http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (7): 1449-1453, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1449.1453

A Fast Image Matching Algorithm Based on GPU Parallel Computing

Li Haiyang, He Hongzhou and Wen Yongge
College of Mathematics and Computer Science, Mianyang Normal University, 621000, China

Abstract: In process of image matching, Scale Invariant Feature Transform (SIFT) algorithm is one of the best performance algorithms. But the drawback of being complex and time-consuming limits its application in more fields. Aiming at this shortage of SIFT algorithm, this study proposes a fast SIFT algorithm based on Graphic Processing Unit (GPU) and analyzes its parallelism. It is further optimized according to the detailed analysis on the thread and memory model of the graphic hardware. It is experimentally shown that the speed of the proposed algorithm is 25-45 times faster than the original algorithm. Processing 640×480 images, its speed can be up to 24 frames per second. The conclusion is that proposed algorithm can meet the needs of real-time applications.

Key words: Scale invariant feature transform, graphic processing unit, parallel computing, real-time performance, image matching

INTRODUCTION

Image matching is to identify the same name point between two or more images through matching algorithm. It is a premise of many applications. The SIFT algorithm (Lowe, 2004) is scale space-based, image scaling, rotating and even invariance maintained in affine transformation. Mikolajczyk and Schmid (2005) proved it to be one of the best performance algorithms. Using the method of Principal Component Analysis (PCA) to reduce SIFT feature descriptor dimension (Ke and Sukthankar, 2004) shows that SIFT algorithm stability can be improved. Using global context of SIFT feature descriptor (Mortensen *et al*., 2005) shows that the matching rate of SIFT algorithm can be improved. Using Gabor filter to improve SIFT feature descriptor (Moreno *et al*., 2009) shows that SIFT algorithm can be applied to accuracy higher scenes.

On GPU and multi core CPU environment, the implementation and optimization of SIFT algorithm is becoming a research focus in recent years. Using OpenGL and NVIDIA 7900GTX graphics card to process 640*480 images (Sinha *et al*., 2011) shows that the algorithm execution speed can be up to 10 Frames per Second (FPS). Using OpenMP and Intel 8-core CPU to process 640*480 images (Zhang *et al*., 2008) shows that the algorithm execution speed can be up to 45 FPS.

Although, there has been a significant improvement for SIFT, it is difficult to meet the real-time applications. Based on NVIDIA Compute Unified Device Architecture (CUDA), a fast SIFT algorithm is presented in this study. In order to take advantage of the GPU's superiority in image processing, the GPU is chosen as computing units. Relative to the traditional OpenGL and GPGPU programming, CUDA has broader applicability. It provides a more intuitive programming model and optimization principles and its code can run on a multi-core CPU. Taking advantage of the GPU parallel computing capability, proposed algorithm can meet the real-time applications.

PARALLEL DESIGN

Task division: In the division of tasks, the following principles should be taken into account:

- According to the scale of problem, the most large-scale and time-consuming data should be paralleled and steps with high computational intensity should be mapped to the GPU
- Considering the parallel threads on the CPU are coarse-grained while the parallel threads on the GPU are fine-grained, the parallel thread on the CPU design can not be simply mapped to the GPU
- PCI-E bus is responsible for communication between the host and the device, but its bus bandwidth is far less than that of the graphic memory. As a result, the GPU should be arranged to complete calculation as much as possible between the two hosts and devices

Based on the principles stated above, CUDA realization is divided into two parts. One is SIFT feature point of detection and the other is the SIFT feature point matching. The CPU side is used to create and initialize the

Corresponding Author: LI Haiyang, College of Mathematics and Computer Science, Mianyang Normal University, 621000, China Tel: (0086) 13881174125

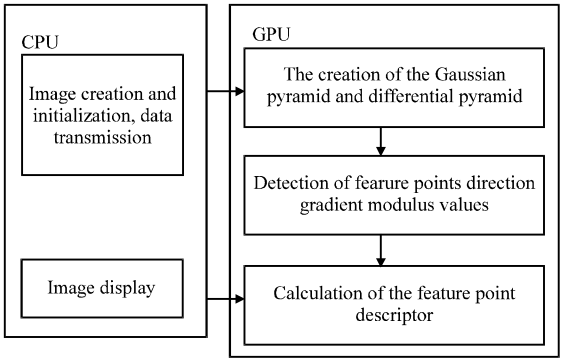

Fig. 1: Task division of CPU and GPU

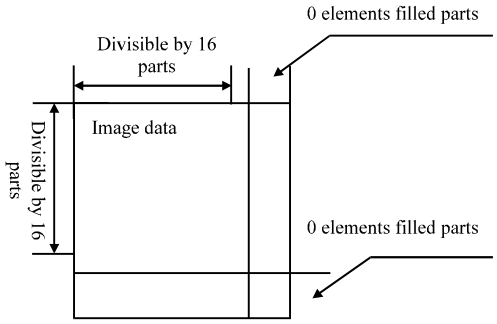

Fig. 2: Feature point extraction operation

image because of small amount of computation and the GPU side is used to process data because of large amount of computation. A module corresponds to a kernel function. This design can take full advantage of the CUDA platform's characteristics to deal with intensive data and reduce application run time, consequently improve program performance. The task division of CPU and GPU is shown in Fig. 1.

PARALLEL IMPLEMENTATION

Scale space build

Scale space build consists of three steps: Gaussian filtering, image transformation and Difference of Gaussian operator (DoG). Gaussian filtering is a process of the most time-consuming. Mikolajczyk and Schmid (2005) gave an algorithm as follow: Assuming DoG Level S = 3 and 6 images generated within each Octave. Besides the original image, it needs 5 times Gaussian filtering. Before each Gaussian filtering, the one-dimensional Gaussian kernel which computed on the CPU side is uploaded to the global memory or constant memory. Then, two kernel functions have been successively activated. Finally, in the kernel function, the Gaussian kernel and the image data is read to complete the convolution filtering.

In practical applications, the memory bandwidth often can not meet the need. For one-dimensional Gaussian filtering which standard deviation is σ and window width is $6\sigma+1$, it needs to perform $3\sigma+1$ memory access to read Gaussian kernel and $6\sigma+1$ memory accesses to read neighboring pixel values for each pixel. In this case, completing the convolution filter calculation needs execute $9\sigma+2$ times the memory read and $6\sigma+1$ MAD instruction in additional. This indicates that the performance of the algorithm is restricted by memory bandwidth seriously.

The solution to the above is to synchronize in the calculation process to generate Gaussian kernel, thus the memory read can be reduced. By increasing $6\sigma+1$ times multiplications, the memory read is reduced by $3\sigma+1$ times in this algorithm. To some extent, the solution helps to alleviate the pressure on the memory bandwidth.

For a difference of Gaussian pyramid which has O Octaves and each Octave includes I intervals, the number of points between Octaves are different, they are relatively independent. Therefore, the feature point detection between the Octaves can be performed separately. Because one execution of Kernel function can be used to calculate a set of the image feature point in the Gaussian pyramid, all calculations can be finished by loop O times.

Since the image is two-dimensional, the block of a Grid should be defined as two-dimensional. Each block is responsible for calculating 16×16 = 256 pixels in each interval image. To an image which size is height×width pixels, if its height and the width are not divided by 16, it would need (height/16+1)×(width/16+1) block. In order to calculate the excess branches judgment, it needs to judge the data out of range and generate additional branch judgment. In this case, the program efficiency will be greatly reduced. In order to reduce the branch judgment in the block, for each image of height and width not be divided by 16 in DOG pyramid interval, some zero element are added in its right side and the lower side so as to its height and width are both divided by 16. In this case, it does not require additional branch judgment in the Kernel function, so program efficiency will be improved. The operation is shown in Fig. 2.

If the DOG pyramid intervals is I, there set loops I within the Kernel function. Because one execution of each loop can be used to calculate one interval image data, all image data in DOG pyramid can be finished by loop I times. If the maximum number of threads for each block is set as 256, a thread can be responsible for the calculation of a pixel. That the maximum number of threads is set as 256 can reduce the branch judgment.

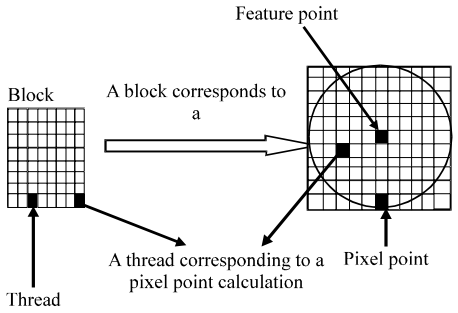

Fig. 3: Direction gradient computation

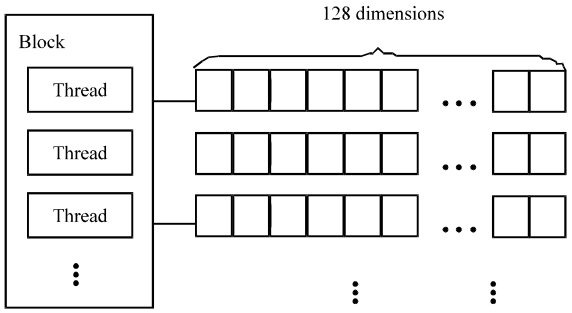

Fig. 4: Task division of the feature matching (a) Experimental results 1 and (b) Experimental results 2

Feature point direction calculation: In calculating the direction of the feature point (Mikolajczyk and Schmid, 2005) calculates and saves the gradient direction and magnitude before calculating the direction of the feature point. The gradient calculation is a step of calculating feature points and is combined in the same kernel function. In this case, calculating the feature descriptor requires calculating gradient additionally and this will reduce the efficiency of the algorithm performed.

Proposed algorithm is to make the histogram which is stored in the shared memory and use atomicAdd function of the CUDA to achieve a cumulative atomic operation in shared memory. This will not cause memory inconsistencies between threads. The direction gradient computation is shown in Fig. 3.

The steps of calculating direction are as follows: Firstly, each of the feature point which on a feature point list should be mapped to a thread block. Each thread corresponds to Gauss image of a neighborhood pixels point. Then, the gradient direction and size are calculated and accumulated to the histogram which packet number is 36 and the histogram of the gradient has been completed. The gradient histogram of feature points is put on the thread block shared memory. Finally, the histogram stored in the shared memory is traversed by a thread and the direction of maximum statistical value is obtained.

Feature descriptor calculation: Just like calculating the direction of the feature point, the feature point neighborhood size is set as 16×16 pixels. Each of 4×4 neighborhood statistics gradient direction can generate a histogram which packet number is 8. Thus, a 4×4×8 = 128 feature descriptor can be obtained. When calculating the gradient direction, the direction of the feature points should be taken into account in order to obtain the rotation invariant feature. After the normalization processing, the image brightness variations invariance of the feature descriptor can be kept up to output. Each feature point corresponds to a 16×16 thread block and each thread inside a thread block corresponds to a sampling point. Gradient information is accumulated to descriptor stored in the shared memory area. Finally, the feature descriptor is copied to the global memory output from the shared memory.

Feature matching: The essence of SIFT feature point matching is to search on the KD-tree. CUDA multithreading can make the execution concurrently to improve program performance in the process of looking for the nearest neighbors. According to the task division in 3.1, the creation of KD-tree is put on the CPU side and the search of feature points and the judgments of matching points are put on the GPU side. In the GPU-side implementation, a thread corresponds to a feature point search and judgment. A block is responsible for the calculation of 128 feature points. The task division of the feature matching is shown in Fig. 4.

It is dynamic for the allocation of block number. For an image with a large number of feature points, it is calculated simultaneously by multiple blocks instead of loops. In this case, its degree of parallelism can be increased. For an image with its feature point's number less than 128, the zero elements padded operation can be done to meet the minimum computing needs of a block. In this case, the extra judgment can be eliminated in order to reduce the execution time of the Kernel.

Before the execution of the kernel function, the created KD-tree on the CPU side is transmitted to the texture memory. The speed of data access can be improved by this method.

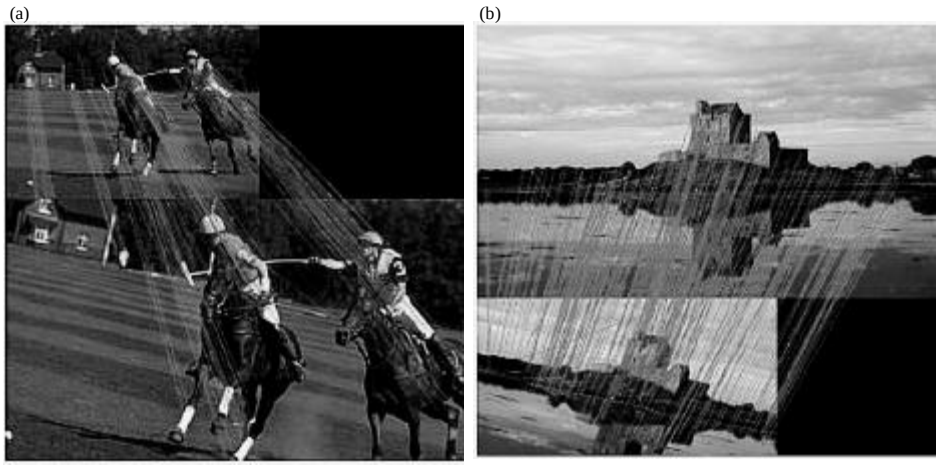

Fig. 5(a-b): Matching results of proposed algorithm

EXPERIMENTAL VERIFICATION

Experimental setup:

- **Operating system:** 64-bit Microsoft Windows 7
- **Memory capacity:** 8GB
- **CPU:** Intel Core 2 Duo P8600@2.40 GHz
- **GPU:** NVIDIA GTX260, 192 stream processors inside

Algorithm parameter settings: The DoG interval number of each Octave is 3 and the number of Octave is automatically selected by the algorithm according to the size of the image.

Parameters in experiment 1: The picture size is 256 kB; the picture dimensions are 640×480 pixels; the picture resolution is 96 dpi; the number of Octave is 5.

Parameters in experiment 2: The picture size is 2785 kB; the picture dimensions are 2048×1536 pixels; the picture resolution is 96 dpi; the number of Octave is 7.

Experimental results: In conditions of different scale and rotation transformation, the two experimental results of image matching are shown in Fig. 5.

In Fig. 5a, the picture scale variation is 2.0 and its viewing angle range is 15°. The number of matching points is 136 pairs and 132 pairs of them are properly. The properly rate is 97.1%. In Fig. 5b, the picture scale variation is 0.6 and its viewing angle range is 15°. The number of matching points is 246 pairs and 239 pairs of them are properly. The properly rate is 97.2%. It shows that the proposed algorithm preserves the robustness of the SIFT algorithm.

Table 1: Experimental results of two algorithms performance

	Experimental 1		Experimental 2	
	Original	Proposed	Original	Proposed
Data transmission		4.00		12.73
Scale space build	735.69	12.45	7669.35	57.60
Extracting points	72.30	16.46	415.32	61.24
Direction calculation	63.23	10.50	98.06	27.83
Descriptor calculation	795.80	16.78	8213.60	42.97
Feature matching	87.52	8.62	135.41	186.78
Total time	1754.54	64.81	16531.7	376.42
Frames per second	0.91	24.60	0.08	3.50

Contrasting with CPU reference implementation the results are shown in Table 1.

The experimental results show that the performance of proposed algorithm exceeds that of the original algorithm by a great deal. In Experimental 1, the total time of original algorithm is 1754.54 msec and proposed algorithm is 64.81 msec. The difference of them is 1689.73 msec. In experimental 2, the total time of original algorithm is 16531.7 msec and proposed algorithm is 376.42 msec. The difference of them is 16155.28 msec. Proposed algorithm is about 25-45 times faster than the original CPU algorithm.

Processing 640×480 images, original algorithm speed is 0.91 FPS while proposed algorithm can be up to 24.60 FPS. Obviously, the original algorithm can not meet the needs of real-time applications while proposed algorithm can do this.

CONCLUSION

A fast SIFT algorithm is proposed to improve the SIFT matching algorithm to meet the needs of real-time applications. This algorithm is based on the GPU parallel computing. By this algorithm, not only the robustness of the SIFT algorithm is retained, but also is its real-time performance improved. It is experimentally proved that its matching effect is stable and can meet the needs of real-time applications.

ACKNOWLEDGMENTS

This study was sponsored by the Education Foundation of Sichuan, China under Grant No. 12ZB070.

REFERENCES

Ke, Y. and R. Sukthankar, 2004. PCA-SIFT: A more distinctive representation for local image descriptors. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 2, June 27-July 2, 2004, Pittsburgh, PA., USA., pp: 506-513.

Lowe, D.G., 2004. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision, 60: 91-110.

Mikolajczyk, K.I. and C. Schmid, 2005. A performance evaluation of local descriptors. IEEE Trans. Pattern Anal. Mach. Intell., 27: 1615-1630.

Moreno, P., A. Bernardino and J. Santos-Victor, 2009. Improving the SIFT descriptor with smooth derivative filters. Pattern Recog. Lett., 30: 18-26.

Mortensen, E.N., D. Hongli and L.G. Shapiro, 2005. A SIFT descriptor with global context. Proceedings of the IEEE Computer Vision and Pattern Recognition, June 20-26, 2005, San Diego, USA., pp: 184-190.

Sinha, S.N., J.M. Frahm, M. Pollefeys and Y. Genc, 2011. Feature tracking and matching in video using programmable graphics hardware. Mach. Vision Appl., 22: 207-217.

Zhang, Q., C. Yurong, Z. Yimin and X. Yinlong, 2008. SIFT implementation and optimization for multi-core systems. Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, April 14-18, 2008, Miami, USA., pp: 1-8.