ATLANTIS PRESS

Advances in Computer Science Research, (ACSR), volume 52

2016 International Conference on Computer Engineering and Information Systems (CEIS-16)

The Improved Full Search Algorithm for Motion Estimation with GPU Accelection

Zhuo Liu, Yi-Nan Lu, Tao Liu, Tian-Wen Yuan

College of Computer Science and Technology, Jilin University, Changchun, China

E-mail: liuzhuo14@mails.jlu.edu.cn, luyn@jlu.edu.cn

***Abstract*-Robust motion vector estimation algorithm plays an important role in video processing. Block matching algorithm for motion estimation as a mature search technique has been widely used in video applications because of its simplicity and easy implement. To increase the accuracy of motion vector prediction, an improved motion estimation algorithm is proposed that combines full search algorithm with spatial and temporal correlation. To further reduce the computational cost, a powerful parallel algorithm based on GPU is designed by using a series of optimization methods. Experimental results show that the proposed method gives the high PSNRs by testing the video sequences and the computation speed of the optimized algorithm is greatly improved.**

Keywords-motion estimation; block matching; full search; spatial and temporal correlation; parallel computing

I. INTRODUCTION

It is important and challenging to design motion estimation (ME) unit in real-time video applications such as video coding and video compression. ME is a process of estimating the motion vector field of a scene or a target by exploiting and reducing the redundant information in the video sequence. It is generally extremely complex and time-consuming. Block matching algorithm (BMA) has become the main method of motion estimation and been adopted as current video coding standards, because it provides a simple and straight way to realize in hardware.

BMA first divides the entire current frame into small rectangular blocks of pixels, and finds the best possible matching block (MB) in the reference frame within a given search area for each block in the current frame through a certain well-defined matching criteria.

Block matching algorithms can be divided into two classes, namely Full Search (FS) algorithm and fast search algorithm [1]. Full search algorithm is the most accurate, but slow, which results in difficulty in real-time implementation.

Many fast search algorithms have been developed in the last two decades. The shape and size of the search pattern used in these methods affect mostly to the performance of the search. Three-step Search (TSS) algorithm reduces the time complexity of motion estimation, but it is easy to fall into local optimal solution and prediction accuracy is not high enough; New Three-step Search (NTSS) algorithm improves the search speed by using the center–biased distribution property of motion vector, but the first step length is very large; The Diamond Search (DS) method can be used to search matching in a large space, which can avoid fall into local minimal effectively, but the shape of the hexagonal template is large and close to a circle; Hexagon-based search (HEXBS) method spans uniformly in all directions and converges faster than the diamond search.

Haan et al. [2,3] proposed a three dimensional recursive search (3DRS) motion estimation algorithm to estimate accurately the motion vector of current block. Due to taking the spatial and temporal correlation between the blocks into account, 3DRS converges faster. Once the scene is changed, the time candidate vectors will reduce the accuracy of motion estimation and the previous deviation will lead to the accumulation of errors [4].

Full search algorithm has the highest accuracy, but because it requires a large amount of computation, it is not suitable for real-time computing, so in the field of motion estimation, the parallel algorithm for searching motion vector is designed in the full search algorithm [5,6]. It is important to make full use of the high computing capability of Graphic Processor Unit (GPU) and map traditional sequential algorithms for motion estimation to parallel computing model by considering the memory architecture of GPU [7,8].

In this paper we present an improved full search motion estimation algorithm. This improved algorithm introduces the concepts of the spatial and temporal correlation from 3DRS algorithm into the full search. It can ensure high accuracy of motion vector estimation and provide the true movements of objects as well as. So in order to reduce the time complexity of the above algorithm, the improved block-based matching algorithm can be realized using the parallel algorithm based on GPU by effectively executing hundreds of threads per clock cycle with a large number of Processing Elements. Section 2 describes the improved motion algorithm, and Section 3 presents the implementation of an efficient parallel algorithm. The results are shown in Section 4 followed by the conclusion in Section 5.

II. THE IMPROVED ALGORITHM

The search process for BMA is finding the best match of the current frame macro-block in the reference frame. The computation of the Sum of Absolute Difference (SAD) for a given location (p, q) with the search window (SW) is done as:

$$SAD(p,q)=\sum_{k=1}^{m}\sum_{l=1}^{n}\left|U(k,l)-U_R(k+p,l+q)\right| \qquad (1)$$

Where the size of the SW is predefined, $U(k, l)$ is the current MB of size $m\times n$ at the coordinate location (k, l), $U_R(k+p, l+q)$ is the reference MB within the SW at the coordinate location ($k+p$, $l+q$) and the associated location (p, q) is the MV of this current MB.

The term $|U(k, l)-U_R(k+p, l+q)|$ is known as the distortion value which is the absolute difference in intensity between the current pixel $U(k, l)$ and the reference frame pixel $U_R(k+p, l+q)$.

In full search BMA, the one with the smallest SAD among all the search candidates within the SW is selected as the best search candidate. The details of computing the motion vector of each block in the current frame are as follows:

(1) For each candidate block within the search window, calculates its SAD value according to Equation (1).

(2) Compare the SAD values with the threshold set previously, if they are greater than the threshold, indicating that no candidate within this SW can be matched with the current block, the motion vector of the current block is set to 0. Otherwise, the block with the minimum SAD value is the best matched search candidate and the MV is obtained.

(3) Apply the median filtering method to remove discontinuous motion vectors. Select the median from the MV of the current MB and its eight neighborhoods as the final MV of the current MB to eliminate the blocking artifacts.

This improved algorithm strengthens the spatial and temporal correlation between the blocks. The process of the improved algorithm is as follows:

(1) In the current frame, selecting the upper left and upper right blocks of the current central block as the spatial blocks to be matched, named B1, B2 and select the best search candidate block from the previous frame matched with the current central block and its lower left, lower right as temporal blocks to be matched, named B3, B4, B5. A total of five blocks are selected as shown in Fig. 1.

(2) The MVs of the above five blocks are calculated using the above full search process, the smallest SAD among five values is found, finally the associated MV of the block is obtained.

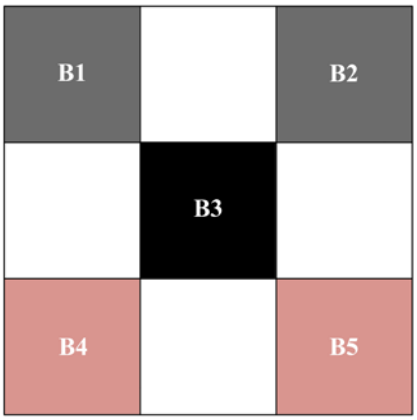

Figure 1. Relative positions of five blocks.

III. THE IMPLEMENTATION BASED ON GPU

Improved BMA algorithm is inefficient and it cannot fully meet the real-time applications. Therefore, the above proposed algorithm needs to be optimized on GPU platform to ensure real-time operation. In this paper, we use the OpenCL platform for GPU programming [9]. This section explains the various optimization methods to implement the parallel algorithm.

The block used in this paper is the size of 4×4 pixels. The search window is set to 16×16 blocks, and has 256 candidate search locations for the current block. SAD(p, q) is calculated by adding all the distortion values for each block, and assigned a thread. A thread block has 256 threads executed simultaneously, and one thread computes a SAD for the search position (p, q). We use the reduction way to process the summation, for $n\times n$ size of a block, divide the block into two sections, each section has $n^2/2$ size, and add the corresponding SAD values of each section; and for the result vector, again divide it into two sections and add them, the same process continues to do until the final SAD is obtained. Next using merge sorting strategy to sort all the SAD values within the search window, and finally get a minimum SAD value, and obtain the motion vector of the current block.

In this paper, we use the fast median filtering algorithm [10, 11]. To sort each column, and sort each row, at last, the diagonal, the algorithm takes the middle value as the output, and uses the characteristics of texture memory and register to reduce the numbers of comparison among the elements. This process is illustrated in Fig. 2.

Here, $S_0 \sim S_8$ are the 9 pixel values in the 3×3 window, Max, Med and Min stand for the maximum, the median and the minimum vectors for corresponding rows after column sorting. A(i), B(i), C(i), stands for the value (maximum, the median or the minimum) in three rows respectively after rows sorting in descending order. D is the vector obtained after the three pixel values C_0, B_1 and A_2 on the diagonal sorting, and D_1 is the median value.

In the fast median filtering algorithm on GPU, the program needs to read massive data and the data are accessed in the texture memory instead of shared memory within the same block or global memory, so it can avoid repeat accessing the same data and its accessing speed can reach the speed level of the register in GPU. The optimized fast median filtering algorithm reduces the number of comparisons between the data. Because the general sort algorithm has lots of cycle and judgements, this paper uses inline function as kernel to compare two numbers. In this way, a thread executing repeatedly the inline function can get maximum value, minimum value and the intermediate value.

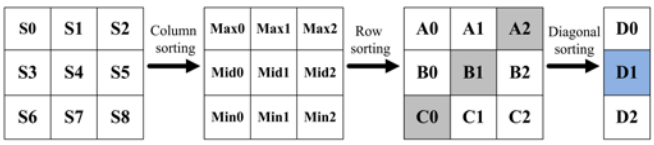

Figure 2. Sorting process.

IV. EXPERIMENTS

In our experiments, we use Intel (R) i5 450M 2.40GHz with 2G memory, NVIDIA Geforce 610M graphics card and Windows 7 Ultimate Edition. Tests are performed for

standard video: the Container, Foreman, Coastguard, Tennis, Football, Stefan with QCIF formats.

We calculate the average Peak signal-to-noise ratio (PSNR) for the compensation images of each frame by the motion vectors to test the accuracy of the motion vectors. For the images after motion compensation, the average PSNRs are listed in Fig. 3. We can see that the average PSNR for the several test video sequences using our method are shown is higher than the traditional methods. Our method has a higher PSNR. The results show the reliability of the new algorithm and prove the effectiveness of the algorithm and the rationality of the block size of 4 × 4.

The comparisons in PSNR for the same video sequences are shown in Fig. 4 and Fig. 5 for FS algorithm and the improved algorithm to obtain the motion vector. We select 20 frames, and compare the PSNRs of the 20 frames after compensation. From the figures we can see, in terms of the small amount of movements for the Container sequence, the PSNRs of the improved algorithm and FS algorithm are roughly the same; and in terms of the large amount of movements for the Football sequence, the improved method is better than the FS algorithm, which demonstrate the effectiveness of the new algorithm. Fig. 4 and Fig. 5 respectively show the different PSNRs of the Container sequence and Football sequence using FS and our method.

Using the improved motion estimation algorithm for the above standard sequences, we test their running time on CPU and GPU, separately. Set the resolution of the video sequence to 640 × 480, with time (ms) per frame required for representation, as shown in Table 1. The speedups of the improved algorithm running on the GPU are improved several times when compared to the algorithm on the CPU.

We also test the Foreman sequences with various resolutions, and obtain the different fps values based on CPU and GPU. The results in Table 2 show that the parallel algorithm produces a speedup of ten times comparing to the algorithm using CPU, and can meet the real requirement.

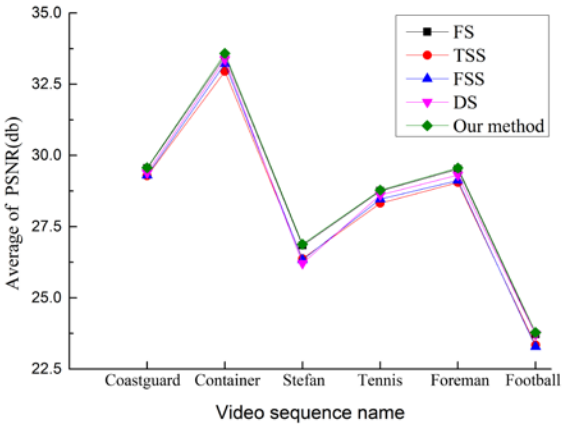

Figure 3. The average PSNR of each algorithm for different test sequences.

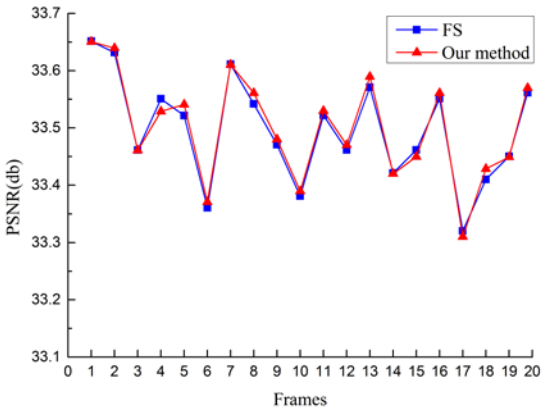

Figure 4. Container sequence(PSNR:db).

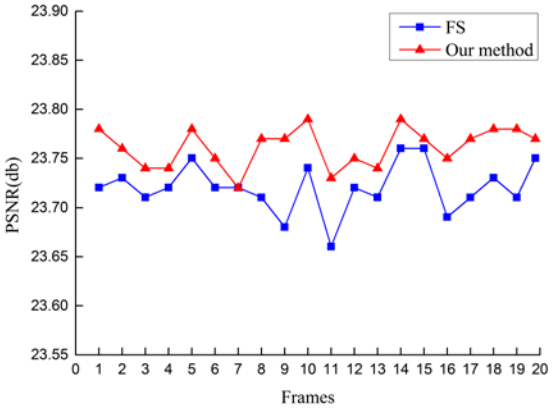

Figure 5. Football sequence(PSNR:db) .

TABLE I. THE TIMING COMPARISON ON CPU AND GPU IMPLEMENT OF IMPROVED（MS）（640×480）

	Container	**Foreman**	**Coastguard**	**Tennis**	**Football**	**Stefan**
CPU	302	372	357	344	368	385
GPU	12.31	16.11	14.82	14.41	16.05	15.84
Speedup	24.55	23.09	24.12	23.89	22.93	24.31

TABLE II. PERFORMANCE COMPARISON OF THE ALGORITHMS ON GPU AND CPU WITH DIFFERENT RESOLUTIONS

resolutions	**CPU(fps)**	**GPU(fps)**	**Speedup**
176*144	12.156	285.909	23.52
352*188	4.623	151.980	32.87
640*480	0.821	41.066	50.02
720*576	0.425	26.451	62.23

V. CONCLUSION

This paper presents an improved full search algorithm based on block matching and spatial and temporal correlation, and elucidates a parallel design based on GPU, which mitigates the computational cost. Experimental results show that the improved algorithm obtains the higher PSNR comparing to the other methods. Due to the GPU powerful computing, the improved algorithm has been further optimized. Still a lot of work will be needed in the future. We will explore how to further improve the search efficiency when very little change will occur in two adjacent frames, and how to rationally allocate the threads and quickly calculate each thread.

ACKNOWLEDGMENT

Supported by Specialized Research Fund for the Doctoral Program of Higher Education, China (No. 20130061110054)

REFERENCES

[1] M Brunig, W Niehsen, “Fast full-search block matching,” IEEE Transactions on Circuits and Systems for Video Technology. Piscataway, NJ, vol. 11, pp. 241–247, Feb. 2001.

[2] G. de Haan, Paul W. A. C. Biezen Henk Huijgen and Olukayode A. Ojo, “True-Motion Estimation with 3D Recursive Search Block Matching,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 3, pp. 368–379, Oct. 1993.

[3] G. de Haan, and P. W. A. C. Biezen, “Sub-pixel motion estimation with 3-D recursive search block-matching,” Signal Processing, vol. 6, pp. 229–239, 1994.

[4] Tian Tang, Jin Wang, Yunqiang Liu, Xiaokang Yang and Yizhi Gao , “Adaptive frame recovery based on motion activity,” Biochemistry, vol. 38, pp. 692–697, 2007.

[5] Z Jing, J Liangbao and C Xuehong, “Implementation of parallel full search algorithm for motion estimation on multi-core processors,” International Conference on Next Generation Information Technology. Korea, vol. 22, pp. 31–35, Jan. 2011.

[6] X Gan, Shen, AW Zhiying, “Parallelizing Full Search Algorithm for Motion Estimation Using CUDA,” Journal of Computer-Aided Design and Computer Graphics. Beijing, vol. 22, pp. 457–460, 2010.

[7] Du P, Weber R, Luszczek P, “From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming,” Parallel Computing. Amsterdam, vol. 38, pp. 391-407, Aug. 2012.

[8] Rodriguez-Sanchez R, Martinez J, Fernandez-Escribano G, “Optimizing H.264/AVC interprediction on a GPU-based framework,” Concurrency and Computation Practice and Experience. Hoboken, vol.24, pp. 1607-1624, Jan. 2012.

[9] J. Stone, D. Gohara, and G. Shi, “OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems,” Computing in Science & Engineering, vol. 12, pp. 66–72, 2010.

[10] Yu-qiang Fu, “Research and Hardware Design of Image Processing Algorithms Based on FPGA,” thesis of Nanchang University , 2006.

[11] Ya-fei LV, Kun-yang JIA, “Fast Median Filtering Algorithm Based on CUDA,” Modern Computer, vol. 38, pp. 3–6, 2011.