Canny Edge Detection Performance Comparison Between Parallel Programming and Serial Programming

Zhehui. Chen
Mechanical Engineering
National Chaio Tung University
Hsinchu
zhehui8805@gmail.com

Eason, Huang
Robotic
National Chaio Tung University
Hsinchu
eason27271563@gmail.com

Tsung-Han Liu
Mechanical Engineering
National Chaio Tung University
Hsinchu
steven.me05@nctu.edu.tw

The participant(s)

EasonHuang : Serial 程式撰寫

Tsung-Han Liu：OpenMP、CUDA、OpenCL 程式撰寫、架構統整

Zhehui. Chen : Pthread 程式撰寫

ABSTRACT

在工程上進行電腦視覺的分析，影像處理時，時常會使用到一個邊緣檢測的算法"Canny Edge detection"，然而一般使用 Serial 的程式架構下只有 4.5 Hz 的更新頻率，無法應付高畫質的影像處理需求，因此我們在此次的期末專題中嘗試使用各種平行化方式進行程式的加速。

Introduction

邊緣探測是許多影像處裡的第一步，他用來識別圖像中尖銳的不連續性，例如亮度或是密度的改變，這項技術已被廣泛的研究，因此有許多不同的演算法，例如 Robert detector, Prewitt detector, Canny detector 等，而在這些演算法中，Canny 演算法是其中最被廣泛使用的，因為他對於高斯分布的雜訊有一定程度的抵抗能力，並找出最佳的邊緣，且擁有失敗率低的特色。

然而 Canny 演算法在實作時要經過 1. 預處理 (轉灰階、高斯濾波) 2.取得每個 pixel 的梯度值與梯度方向 3. 利用非極大值抑制尋找可能的邊緣 ，因此在實務上這是個相當耗時的計算過程。為了達到工程上 real-time 的運算需求 (20hz)， 我們試圖以平行程式的技巧對 Canny 演算法進行加速。

我們的加速方式有兩個切入點，一個是 CPU 加速另一個則是 GPU 的加速，CPU 擅長於邏輯運算而 GPU 利用核心眾多的機制則可以一次處理大量的獨立資料。

CPU 的加速我們的實作方式是使用 Pthread 跟 OpenMP，他們會使程式產生多個執行序，讓電腦進行排程，若電腦中有空閒的 CPU 核心可以計算，則可以達到多執行序的加速。

GPU 的部分則是使用 CUDA 和 OpenCL，將資料傳送到 GPU 的記憶體上進行計算，GPU 上所有的核心數是 CPU 的好幾倍，因此對於圖像這類資料型態的計算，可以達到較好的優化效果，在其他論文[2]中有看到 1024x1024 的圖像可達到 100hz 的執行速度。

Problem Definition

Canny 實作內容實際上可以拆解成 5 個步驟:

1. Gaussian Filter

   將照片對一個高斯分布的 kernel 做 convolution ，藉此模糊照片中原本的躁點。其中最花時間的部分是做 convolution 的計算，且因為 convolution 運算時資料間沒有相依姓，所以我們可以運用平行化的方法降低運算時間。

2. Sobel Filter

   將照片分別對 X 和 Y 方向的 sobel kernel 做 convolution ，藉此找出原照片中各像速點的強度變化方向和大小。其中最花時間的部分是做 convolution 的計算，且因為 convolution 運算時資料間沒有相依姓，所以我們可以運用平行化的方法降低運算時間。

3. Non-maximum Suppression

   藉由以上兩個方式可以大概找出邊界的垂直向量，但由此找出的邊界仍具有一定的厚度，與我們希望單純找到一條細的線以表示邊界的期望不符。因此透過第二步得到的資訊，將沿著向量方向的為大值保留且其餘部分設為 0 的方式，濾除不需要的邊界厚度，以達到我們希望的單存的邊界的期望。該部分的運算因為資料間也沒有相依性，所以也可以用平行化的方式降低運算的時間。

4. Double Threshold

藉由設定 threshold 表示該條線是否為我們認為的邊界，threshold 愈大對該條線是否為邊界的判斷越嚴格，反之亦然。如果說變化強度值大於 upper threshold 則該 pixel 視為邊界點，小於 lower threshold 則視為非邊界點，若值位於 upper threshold 和 lower threshold 之間，則該 pixel 留至下一步驟做進一步的判斷。這部分的運算資料間都沒有相依性，所以可以使用平行化的方是做運算的加速。

5. Hysteresis

   對於判斷強度變化值介於 upper threshold 和 lower threshold 之間的 pixel 是否為邊界點的方式是，如果該 pixel 附近有兩個 pixel 被保留成邊界點的話，該 pixel 也可以被視為邊界點，反之，該 pixel 就不是邊界點。但由於這部分的運算資料間具有相依性，所以這部分的運算不能使用平行化的方式降低運算的時間。

Proposed Solution

Pthread

設計 Pthread 的平行化我們是將一張圖由 height 去將圖形切成條狀，並分別由 CPU 使用不同核心去做同步的運算，經由以上該念為出發，考慮到不能因為將圖形切塊，而影響邊緣(edge)的連續性， 設計程式時在圖案上下各預留了一定大小的裕度，雖然這麼做會增加一些執行時間，但是對於邊緣(edge)的完整性有很大程度的改善。

OpenMP

OpenMP 的實作我們是將所有使用到 for 迴圈且不具有資料相依性的地方平行化，借助不同核心的 CPU 加速 for 迴圈的運算速度。

CUDA

CUDA 的實作我們進行了兩個部份的平行化，首先第一個部分是 convolution 運算的平行化，Canny edge detector 有許多 convolution 的運算，而這些運算是最耗時的，因此本次實作最主要就是將這些 convolution 的計算改以 GPU 平行運算的方式，用 GPU 的 thread 去計算與儲存各個 pixel 的運算結果，第二個是我們發現在梯度值與方向計算時，各個 pixel 的運算也是獨立的，因此也對其做了平行化運算的處理。另外，為了要更加快運算的速度，我們忽略了第 5 個步驟中資料間的相依性，加快速度的同時檢查輸出的結果是否和之前有區別。結果發現以肉眼來判斷的話，圖形差異不大，但加速時間降低了約 1 倍。

OpenCL

OpenCL 的實作我們進行了兩個部份的平行化，首先第一個部分是 convolution 運算的平行化，Canny edge detector 有許多 convolution 的運算，而這些運算是最耗時的，因此本次實作最主要就是將這些 convolution 的計算改以 GPU 平行運算的方式，用 GPU 的 thread 去計算與儲存各個 pixel 的運算結果，第二個是我們發現在梯度值與方向計算時，各個 pixel 的運算也是獨立的，因此也對其做了平行化運算的處理。另外，為了要更加快運算的速度，我們忽略了第 5 個步驟中資料間的相依性，加快速度的同時檢查輸出的結果是否和之前有區別。結果發現以肉眼來判斷的話，圖形差異不大，但加速時間降低了約 1 倍。

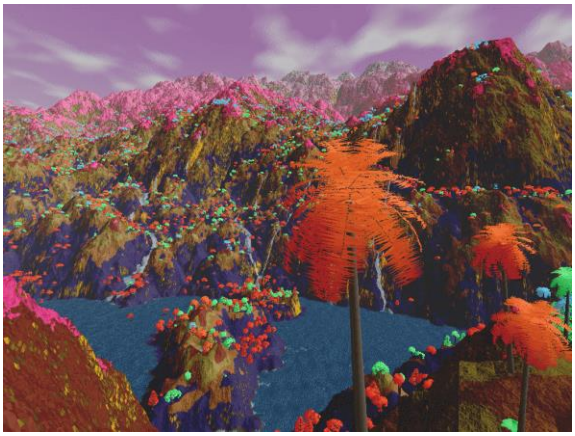
Experimental Methodology

本次實作的實驗方式是求取 Canny 運算時所耗費的時間，總共會測試十次找中間的 4 次來取平均。並將各個方式的運算結果與運算速度去做分析。

另外我們分別對大圖片 (1024x768) 做運算和小圖片 (600x400) 做運算，並比較各個 implementation 的差異。

Experimental Result

Original image (1024x768):

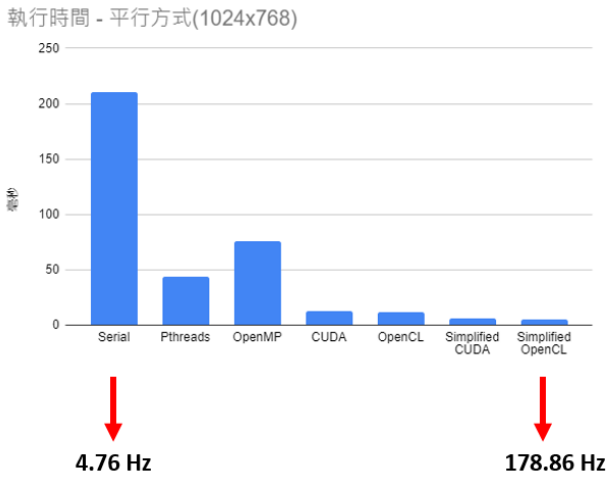


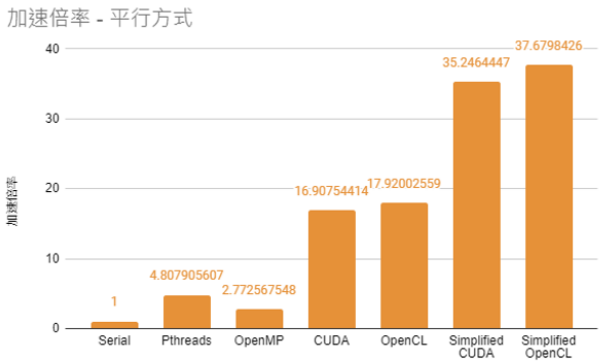
Original image (600x400):

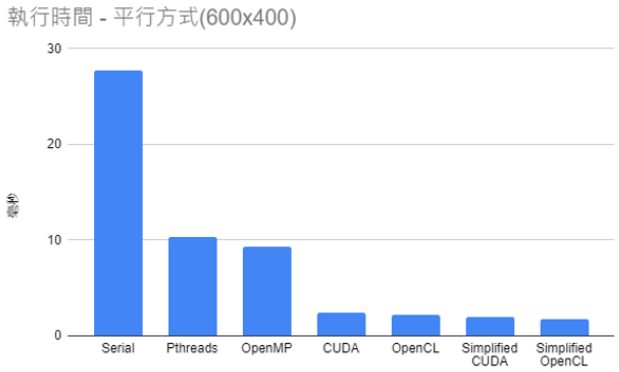
Image after Canny Edge Detection (1024x768):

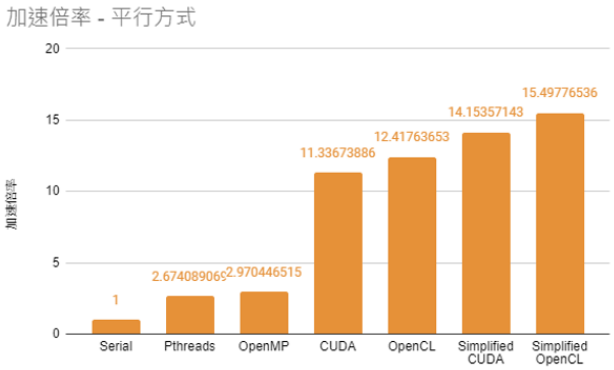
Image after Canny Edge Detection (600x400):

加速結果比較圖









Related Works

在論文[2]中可以看到 CUDA 與 CPU 運算的執行時間如下

- CPU:Intel Core2 CPU 6600 @ 2.40 GHz, 2 GB RAM
- GPU: NVIDIA GeForce 8800 GTX, 768 MB global memory (not overclocked)

Image Size	CUDA (ms)	OpenCV	Speedup
256× 256	1.82	3.06	1.681
512× 512	3.40	8.28	2.435294
1024× 1024	10.92	28.92	2.648352
2048× 2048	31.46	105.55	3.353465
3936× 3936	96.62	374.36	3.874560

Conclusions

經由此次的實作我們發現，在 1024 x 768 的大尺寸圖可以看到 Pthread 的執行時間比 OpenMP 還要短，這是因為 Pthread 是實質意義上將工作分成若干等份，而 OpenMP 只有將有 For 迴圈的部分做分工，其餘部分皆為 serial 執行。而 CUDA 以及 OpenCL 因為將大部分的運算都移至 GPU 平行運算，因此相比單純使用 CPU 的版本必然快上好幾倍。

至於在 600 x 400 的小尺寸實驗圖，我們可以發現 Pthread 的執行時間比 OpenMP 還要慢，與上述實驗結果不同，原因是因為 Pthread 版本的實作因為要考慮圖片連續性，我們在工作切割區都加上了一些裕度，因此增加了計算量，所造成的 overhead 已經大過於實質平分工作的效率，因此 OpenMP 的執行速率較快。

最後可以看到簡化版的 CUDA 以及 OpenCL 在速度上都有明顯的加速，而所造成的影響在工程上也沒有明顯差異，因此利用我們簡化版的 OpenCL 可以達到 178.86 Hz，而上述的結果也遠超過於我們所需的工程需求。

REFERENCES

[1] J.F. Canny, "A computation approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no 6, pp. 769-798, November 1986

[2] Y. Luo and R. Duraiswami, "Canny Edge Detection on NVIDIA CUDA," Proc. Of IEEE Computer Vision and Pattern Recognition Workshops, 2008 , pp. 1-8