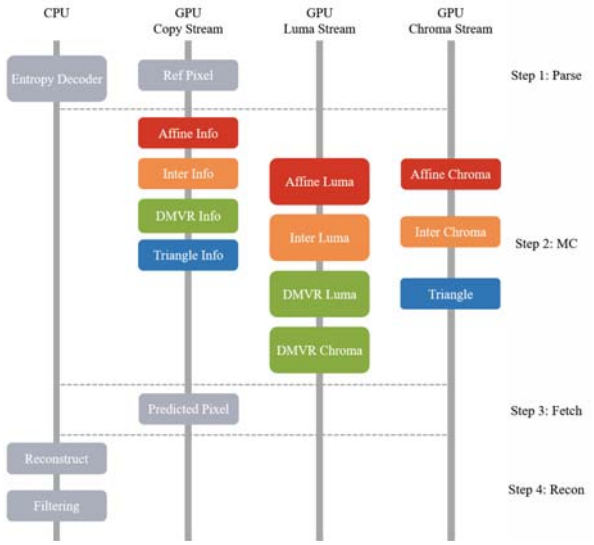

Fig. 3. Overview of the proposed GPU-based motion compensation

ating enough threads so that when some threads are waiting for memory, the other threads can execute normally. In the proposed GPU-based motion compensation module, a thread block is responsible for the motion compensation of one luma block or two chroma block. The number of thread block is same as the number of CTU. The size of thread block is 128 for luma blocks and 64 for chroma blocks. Another is to take the advantage of the multi-level memory architecture of GPU. Shared memory can be accessed in only dozens of cycles. We use shared memory to cache the reference pixels which are accessed frequently and store the temporary results. The shared memory is very limited and each thread block must be allocated the same size of shared memory. To reduce the shared memory used by every thread block, we split large CU into several sub-blocks, which increases the number of active thread blocks that can execute simultaneously.

Considering the characteristics of DMVR, we split DMVR CU into several blocks not larger than 16x16. For the motion compensation of a DMVR luma block, the first step is to search a refined MV which makes the SAD between the bi-direction prediction signals lowest. When calculating the SAD of the two blocks obtained from the initial MV, we need to get the SAD as soon as possible. Therefore, each warp calculates a quarter of current block and the four SAD from different warps are summed up by atomic instruction in shared memory. If the SAD is larger than a threshold, the another 24 SAD will be calculated. In this situation, it is important to reducing the block-level synchronization and the memory operation conflict caused by atomic instruction. So, each warp is responsible for calculating the SAD of two entire blocks at a time. The 24 SAD are all stored in shared memory. In the process of calculating, the lowest SAD of each warp is also stored in shared memory. After getting all of the 24 SAD, only 3 comparisons are required to get the final refined MV. The refined MV will be stored in global memory for later computation. Since the threads in different kernels can only communicate through global memory, the DMVR chroma kernel is performed in the same stream as the luma kernel, which ensures the motion compensation of chroma block starts after all refined MV are determined. Besides, the refined MV buffer will be transferred to CPU asynchronously after the DMVR luma kernel for the MV derivation of other frames. After the MV refinement, the interpolation and the weighted average are performed. The thread organization is similar to the calculation of the first SAD, each wrap operates a quarter of the block. Especially, for the blocks using bi-directional optical flow, the prediction pixels need to be further refined based on the gradients. The gradients calculation is similar as SAD, but when deriving the offset of prediction, each warp is responsible for a 6x6 subblock. When calculating the auto-correlation and cross-correlation of the gradients, with the help of warp shuffle instructions, different threads in a warp can exchange data through registers, which has lower latency than any other memory. It is worth noting that the gradients are also stored in the reference pixels and temp pixels buffer because the data in theses buffer will not be used anymore.

For CU using affine motion compensation, each 4x4 subblock has its own MV. As the minimum memory transaction is 32 bytes, it is inefficient to use 4x4 block as the basic unit of motion compensation. So we split Affine CU into several blocks not greater than 16x8 and each warp performs the motion compensation of one 4x4 subblocks at a time. The prediction pixels of 4x4 subblocks are stored in shared memory and written to global memory after all subblocks are finished. For the blocks using prediction refinement with optical flow, the gradients are unnecessary to be stored. Instead, we can calculate the prediction offset and add it on prediction results directly, which reduces the memory access times. In addition, when deriving the MV of the a chroma subblock, the MVs of the two corresponding luma subblcoks are recalculated, which introduces redundant calculation but removes the dependence between the luma and chroma components.

We split common CU into several blocks not larger than 16x16 because of bi-directional optical flow may be applied. The motion compensation of common CU is identical to the process of DMVR CU after the MV refinement. As for triangle CU, it is difficult to determine the shape of original triangle blocks after partition. The original size are unchanged and shared memory are not used in the triangle kernel. Considering the number of triangle blocks is very few, we perform the motion compensation of the triangle blocks in the chroma stream of GPU. The kernel requires only a little resource so that it can be performed during the motion compensation of DMVR blocks.

Table 1. Decoding speed(FPS) of proposed method

Seq	QP	CPU	GPU	Speed Up
Campfire	22	1.90	2.11	10.94%
	27	3.07	3.52	14.92%
	32	3.81	4.54	19.20%
	37	4.42	5.30	19.85%
CatRobot1	22	2.71	3.65	34.76%
	27	3.73	5.58	49.65%
	32	4.40	7.04	59.93%
	37	5.03	8.36	66.28%
DaylightRoad2	22	2.38	2.99	25.49%
	27	3.49	5.25	50.71%
	32	4.12	6.60	60.19%
	37	4.82	8.44	75.18%
FoodMarket4	22	3.12	4.39	40.67%
	27	3.59	5.40	50.55%
	32	4.09	6.50	59.02%
	37	4.72	7.89	67.35%
ParkRunning3	22	1.69	2.20	29.91%
	27	2.21	3.11	40.44%
	32	2.78	4.12	48.59%
	37	3.41	5.27	54.40%
Tango2	22	2.94	3.74	27.24%
	27	3.73	5.31	42.41%
	32	4.14	6.09	46.95%
	37	4.70	7.08	50.49%
Average		3.54	5.19	46.47%

Table 2. Time percentage of inter prediction

	22	27	32	37
Campfire	15.9%	18.7%	22.7%	24.6%
CatRobot1	33.4%	44.6%	53.4%	61.2%
DaylightRoad2	30.9%	43.2%	48.7%	58.3%
FoodMarket4	30.9%	43.2%	48.7%	58.3%
ParkRunning3	36.8%	43.3%	49.2%	57.0%
Tango2	33.3%	39.5%	42.5%	44.4%
Average	30.4%	38.8%	40.9%	45.8%

Table 3. Time consuming(ms) of every type block

	DMVR	Affine	Inter	All
CPU	45.08	19.36	14.77	79.20
GPU	1.60	0.29	0.59	4.87
speed up times	28.22	65.78	25.10	16.26

4. EXPERIMENT RESULT

To evaluate the efficiency of the proposed GPU motion compensation approach, the JVET common test condition(CTC) is adopted with random-access configuration[10]. Because ultra high definition video is the key application area for the use of VVC, only the 4K sequences in Class A are considered in our experiments. The proposed algorithms are implemented with CUDA 10.2 and integrated into VTM 6.1, which is the reference software of VVC. The motion compensation are performed in GPU and the other modules, including entropy decoding, intra prediction, inverse transform, dequantization and in-loop filter are executed in CPU.

Table 1 shows the decoding speed of different decoders when decoding different bitstreams. We evaluated our algorithms on NVIDIA GeForce RTX 2080Ti, which has 4352 CUDA cores and 1545 MHz boost clock. The baseline is the decoding speed of VTM 6.1 on Intel Core I7-8700K CPU, whose max turbo frequency is 4.70 GHz. As can be observed in Table 1, the proposed method achieves significant lower execution time. Compared to VTM6.1, the decoding speed of our decoder has increased by 46% on average. It should be noted that the acceleration is greatly affected by the sequence content and the bitrate. Table 2 shows the percentage of processing time of motion compensation in the whole decoding process. A large part of CU in sequence Campfire did not use inter prediction, which makes the acceleration ratio significantly lower than the other sequences. Besides, as the QP increasing, the bitrate of bitstreams decrease, which makes the encoder prefers to decrease the usage of intra prediction and make prediction in skip or merge mode. So the acceleration ratio increases as the bitrate decreases.

Table 3 shows the detailed data about the execution time of different modules. The time of triangle CUs are hided completely, so there is only decoding time of 3 types of CU and the total time of motion compensation including the memory transfer time. We can see the computing process are greatly accelerated. Affected by the transfer time of prediction pixels, finally, the the motion compensation can be finished in about 5ms which accelerated about 16 times.

5. CONCLUSION

This paper proposed a GPU-based motion compensation scheme for video decoder compatible with the VVC standard. We exploited how to schedule threads and allocate shared memory in a complex inter prediction process. Compared to the reference software VTM 6.1, the proposed scheme achieve 16x speedup for motion compensation process and about 46% acceleration for the entire decoding process on NVIDIA RTX 2080Ti. After our optimization, the in-loop filter become the new bottleneck of decoding speed. Next step we will explore the GPU-based optimization methods of other modules and the asynchronous cooperation between CPU and GPU, so as to finally achieve real-time decoding of 4K or even 8K video.

6. ACKNOWLEDGEMENT

This work was supported by Key-Area Research and Development Program of Guangdong Province (2019B010133001), National Key Research and Development Project (2019YFF0302703) and High-performance Computing Platform of Peking University, which are gratefully acknowledged.

7. REFERENCES

[1] Frank Bossen, Xiang Li, and Karsten Suehring, "Jvet ahg report: Test model software development (ahg3)," *JVET Document, P0003*, 2019.10.

[2] Maleen Abeydeera, Manupa Karunaratne, Geethan Karunaratne, Kalana De Silva, and Ajith Pasqual, "4k real-time hevc decoder on an fpga," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 236–249, 2015.

[3] M Chavarrias, Fernando Pescador, Matías J Garrido, Eduardo Juarez, and Mickaël Raulet, "A dsp-based hevc decoder implementation using an actor language dataflow model," *IEEE Transactions on Consumer Electronics*, vol. 59, no. 4, pp. 839–847, 2013.

[4] Chi Ching Chi, Mauricio Alvarez-Mesa, Benjamin Bross, Ben Juurlink, and Thomas Schierl, "Simd acceleration for hevc decoding," *IEEE Transactions on circuits and systems for video technology*, vol. 25, no. 5, pp. 841–855, 2014.

[5] Chi Ching Chi, Mauricio Alvarez-Mesa, Ben Juurlink, Gordon Clare, Félix Henry, Stéphane Pateux, and Thomas Schierl, "Parallel scalability and efficiency of hevc parallelization approaches," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1827–1838, 2012.

[6] Yizhou Duan, Jun Sun, Leju Yan, Keji Chen, and Zongming Guo, "Novel efficient hevc decoding solution on general-purpose processors," *IEEE Transactions on Multimedia*, vol. 16, no. 7, pp. 1915–1928, 2014.

[7] Diego F de Souza, Aleksandar Ilic, Nuno Roma, and Leonel Sousa, "Ghevc: An efficient hevc decoder for graphics processing units," *IEEE Transactions on Multimedia*, vol. 19, no. 3, pp. 459–474, 2016.

[8] Bo Jiang, Falei Luo, Shanshe Wang, Xiaoqiang Guo, and Siwei Ma, "Efficient gpu-based inter prediction for video decoder," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 1109–1113.

[9] Adam Wieckowski, Gabriel Hege, Christian Bartnik, Christian Lehmann, Christian Stoffers, Jens Brandenburg, Tobias Hinz, Benjamin Bross, H. Schwarz Detlev Marpe, and Thomas Schierl Thomas Wiegand, "Development of a vvc live software decoder," *JVET Document, P0973*, 2019.10.

[10] Frank Bossen, Jill Boyce, Karsten Suehring, Xiang Li, and Vadim Seregin, "Jvet common test conditions and software reference configurations for sdr video," *JVET Document, N1010*, 2019.3.