EFFICIENT GPU-BASED INTER PREDICTION FOR VIDEO DECODER

Bo Jiang[1], *Falei Luo*[2], *Shanshe Wang*[2], *Xiaoqiang Guo*[3], *Siwei Ma*[2]

[1]School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School
[2]National Engineering Laboratory for Video Technology, Peking University, Beijing, China
[3]Academy of Broadcasting Science, SAPPRFT, Beijing, China.

ABSTRACT

Interpolation is a very important module in inter prediction for any video decoder, e.g. AVS2 [1] and HEVC [2], which occupies most of the time in the whole decoding process . Thus, the real-time decoder is largely limited by the speed of inter prediction. To solve this problem, we propose an efficient GPU-based interpolation framework for inter prediction. Through optimizing shared memory allocation and thread scheduling on the GPU side, GPU are utilized efficiently and inter prediction is accelerated effectively. The experimental results on AVS2 show that for all Ultra HD 4K, WQXGA and full HD video sequences tested, the inter prediction acceleration ratio is over 6 times, and the average processing time is up to 1.25ms, 0.75ms and 0.45ms, respectively, with the NVIDIA GeForce GTX 1080TI GPU.

Index Terms— AVS2, HEVC, GPU, interpolation, video decoder

1. INTRODUCTION

Inter prediction is an indispensable part of video decoder, which takes up a large part of decoding time, especially on the platform of CPU decoding. Taking the open source decoder davs2 [3] as an example, when the quantization parameter(QP) is 27, the time of inter prediction decoding accounts for 35.77% of the total resolution time. Furthermore when the QP is 45, the inter-prediction time accounts for 56.32%, as shown in Fig.1. This is mainly because there are a large number of blocks with different sizes in inter prediction. During decoding, 1/4 pixel interpolation of luma prediction blocks and 1/8 pixel interpolation of chroma prediction blocks are required, which requires high memory bandwidth and a large number of arithmetic operations.

In previous researches, people tend to accelerate a module with multi-core CPU [4] and SIMD instructions [5], but the acceleration effect is often limited. Diego F. de Souza et al. [6] propose a new GPU parallel acceleration method for inter prediction module. However, it takes an average of 20.4ms to decode a frame for UHD 4K video image, which can not meet our need for 4K 50fps. Biao Wang et al. [7] propose a complete HEVC decoding solution for heterogeneous CPU+GPU systems, and the frame rate of 4K UHD video decoding reaches 150 fps, but it has higher requirements for CPU and GPU.

To realize inter prediction on GPU, we propose a new GPU-based inter prediction acceleration method and implement it with Compute Unified Device Architecture(CUDA). We split the luma prediction block whose width or height is more than 32 before interpolation proceeding, so that its maximum height or width is not beyond 32. Similarly, we split the chroma block so that its height or width does not exceed 16. Through optimizing shared memory allocation and thread scheduling on the GPU side properly, the average decoding time for a 4K video frame is 1.25 ms.

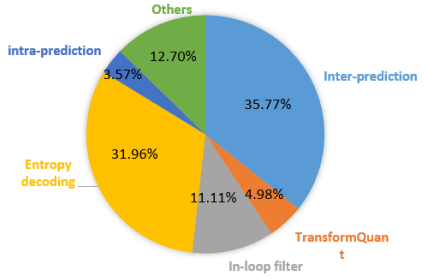

fig.1(a) Sequence:pku_girls QP 27.

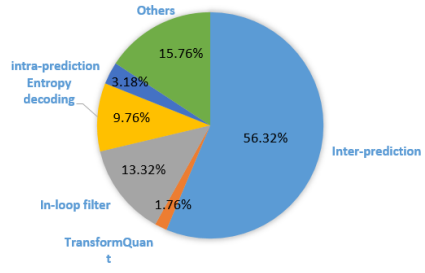

fig.1(b) Sequence:pku_girls QP 45.

Fig. 1. Decoding complexity distribution of davs2.

This work was supported in part by the National Basic Research Program of China (973 Program, 2015CB351800), National Natural Science Foundation of China (61632001), and High-performance Computing Platform of Peking University, which are gratefully acknowledged.

The rest of the paper is organized as follows. Section 2 gives the background information of fractional pixel interpolation and Section 3 shows the GPU-based algorithm, while the experimental results and conclusions are addressed in Sections 4 and 5, respectively.

2. INTERPOLATION IN INTER PREDICTION

The fractional pixel position in inter prediction is shown in Fig.2. The positions with green background color, such as $A_{-1,-1}$, $A_{0,-1}$, $A_{-1,0}$, etc, are integer positions, and the others are fractional positions. Motion vectors with quarter-pixel precision are used in both HEVC and AVS2 video coding standards. When the position pointed by the motion vector is a fractional pixel position, interpolation is required at the pixel position.

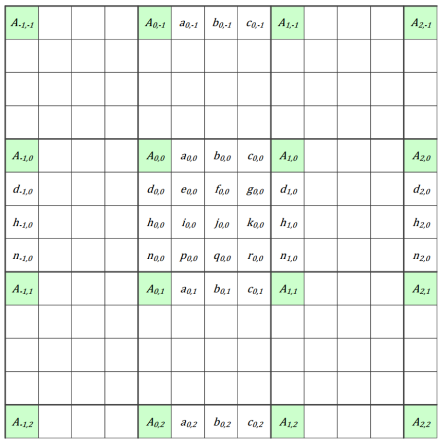

Fig. 2. Example of placing a figure with experimental results.

In the new generation of video coding standard, the pixel values of $a_{0,0}$, $b_{0,0}$, $c_{0,0}$ are the weighted average of $A_{-3,0}$, $A_{-2,0}$, $A_{-1,0}$, $A_{0,0}$, $A_{1,0}$, $A_{2,0}$, $A_{3,0}$, $A_{4,0}$, and different positions have different weighting coefficient matrices. Similarly, the pixel values of $d_{0,0}$, $h_{0,0}$, $n_{0,0}$ are the weighted average of $A_{0,-3}$, $A_{0,-2}$, $A_{0,-1}$, $A_{0,0}$, $A_{0,1}$, $A_{0,2}$, $A_{0,3}$, $A_{0,4}$. However, when calculating the value of $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $i_{0,0}$, $j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$, $r_{0,0}$, the predictions need to be accomplished accordingly in two steps. The difference between HEVC and AVS2 in fractional pixel prediction is that the weighted average factors used are different.

Taking the calculation of $e_{0,0}$ as an example, The calculation formula is:

$$e_{0,0} = \sum_{i=-3}^{4} (w_{0,i} \times a'_{0,i}) \tag{1}$$

Where $w_{0,i}$ is the weighting factor when interpolating, and there are special provisions in different coding standards. In addition, the factor is converted to an integer operation by expanding the coefficient by the power of two in the coding standard. For example, in the AVS2 video coding standard, the calculation formula for this position is:

$$e_{0,0} = Clip1 \left(\begin{array}{l} -a'_{0,-3} + 4 \times a'_{0,-2} - \\ 10 \times a'_{0,-1} + 57 \times a'_{0,0} + \\ 19 \times a'_{0,1} - 7 \times a'_{0,2} + \\ 3 \times a'_{0,3} - a'_{0,4} + \\ (1 << (19 - \text{BitDepth})) \end{array} \right) >> (20 - \text{BitDepth}) \tag{2}$$

Where $Clip1(x)$ means that the value of x is limited to the range of $[-32768, 32767]$, when x is less than -32768, the result is equal to -32768, and when x is greater than 32767, the result is equal to 32767, otherwise, the result is equal to x, and $(19 - BitDepth)$ is used for rounding.

$a'_{0,i}$ is calculated as:

$$a'_{0,i} = \sum_{j=-3}^{4} w_{0,j} \times A_{j,i} \tag{3}$$

According to the above calculation process, when interpolating the pixels at the position of $e_{0,0}$, a two-step interpolation operation is required. To elaborate, there is a need to interpolate to produce intermediate results $\{a'_{0,i} | \mathrm{i} \in [-3, 4]\}$ at first, but it is not equal to $a_{0,i}$ in Fig.2, because it doesn't clip to limit the value between 0 and the maximum pixel value, and then perform a second interpolation based on the intermediate result. According to this operation flow, temporary storage space needs to be allocated to save the intermediate interpolation result.

As for a prediction block with the respective width and height of W_{PU} and H_{PU}, the matrix width of the intermediate interpolation result is W_{PU}, but the matrix height is $(H_{PU} + 7)$, The fractional pixel interpolation at the upper boundary requires additional dependence on the 3-line pixel value on the top of the block, and the fractional pixel interpolation at the lower boundary requires additional 4 rows of pixels on the lower boundary block. Thus, the number of intermediate interpolation result pixels required to complete the sub-pixel interpolation is

$$NTS_{PU} = W_{PU} \times (H_{PU} + 7) \tag{4}$$

As a result, the number of pixels required for a 64x64 prediction block interpolation is 4544, and each pixel needs to be represented by 16 bits. In this case, it takes up about 8.88 KB of storage space. The storage cost corresponding to more different block sizes of prediction is shown in Table 1

3. GPU-BASED INTERPLOTION

In terms of GPU, allocating global memory will lead to increasing memory usage and inefficient memory access , but the use of high-speed shared memory can make it possible to

Table 1 Size of shared memory in inter prediction with different sizes.

shared memory size \ block size	64x64	32x32	16x16
intermediate result(KB)	8.88	2.44	0.72
final result(KB)	8.00	2.00	0.50
total(KB)	16.88	4.44	1.22

fully utilize GPU's computing resources. However, the shared memory size that can be used on each stream processor of GPU is limited. Using too much shared memory will result in the inability to perform more computational tasks on a stream processor, and will affect its computational efficiency. Thus, it is necessary to design an appropriate interpolation process to make full use of the computing resources of GPU.

In a device with different CUDA computing capabilities, the shared memory resource size of each stream multiprocessor is limited, as shown in Table 2. A typical configuration is 48 KB. The application needs to properly arrange the storage usage according to the size of the shared memory to achieve the optimal efficiency.

In the new generation of video coding standards, the maximum inter prediction block size is 64x64, which makes the size of a shared buffer up to 16.88 KB per inter block, as shown in Table 1. The excessive use of shared memory will restrict the number of blocks that GPU's stream multiprocessor can predict at the same time. The relationship among the number of predicted blocks that can be executed simultaneously on one stream multiprocessor(NPU_{SM}) , the shared memory used by each thread block($S_{PU,shared}$) and the shared memory size per stream multiprocessor($S_{SM,shared}$) is as follows:

$$NPU_{SM} = \left\lfloor \frac{S_{SM,shared}}{S_{PU,shared}} \right\rfloor \tag{5}$$

The limit number of blocks supported by a multiprocessor from the perspective of shared memory is calculated by Formula (5) for devices with different CUDA computing capabilities, the results are shown in Table 3.

Due to the complexity of video content, a large number of blocks in the video stream are not the largest inter prediction block size, which makes a great waste in many cases while allocating shared memory according to the maximum block size. Thus allocating memory according to the largest motion prediction block will cause inefficient GPU computation.

At the same time, the smaller the size of the prediction block split will leads to lower efficiency of GPU's acceleration. According to the interpolation processing flow of inter prediction, when the prediction block is split into multiple pieces, the intermediate result from the operation of the partial points will be repeated among various prediction blocks, which will add more redundant calculations. What's more, if

Table 2 Specification of devices with different CUDA capacity.

Source Limits per Multiprocessor											
CUDA Capacity	2.x	3	3.2	3.5	3.7	5.0	5.2	5.3	6.0	6.1	6.2
Max Thread Blocks	8	16				32					
Max Thread Warps	48	64									
Max Threads	1536	2048									
Shared Memory Size (KB)	48				112	64	96	64		96	64
Register File Size(KB)	32	64			128	64					
Source Limits per Thread Block											
CUDA Capacity	2.x	3	3.2	3.5	3.7	5.0	5.2	5.3	6.0	6.1	6.2
Shared Memory Size (KB)	48										
Source Limits per Thread											
CUDA Capacity	2.x	3	3.2	3.5	3.7	5.0	5.2	5.3	6.0	6.1	6.2
Registers (B)	63	255									

Table 3 Number of blocks under different restriction of shared memory.

shared memory size \ block size	64x64	32x32	16x16
48KB	2	10	39
64KB	3	14	52
96KB	5	21	78
112KB	6	25	91

the size of a prediction block is to small, it is not conducive to make full use of the GPU's threads to access memory. For example, a thread warp of CUDA is usually 32 in size, and if 32 threads access 32 32-bit consecutive data, the cost is equivalent to make one memory access. However, when a prediction block is smaller than 32x32, such as a 16x16 size prediction block, 32 threads access memory once will introduce a bank conflict and bring 2 times memory access time, which makes the memory access efficiency drop significantly. In addition, we need to take out the position, size, prediction mode and motion vector of the prediction block from the global memory before operating inter prediction. Smaller prediction blocks will bring more inter information when decoding a same frame, which will leads to an increase in the number of memory access and increase the data transfer cost from CPU to GPU, so it is not conducive to highly efficient decoding.

Through the above analysis, we propose to split the prediction block with width or height greater than 32 into multiple prediction blocks with width and height less than or equal to 32. As a result, the maximum unit of inter prediction

is 32x32, which reduces the need for shared memory per thread block. Consequently, more prediction blocks can be processed simultaneously on a multiprocessor, avoiding the extra cost of excessive small prediction block segmentation at the same time, such as more motion vector information copy, repeated motion vector information reading, repeated intermediate pixel interpolation, thus, achieving more acceleration.

After splitting the inter prediction block into the maximum size of 32x32, each block is allocated to a CUDA block, and each CUDA block is allocated 128 threads. Since each multiprocessor of GPU can run at most 2048 threads at the same time, 16 CUDA blocks can be executed at the same time. The computing capability of GTX1080TI is 6.1, and each multiprocessor has 96KB shared memory. In order to make the most efficient use of GPU, each CUDA block can use up to 6KB, which is greater than 4.44kb and meet the requirement of interpolation. As for the chromaticity component, the maximum PU is 16x16 after splitting, which can be similar to the luminance block, and each PU block is allocated to a CUDA block. Because the width and height of the chromaticity block are half of the luminance block, each block can be allocated 32 threads, and each multiprocessor will have 64 CUDA blocks, which is contrary to the maximum of 32 CUDA blocks per multiprocessor, as a result, this way cannot make full use of GPU. Therefore, we allocate 64 threads for each CUDA block. During inter interpolation proceeding, these threads will be divided into two groups, and each group will interpolate a inter block.

4. EXPERIMENT RESULTS

In order to verify the effectiveness of the proposed method, the proposed optimized decoder was tested on a hardware platform with i7 8700K CPU and NVIDIA GTX 1080 Ti GPU. In the test, the open source encoder xavs2 [8] is used to encode the sequences and the testing configuration is random access, meanwhile the encoding level is 5 and the encoding tools such as NSQT, AMP, SAO and ALF are turned off. The encoder encodes each sequence using a fixed QP. The QP used is the 4 QPs specified by the AVS2 common testing condition, which are 27, 32, 38, and 45. The inter prediction time of a sequence is the average inter prediction time of decoding the coded streams configured by 4 QPs.

As shown in Table 4, when all the prediction blocks are split into 4x8/8x4, the speed is several times slower due to the redundancy of interpolation and memory access, however the speed is increased by 40% when the maximum luminance block is splited to 32x32 and the maximum chromaticity block is splited to 16x16.

As shown in Table 5, we use the open source AVS2 decoder davs2 whose SIMD optimization is turned on and the GPU-based decoder proposed in this paper to decode these coded streams, and compare the inter prediction time which is the average of the 4 QPs. After the GPU optimization, the inter prediction speed has been significantly improved, and the decoding speed for Ultra HD 4K sequence is 9 times higher than the speed of the reference software, as for WQXGA and full HD, there is also a 6-time speed improvement.

Table 4 Speed of different size of the prediction block split.

sequence	total frame	qp	64x64 (ms)	32x32 (ms)	8x4/4x8 (ms)
pku_girls 3840x2160@50Hz	150	27	295.0	182.4	1463.8
		32	333.0	190.5	1453.0
		38	340.2	189.9	1453.0
		45	331.9	186.6	1467.9
pku_parkwalk 3840x2160@50Hz	150	27	270.8	166.7	1463.5
		32	265.4	166.7	1433.7
		38	275.9	169.9	1434.9
		45	281.9	169.9	1442.3
average	150		299.3	177.8	1451.5

Table 5 Speed comparison of GPU and CPU in inter prediction.

sequence	total frame	CPU		GPU		Speedup (ratio)
		time (ms)	average (ms)	time (ms)	average (ms)	
pku_girls 3840x2160 @50Hz	150	1696.6	11.31	187.4	1.25	9.1
pku_parkwalk 3840x2160 @50Hz	150	1559.1	10.39	168.3	1.12	9.3
Traffic 2560x1600 @30Hz	300	1374.6	4.58	223.8	0.75	6.1
BasketballDrive 1920x1080 @50Hz	500	1355.4	2.71	225.3	0.45	6.0
beach 1920x1080 @25Hz	250	571.3	2.29	97.2	0.39	5.9
taishan 1920x1080 @25Hz	250	687.6	2.75	113.2	0.45	6.1
Kimono 1920x1080 @24Hz	240	651.1	2.71	106.7	0.44	6.1
Cactus 1920x1080 @50Hz	500	1120.7	2.24	216.7	0.43	5.2

5. CONCLUSIONS

This paper proposes a GPU-based inter prediction scheme for video decoders. Through optimizing task allocation and thread scheduling on the GPU side, the inter prediction is well accelerated. These strategies can also be used to accelerate any video decoders, such as AVS2, HEVC and VP9 [9] etc..

6. REFERENCES

[1] Siwei Ma, Tiejun Huang, and Gao Wen, "The second generation ieee 1857 video coding standard," in *IEEE China Summit & International Conference on Signal & Information Processing*, 2015.

[2] Gary J. Sullivan, Jens Rainer Ohm, Woo Jin Han, and Thomas Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2013.

[3] pkuvcl, "davs2," `https://github.com/pkuvcl/davs2`.

[4] Mihir Mody, "Hevc video encoder & decoder architecture for multi-cores," in *International Conference on Signal Processing & Communications*, 2014.

[5] Ching Chi Chi, Mauricio Alvarez-Mesa, Benjamin Bross, Ben Juurlink, and Thomas Schierl, "Simd acceleration for hevc decoding," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 25, no. 5, pp. 841–855, 2015.

[6] Diego F. De Souza, Aleksandar Ilic, Nuno Roma, and Leonel Sousa, "Gpu acceleration of the hevc decoder inter prediction module," in *IEEE Global Conference on Signal & Information Processing*, 2016.

[7] BiaoWang, MauricioAlvarez-Mesa, ChiChi, BenJuurlink, Diego Souza, Aleksandar Ilic, Nuno Roma, and Leonel Sousa, "Efficient hevc decoder for heterogeneous cpu with gpu systems," in *IEEE International Workshop on Multimedia Signal Processing*, 2017.

[8] pkuvcl, "xavs2," `https://github.com/pkuvcl/xavs2`.

[9] Debargha Mukherjee, Jingning Han, Jim Bankoski, Ronald Bultje, Adrian Grange, John Koleszar, Paul Wilkins, and Yaowu Xu, "A technical overview of vp9—the latest open-source video codec," in *Technical Conference & Exhibition*, 2015, pp. 44–54.