PERFORMANCE STUDY OF PARALLEL IMPLEMENTATION OF TEXTURE IMAGES USING GLCM

SALAMIAH BINTI ESA

A project report submitted in partial fulfilment of the
requirements for the award of the degree of
Master of Engineering (Electrical - Electronics & Telecommunications)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

JUNE 2013

This thesis dedicated to my beloved family, and all who have helped me along the way.

ACKNOWLEDGEMENT

I would like to offer my appreciation and gratitude to the following:

Allah Subhanahu Wa Ta'ala for the many blessings that I have received in my life and helping me to overcome the challenges in this project.

To my supervisor, Dr Usman Ullah Sheikh for his patient guidance, understanding and advice and who helped me clearly focus my objectives in pursuing this project.

To Mr. David Appleton who helped me understand the basic of programming and image processing.

ABSTRACT

The process of the creation of texture images derived from a windowed GLCM coupled with the calculation of Haralick features for each window is a time intensive process due to the intense number of calculations involved. This study examines and seeks to quantify the expected increase in processing speed when migrating this algorithm from a traditional serial implementation to a parallel implementation of the same function using modern Graphical Processing Units and the effects of certain parameters such as window size and image size. The components of texture images and some of the factors relating to the efficiency of CUDA code are described. The problem domain was analysed and a serial version of texture window analysis was implemented and checked for accuracy by comparing it to code written in Matlab. The serial code was tested on a 2.4 Ghz Intel core i5 processor while the parallel code was tested on two different GPU cards, a GeForce 310M and a GeForce GTX 620. The final (fastest) implementation used three kernels. Two of these performed gray scale conversion and intensity scaling while the third performed the entire GLCM construction and feature extraction. The results showed that a single large kernel could outperform the collection of small kernels that was used in the alternative implementation. As a result of parallel implementation, texture analysis of a 2048 x 2048 pixels image was found to be up to 44 times faster than the serial version using the GeForce 310M and even faster on the GeForce GTX 620.

ABSTRAK

Proses membentuk imej-imej tektur dari tetingkap GLCM berserta pengiraan ciri-ciri Haralick adalah satu proses yang lama kerana ia melibatkan banyak pengiraan. Dalam penyelidikan ini, pengukuran kepantasan dalam pemprosesan tersebut dikaji apabila algoritma tradisional sesiri dilaksanakan secara selari menggunakan Unit Pemprosesan Grafik yang moden. Peningkatan kepantasan apabila parameter seperti penambahan saiz tetingkap dan saiz imej juga diselidik. Kajian ini juga menjelaskan tentang komponen imej-imej tektur dan faktor yang berkaitan dengan penggunaan kod CUDA secara optimum. Masalah dalam kajian ini dianalisis termasuk perlaksanaan analisis tersebut terhadap tingkap-tingkap tektur versi sesiri. Ketepatan hasil kerja diperiksa dengan membandingkannya dengan kod yang ditulis menggunakan Matlab. Kod sesiri diuji menggunakan pemproses Intel Core i5, 2.4 GHz. Kod selari diuji menggunakan GPU yang berbeza, GeForce 310M dan Geforce GTX 620. Kajian menggunakan tiga kernel untuk melaksanakan pemprosesan membentuk imej-imej tektur dan pengiraan ciri-ciri Haralick didapati paling pantas. Dua kernel yang pertama memproses penukaran imej berwarna kepada imej hitam-putih/kelabu dan melaksanakan pengskalaan imej kelabu tersebut. Kernel ketiga memproses penghasilan GLCM bagi setiap imej pixel dan melakukan pengiraan untuk mendapatkan ciri-ciri Haralik. Keputusan menunjukkan bahawa penyelesaian menggunakan satu kernel yang menyelesaikan masalah yang banyak adalah lebih baik daripada penyelesaian menggunakan banyak kernel yang menyelesaikan masalah yang kecil. Perlaksanaan selari menggunakan GeForce 310M bagi 2048x2048 imej pixel adalah 44 kali lebih pantas berbanding perlaksanaan secara sesiri dan keputusan ujian menggunakan GeForce GTX 620 adalah jauh lebih pantas.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ABREVIATIONS

ABS	-	Air-lock breaking system
ALU	-	Arithmetic logic unit
ASM	-	Angular second moment
CAD	-	Computer aided design
CPU	-	Centre processing unit
CUDA	-	Compute unified device architecture
1D	-	One dimensional
2D	-	Two dimensional
3D	-	Three dimensional
GLCM	-	Grey level co-occurrence matrix
GPU	-	Graphics processing unit
GPGPU	-	General-purpose graphics processing unit
IBM	-	International business machine
IEEE	-	Institute of Electrical and Electronic Engineers
MIMD	-	Multiple input multiple data
OpenCL	-	Open computing language
OpenGL	-	Open graphics language
PC	-	Personal computer
PDF	-	Probability density function
PPM	-	Portable pixel map
PPE	-	Power processor element
RGBA	-	Red green blue alpha

SIMD	-	Single instruction multiple data
SIMT	-	Single instruction multiple threads
SPE	-	Synergistic processing element
V2	-	Version two

LIST OF SYMBOLS

d	-	Distance
i	-	Row
j	-	Column
p	-	Probability
σ	-	Variance
	-	Mean
V	-	Count of occurrence

LIST OF APPENDICES

CHAPTER 1

PROJECT OVERVIEW

1.1 Introduction

The history of computer graphics goes back to the 1960s. After the IBM PC was introduced in 1981 the world had a standard platform for computing and the PC game industry was born. Graphic processing was initially performed by the CPU which also had to handle everything else until dedicated graphic accelerators started being built which offloaded some of these tasks from the CPU. The formulation of OpenGL in 1989 was a step towards the standardization of graphics implementation, along with the definition of the concept of a graphics pipeline.

The graphics pipeline could make 3D graphic games look far more realistic and responsive. Texturing and pixel shading are some of the elements in the graphics pipeline that contributed to the success of 3D graphic games like Quake and Doom. In 1999 fast interactive 3D graphics had became a full reality when billion of transistors were built into the hardware of the first Graphics Processing Unit (GPU) by NVidia which embodied all the stages of the graphics pipeline. Today, the GPU is widely used not only in personal computers and workstations but also in embedded systems, mobile phones and game consoles.

Modern GPUs are extremely efficient for graphics computation compared to a general-purpose CPU. The CPU has a single threaded architecture that allows multiple processes to be run through the same single-threaded pipeline which access their specific data through a single memory interface. In contrast, the GPU

architecture uses stream processing from (up to) thousands of stream processors running in parallel with multiple memory models. The GPU is designed to handle large streams of graphic data and is able to process the computations much faster because the processing is done concurrently. As the requirements for graphical processing grew, the GPU evolved into a device able to run generic computation of a non graphical equally well as long as the problem required large amounts of data and could be mapped to the stream concept.

The benefit of using GPUs to process images is not a new issue. Work has been done to examine the performance of GPUs and research related to image processing and GPU performance dates back to early 2000. Early papers included Strzodka *et. al.*[1] implementing a motion estimation algorithm that provides dense estimates from optical flow where they get a 2.8 times speed increase on a GeForce 5800 Ultra GPU, while Hadwiger *et al.*[2] presented a review of GPU image segmentation in medical imaging application for data visualization.

Not all parallelizable applications have been able to benefit from the advantage of the parallelized architecture of GPU. In order to benefit significantly to this parallelism the data has to be independent, aligned, and have regular data access.

In this project we investigate and study the creation of Gray Level Co-Occurrence Matrices (GLCM) on image of a variety of sizes. The GLCM has been widely used to extract texture features from images. Texture feature extraction and comparison is used in applications as diverse as remote sensing, medical image, recognition of materials such as wood and granite, finger print analysis, depth analysis and many more. The GLCM is analysed by extracting the Haralick texture feature set in order to make all these applications a reality. Finally, doing this using a sliding window creates “texture images” which is a method of image segmentation based on texture.

Haralick’s texture features are statistical features that reduce GLCM data to a few statistical data values that give information about the area being analysed. These

will be explained in chapter 2. Computation of the GLCM and extracting the Haralick texture feature set is very heavily computationally intensive, thus time consuming. Doing these calculations repeatedly on a sliding window across an image magnifies the level of computation incredibly with each pixel being processed a large number of times. In this report these computations will be implemented both in the traditional serial method, and in parallel, on an Acer Aspire notebook using an NVidia GeForce 310M graphical processor and a 2.4GHz Intel Core i5-450M processor.

1.2 Problem Statement

- The data in GLCMs forms a sparse array which leads to an irregular data access pattern. This may make it harder to get an efficient parallel implementation of algorithms to be processed by a GPU. The identification of algorithms to make parallel, and how, is necessary in order to exploit the benefit of having a GPU.

- In parallel implementations, there is an overhead associated with setting up and managing parallel programming units. If there is only small amount of work to perform the overhead can outweigh the performance benefit. Another issue to be considered is coordinating the data which is required to when the workload is shared between the CPU and GPU or if there is a need to work in a concerted manner. The more coordination that is required the poorer the performance of parallel programming. Thus the decision of such algorithm must consider the overhead and coordination of data in order to achieve better performance.

- Decision on which algorithms to make parallel and which should stay sequential. Some problems are excellent targets to be parallelized but others are better to be performed sequentially. A well planned mix of parallel and sequential code offers the possibility of getting optimum performance. In

order to make informed decisions, measurement tools are needed to help in deciding which approach gives the best performance gain.

- GLCMs are two dimensional histograms where the bins are accessed by the intensity of each pixel versus the intensity of a given neighbour. Traditionally, because of the limitation of computing power, the distance is usually limited to a single value, d=1 which is computed in horizontal and/or vertical directions. Computation for a range of distance such as 2, 3 and more could possibly capture new information and have significant impact.

- For a texture image there are many GLCMs to be computed and features extracted. Computation is very intensive as many matrices need to be computed as well as the additional feature calculations. The computation intensity will increase with the square of the image size (side). The computation of the Haralick statistical feature set is equally time consuming.

1.3 Objectives

The objective of this study is described as follows:-

- To evaluate the effect of migrating computationally complex algorithms i.e. GLCMs and feature extraction to form texture images from a traditional architecture to parallel implementation using GPU.

- To compute GLCM and Haralick's texture features to create texture images for any given image size with varying windows sizes.

1.4 Scope of the study

The following is the scope of this study:-

- To understand GLCM, and Haralick's texture features and texture images and implement it using Processing for an initial proof of concept of serial computation.

- Program the implementation for varying parameters such as distance and window size in Processing to examine the effect of changing these features.

- Study parallel programming in CUDA. Implement texture images from GLCM and Haralick's feature extraction in both CUDA and serial code.

- To study how to convert the serial implementation to parallel and optimization on the parallel programming. At the same time convert the Processing code to Visual C++ to allow fair comparison with the CUDA version since Processing is Java based and so does not produce native code.

1.5 Thesis Organization

This thesis is organized into 5 chapters.

Chapter one (this chapter) provides the introduction of GPU, discussion of problem statement, project objectives, scope of study and thesis organization.

Chapter two presents a literature review of GLCMs covering prior work that is related to this project, an explanation of GPU architecture and an understanding CUDA.

Chapter three describes the project methodology and provides a full discussion about the flow of the work. This includes creating GLCMs and Haralick's feature computation and an introduction to the CUDA programming model.

Chapter four evaluates the results of the work. The accuracy of the algorithms for extracting the GLCMs and Haralick features, test results and the performance gain that was obtained from moving from serial to parallel. In this chapter the performance gains will be compared based on image size will be presented followed by a discussion of any problem that will be encountered.

The last chapter is the conclusion of all chapters, and provides explanation for future works and recommendations for future study.

1.6 Summary

Since parallel computing has been proposed as the way forward for intensive computing, this project will attempt to quantify the benefit that will be gained from using parallel processing. Making use of the benefit of parallel programming by using graphics processors is expected to enable us to compute these algorithms faster especially when the image size is huge. That being the case, we hope to be able to identify some real life applications that this work may make possible by bringing such computations into real time.

REFERENCES

[1] R. Strzodka and C. Garbe, “Real-time motion estimation and visualization on graphics cards,” in *ProceedingsIEEE Visualization 2004*, 2004, pp. 545–552.

[2] M. Hadwiger, WK Jeong, H. Pfister and J. Beyer Strzodka, “GPU-Accelerated brain connectivity reconstruction and visualization in large scale electron micrograph,” *GPU Computing Gems*, Emerald Edition 2011, pp. 793–812.

[3] Wilson, Gregory V (1994). *"The History of the Development of Parallel Computing"*. Virginia Tech/Norfolk State University, Interactive Learning with a Digital Library in Computer Science. *Retrieved 2008-01-08.*

[4] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. 3rd edition, 2002. Morgan Kaufmann, *ISBN 1-55860-724-2*. Page 43.

[5] J. M. Rabaey. *Digital Integrated Circuits.* Prentice Hall, *1996*.

[6] A. Downton and D.Crookes, *Electronics & Communication Engineering Journal*, June 1998.

[7] David Luebke, GPU Architecture: Implication and Trends, *Nvidia Research*, *Siggraph* 2008.

[8] Timothy G. Mattson, Beverly A. Sanders and Berna L.Massingill, *Pattern for parallel programming*, 2005, pg 9-10, *Addison-Wesley*.

[9] CUDA C Programming Guide, pg 153, Nvidia, www.nvidia.com .

[10] CUDA C Programming Guide, pg 76-77, Nvidia, www.nvidia.com .

[11] Guochun Shi, Volodymyr Kindratenko, Rob Kooper and Peter Bajcsy. *GPU Acceleration of an Image Characterization Algorithm for Document Similarity Analysis*. 2011 IEEE

[12] Ashwin M, Aji, Liqing and Wu-Chun, GPU-RMAP : *Accelerating Short-Read Mapping on Graphics Processor*, 2010, IEEE.

[13] Cliff Woolley, *CUDA Overview*, pg 11, Nvidia Developer Technology Group, Nvidia.

[14] Nan Zhang, Jian-li Wang and Yun-shan Chen. *Image Parallel Processing Based on GPU*. 2010 IEEE

[15] Bi-Hui Wang,Hang-Jun Wang, Heng-nian Qi, *Wood recognition based on GLCM*, 2010, International conference on computer application and system modelling.

[16] Mari Partio, Bogdan Cramariuc, Moncef Gabbouj, and Ari Visa*, "Rock Texture Retrieval using gray level Co-occurrence Matrix*" Tempere University of Technology

[17] Amjad Ali, Xiaojun JNing, Nasir Saleem, *GLCM-based fingerprint recognition algorithm*, 2011, IEEE.

[18] Markus Gipp, Guillermo Marcus, Nathalie Harder, Apichat Suratanee, Karl Rohr, Rainer Konig, Reinhard Manner, *Haralick's texture Features Computed by GPUs for Biological Application*, IAENG International Journal of Computer Science, 2008

[19] Dr. N.P.Rath, Prasanjeeta, Janyadatta, *Depth Analysis of Monocular Natural Scenes using GLCM*, 2012, 4th International conference on intelligent and advanced system.

[20] Asadollah Shahbahrami, Tuan Anh Pham, Koen Bertels, *Parallel implementation of Gray Level Co-occurrence Matrices and Haralick texture features on cell Architecture*, 2011, Springer Science + Business Media.

[21] Yong Hu, Chun-xia Zhao, Hong-nan Wang, *Directional analysis of texture Image using Gray level co-occurrence Matrix*, 2008, IEEE.

[22] Jing Yi Tau, Yong Hour Tay, Phoii Yee Lau, *One-dimensional GLCM for texture classification*, 2008, IEEE.

[23] M. Abolghasemi, H. Aghainia, K.Faez, M.A. Mehrabi, *LSB Data Hiding Detection Based on Gray Level Co-Occurrence Matrix (GLCM)*, 2008, International Symposium on Telecommunications.

[24] http://www.fp.ucalgary.ca/mhallbey/contrast.htm

[25] Yu Jian, *Texture image segmentation based on Gaussian Mixture Models and GLCM*, 2010. IEEE.

[26] Cuda_OpenCV GPU architecture.pdf

[27] http://www.online-utility.org/image_converter.jsp?outputType=PPM

[28] Hillis, W. Daniel, and Guy L. Steele, Jr. 1986. "*Data Parallel Algorithms*." Communications of the *ACM* 29 (12)

[29] http://www.cse.nd.edu/courses/cse60881/www/lectures/logsum.pdf

[30] Introduction to Parallel Programming (CS344) http://www.udacity.com/course/cs344