Journal of Computational Information Systems 9: 17 (2013) 6937–6944
Available at http://www.Jofcis.com

Implementation of Parallel Computing FAST Algorithm on Mobile GPU

Chienhsing CHOU*, Peter LIU, Taiyi WU, Yihsiang CHIEN

Department of Electrical Engineering, Tamkang University, New Taipei City, Taiwan

Abstract

Corner detection is an extremely important technique in image recognition, which is widely employed in various applications for image recognition. With the widespread use of mobile devices, image recognition techniques are frequently applied in such devices. However, the hardware resource of smartphones is lacked and restricted, it is a difficult task to apply the techniques of corner detection smoothly in these devices. To enhance the computational speed, the FAST corner detection algorithm is implemented with parallel computing of GPU in mobile devices. In the experiments, the computational speed of the FAST corner detection algorithm increases 24 times after using GPU parallel computing. Compared with the widely known SURF algorithm, which is computed with mobile CPU only, the proposed technique in this study is 468 times faster than SURF algorithm.

Keywords: FAST Algorithm; GPU; Corner Detection; Mobile Device; SURF

1 Introduction

Corner detection is an extremely important technique in image recognition, which is widely employed in various applications for image recognition. Corner detection is applied to applications including 3D reconstruction [1], object recognition [2], image mosaicing [3], and motion estimation [4], etc. Since the debut of the iPhone in 2008, smartphones have become necessary products in the lives of many people. Because smartphone characteristics are portable and practical, more and more image recognition applications are appearing in these devices. Examples include using facial recognition to unlock the smartphone; combining augmented reality with image recognition in applications [5], using auto-stitching to merge multiple images into one [6]-[8], and using seam carving to adjust image sizes [9]-[11]. However, the hardware resource of smartphones is lacked and restricted, it is a difficult task to apply these recognition techniques smoothly in these devices.

To solve the above problem, the most straightforward solution is to employ GPU to assist the computation of the image recognition applications. In the latest version of the well-known open source computer vision library (OpenCV), part of the program already uses GPU to perform

*Corresponding author.
Email address: chchou@mail.tku.edu.tw (Chienhsing CHOU).

1553–9105 / Copyright © 2013 Binary Information Press
DOI: 10.12733/jcis6666
September 1, 2013

computations. Most methods employ CUDA[12] to execute numerous computations in GPUs[13]-[14]. However, these CUDA accelerated methods are only suitable for the latest NVIDIA graphics chips, can only run on standard PC platforms, and have no support for current mobile devices. In addition, the GPU inside smartphones may not use NVIDIA graphics chip, this also hinders researchers to transfer their research techniques to smartphones.

In order to enhance the computational speed during detecting corners, the FAST algorithm [15] is chosen as the corner detection method to detect corners in an image, because it requires fewer computations. Then this study implements the FAST algorithm in mobile devices by employing GPGPU technique to control GPUs. In the Section 2 of this study, we introduce GPUs and the FAST algorithm. In Section 3, we explain how to use the parallel processing characteristics of GPUs to implement the FAST algorithm. During the experiment in Section 4, we compare the method proposed in this study with other available methods. In Section 5, we present our conclusions.

2 Introduction to GPUs and the FAST Algorithm

2.1 Graphics processing units

In 1990, graphics processing units (GPUs) were employed to assist in the drawing of 2D images, such as for line drawing, tile rendering, and VCD/DVD playing acceleration. By the end of the 1990s, they were used in 3D accelerations. Around 2000, many affordable 3D acceleration display cards emerged in the market for 3D rendering. These 3D display cards reduce the computational loading for the central processing unit (CPU) while rendering 3D graphics. Before the existence of GPUs, all rendering computations were processed by the CPU. However, with GPU support, much of the computational load is now shared, increasing the efficiency of 2D and 3D image rendering for users.

The hardware architecture of GPUs is a multi-core CPU, with up to hundreds of cores. Because most of the pixels in the image are rendered independently, no sequential relationships typically exist between pixels. Thus, if a GPU contains 32 processing units, theoretically, the speed of rendering a single image might be 32 times faster than that of a single core CPU. Figure 1 is the architecture of CPU and GPU [16].

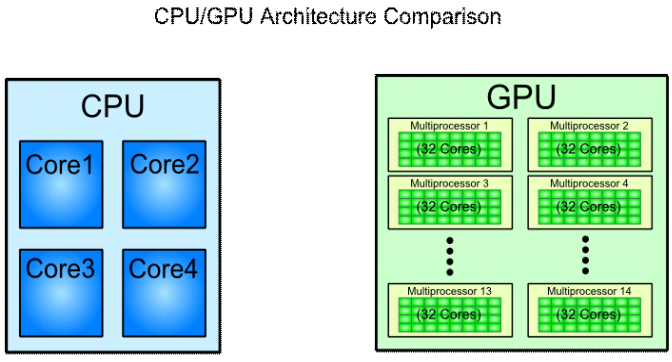


Fig. 1: The architecture of CPU and GPU [16].

Originally, GPUs were designed to render 3D graphics rather than to assist the computational tasks of variety image processing. However, in recent years, many researchers have used GPUs

on PC platforms to perform computations involving convolution, RGB-grayscale conversion, and Gaussian blurring to save significant amounts of time. For most researches, these image processing techniques were implemented by using the CUDA programming language. However, CUDA programming language cannot be directly applied on mobile devices; in this study, we used the general purpose GPU (GPGPU) technique [17] to control the tasks of GPUs in mobile devices.

2.2 FAST corner detection algorithm

The FAST corner detection algorithm was proposed by Edward Rosten [15]. Compared with several corner detection algorithms, the FAST algorithm requires fewer computational load while detection corner points in an image. The computational methods of the FAST algorithm is described as following (see Fig. 2 and Eq. (1)), more details can be found in [15].

$$S_p \to X = \begin{cases} d, & I_p \to x \leq I_p - t \ (dar\,\text{ker}) \\ s, & I_p - t < I_p \to x < I_p + t \ (similar) \\ b, & I_p + t \leq I_p \to x \quad (brighter) \end{cases}$$

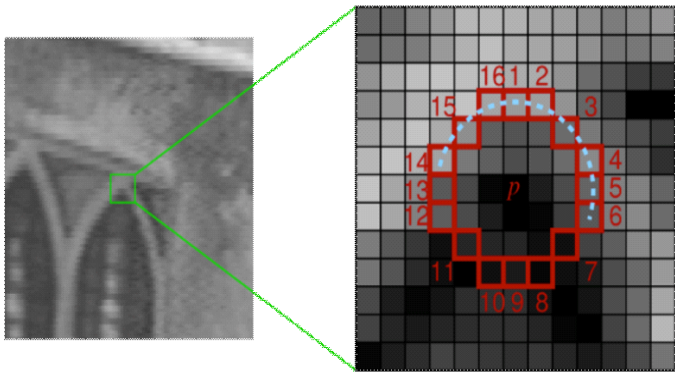


Fig. 2: The computational methods of the FAST algorithm [15].

Because the FAST algorithm offers the advantage of requiring few computations to detect corners, it is suitable for situations requiring real-time processing, such as extracting corner points from a video in real time. Furthermore, it is also suitable to be applied in mobile devices with limited computational resources.

3 The Research Method Combining GPUs and the FAST Algorithm

Currently, the graphics application programming interfaces (APIs) in mobile devices mainly use OpenGL Shading Language (GLSL) [18] from OpenGL ES 2.0. This graphics API is a high-level shading language based on the syntax of the C programming language. It provides the developer with more direct control of the graphics pipeline. Because of the development of this standard, modern 3D graphics can draw astonishing scenes that appear almost real. In this study, the GLSL is the main tool to implement the FAST algorithm by using GPUs.

This proposed technique comprises two stages (as shown in Fig. 3). The first stage is that convert the image pixels from RGB color space to grayscale, and then execute the FAST algorithm to detect corner point in the second stage. During both stages, the GPU is controlled by using GPGPU technique. The grayscale converting and FAST algorithm computations are conducted using parallel computing. To the best of our knowledge, no other researcher has used GPU in mobile platforms to perform the FAST algorithm. This is the main contribution of this study. The testing program is also available on our website for interested researchers to download and test [19].

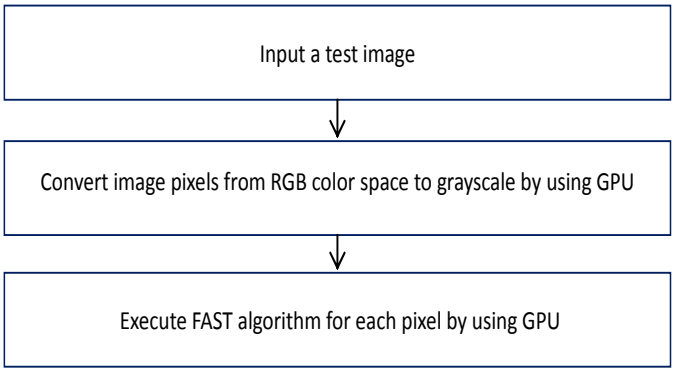


Fig. 3: Flowchart of using GPU to conduct the computations of grayscale converting and the FAST algorithm.

When developing the proposed method, the GLSL tool (Render Monkey) of PC could not be easily employed for testing, because of the differences in the specifications for mobile devices and desktop PCs. Therefore, the development procedure and debugging only relies the return messages from OpenGL, and the majority of the execution errors were solved through guesswork. This greatly increased the difficulty of development. In addition, because GPU needs convert RGB values from integers range (0∼255) to float-point range (0∼1.0), the conversion process resulted in some slight errors in the RGB values. Fortunately, the slight deviation has minimal effect on the overall results.

4 Experimental Results

The test image used for the experiment was Lena (Fig. 4). Because of the experiment requirements, this test image was stored in four sizes, namely, 512×512, 1024×1024, 1536×1536, and 2048×2048. The tested mobile device platforms were the iPhone 3GS and iPhone 4. Because of the limited memory in the mobile device, the largest image size during the experiments is 2048×2048. We conducted three experiments in total. The first experiment compared the computational speed of the FAST algorithm, which is executed with GPU acceleration, on various hardware platforms. The second experiment compared the computational speed of three corner detection methods. During the third experiment, we tested the computational speed using various images.

(1) Experiment 1:

For the first experiment, we used an iPhone 3GS and iPhone 4 as the test platforms to compare the computational speed of the FAST algorithm after GPU acceleration. Table 1 shows the

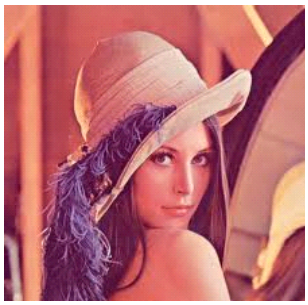


Fig. 4: The test image Lena

Table 1: The hardware specifications of the iPhone3GS and iPhone4.

	iPhone 3GS	iPhone 4
CPU	ARM Cortex-A8 833 MHz	Apple A4 1 GHz
Graphics	PowerVR SGX535 GPU	
Memory	256 MB DRAM	512 MB DRAM

hardware specifications of the iPhone 3GS and iPhone 4. The iPhone 4 has a faster CPU and more memory; however, both devices use the same graphics chip. For this experiment, we used four sizes of test images. The computational times are shown in Table 2. Based on the data in Table 2, we summarized the following two points:

1. Although the iPhone 4 is superior to the iPhone 3GS for certain hardware specifications, the computational time of FAST algorithm conducted by GPUs was almost the same because of the identical graphics chip used.

2. The required computation time did not increase linearly with increases in image size. For example, when the image size increased from 1024×1024 to 2048×2048, although four times as many pixels were used, the computational time was only 1.8 times longer. These result further validated the benefits of using GPU parallel computing.

(2) Experiment 2:

During this experiment, we compared the time required to compute the FAST algorithm using a GPU (FAST-GPU) or without (FAST-CPU). In addition, another well-known corner detection algorithm SURF (using CPU) was also compared. The experimental data are shown in Table 3.

Table 2: Experimental results of the iPhone 3GS and iPhone 4.

Image Size	iPhone 3GS (ms)	iPhone 4 (ms)
512x512	64	64
1024x1024	81	81
1536x1536	113	114
2048x2048	147	146

Table 3: Experimental results of three corner detection algorithms.

Image-Size	FAST-GPU (ms)	FAST-CPU (ms)	SURF-CPU (ms)
512x512	64	201	3,452
1024x1024	81	848	15,128
1536x1536	114	1,877	36,301
2048x2048	146	3,509	68,271

Figure 5 shows the experimental results of the three corner detection methods. From the data in Table 3, we summarized the following three points:

1. When the image size is 512×512, computational speed of the FAST algorithm using a GPU is three times faster than that without using a GPU. Compared with SURF, it is 54 times faster. When the image size is increased to 2048×2048, computational speed of the FAST algorithm increases by 24 times after using a GPU. Compared with SURF, the computational speed differ by as much as 468 times.

2. With the two CPU only methods, increases in image size are accompanied by linear increases in the required computation time. This indicates that parallel computing using GPUs is more effective for increasing speed when using larger images.

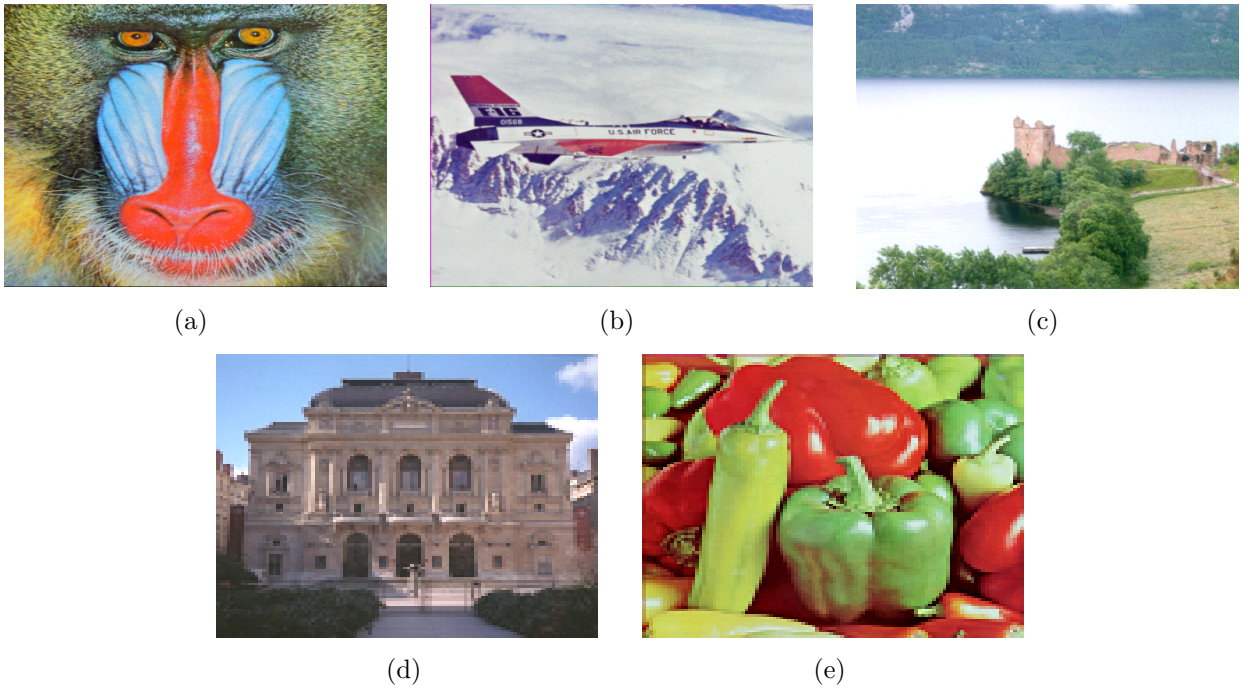
3. Two main reasons lead to the long computation time for SURF algorithm. One is that the SURF algorithm is complex. The other is that when SURF algorithm uses the CPU for computations, the OS of mobile must continuously reallocate memory because of the limited memory of mobile devices, resulting in additional computation time.

(a) (b) (c)

Fig. 5: The results of corner detection after using (a) FAST-GPU; (b) FAST-CPU; (c) SURF-CPU

(3) Experiment 3:

For this experiment, we used various images to assess our proposed methods. Figure 6 shows the five test images used in this experiment [17]. Table 4 lists the average time required to process these images using FAST-GPU. By comparing the experimental data in Table 3, we found that no significant differences in computation times existed. This also indicates that our method is not affected by various test images.



(a) (b) (c)

(d) (e)

Fig. 6: The results of corner detection after using (a) FAST-GPU; (b) FAST-CPU; (c) SURF-CPU

Table 4: The average time for processing five test images by using FAST-GPU.

Image-Size	FAST-GPU (ms)
512x512	64
1024x1024	81
1536x1536	114
2048x2048	147

5 Conclusions

To increase the computational speed of corner detection on mobile devices, we used the GPGPU method to control GPUs in this study. Through parallel computing, we implemented the FAST corner detection algorithm in mobile devices. The parallel computing characteristic of GPUs enabled us to increase the computational speed by 24 times. Compared with the well-known SURF algorithm, the speed of our proposed method was 468 times faster. In addition, our method performed satisfactorily both when processing different images and when using different mobile devices. The testing program is also available on our website for interested researchers to download and test [19].

References

[1] M. Pollefeys, R. K. M. Vergauwen, L. V. Gool, Automated reconstruction of 3D scenes from sequences of images, ISPRS Journal of Photogrammetry and Remote Sensing 55 (2000) 251-267.

[2] D. G. Lowe, Object recognition from local scale-invariant features, in: Proceedings of IEEE Inter-

national Conference on Computer Vision, 2, 1999, pp. 1150-1157.

[3] P. F. McLauchlan, A. Jaenicke, Image mosaicing using sequential bundle adjustment, Image and Vision Computing, 20 (2002) 751–759.

[4] H. Wang, M. Brady, Real-time corner detection algorithm for motion estimation, Image and Vision Computing, 13 (1995) 695-703.

[5] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, B. Girod, Outdoors augmented reality on mobile phone using loxel-based visual feature organization, in: Proceeding of the ACM international conference on Multimedia information retrieval, 2008, pp. 427-434.

[6] M. Brown, D. Lowe, Automatic Panoramic Image Stitching using Invariant Features, International Journal of Computer Vision, 74 (2007) 59-73.

[7] M. Brown, D. G. Lowe, Recognising Panoramas, in: Proceedings of the 9th International Conference on Computer Vision (ICCV2003), 2003, pp. 1218-1225.

[8] iPhone App: Autostitch, http://www.cloudburstresearch.com/autostitch/autostitch.html.

[9] S. Avidan, A. Shamir, Seam Carving for Content-Aware Image Resizing, in: Proceedings of SIGGRAPH 2007, 10, 20073.

[10] M. Rubinstein, A. Shamir, S. Avidan, Improved Seam Carving for Video Retargeting, in: Proceedings of SIGGRAPH 2008, 16, 2008.

[11] iPhone App: Liquidscale, http://www.savoysoftware.com/liquidscale/.

[12] CUDA, http://developer.nvidia.com/object/cuda.html.

[13] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, W. W. Hwu, Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, in Proceedings of PPOPP 2008, 2008, pp. 73-82.

[14] Y. Sun, X. Sun, H. Zhang, Research on parallel cone-beam CT image reconstruction on CUDA-Enabled GPU, in: Proceedings of IEEE International Conference on Image Processing, 2010, pp. 4501-4504.

[15] E. Rosten, T. Drummond, Machine learning for high-speed corner detection, in: Proceedings of ECCV'06, 2006, pp. 430-443.

[16] GPU architecture, http://blog.goldenhelix.com/?p=374.

[17] P. Bui, J. Brockman, Performance analysis of accelerated image registration using GPGPU, in: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, 2009, pp. 38-45.

[18] Khronos Group, OpenGL ES 2.0 Specification, http://www.khronos.org/opengles.

[19] The testing program is available on our website: http://mail.tku.edu.tw/chchou/others/others.htm.