Accelerate Canny Edge Detector with Parallel Programming

Zi Yi Huang
312551072
IOC, NYCU
Hsinchu, Taiwan
m312551072.cs12@nycu.edu.tw

Bo Han Chen
312551074
IOC, NYCU
Hsinchu, Taiwan
bhchen312551074.cs12@nycu.edu.tw

Hsuan Yu Chi
312551080
IOC, NYCU
Hsinchu, Taiwan
chihy.cs12@nycu.edu.tw

1 INTRODUCTION

Canny Edge Detection (CED) 是一種常見的影像處理演算法，用於偵測圖片中的大範圍邊界。它是在 1986 年由 John F. Canny 發明。Canny Edge Detection 可以有效地去除影像中的雜訊、模糊的邊緣與非邊緣的線，產生更高 SNR 的影像，使結果更為精細。以下是 Canny Edge Detection 常見的應用領域 (1) Object detection : 偵測影像中物體的邊緣，用於辨識物體。如用於偵測道路交通中的汽車、行人和其他物體。(2) Image segmentation : 將影像分割成不同的圖像子區域，用於進一步分析。實際可應用於醫學影像、指紋識別、人臉識別等 (3) Feature extraction : 從影像中提取特徵，提取物體的邊緣或建築物的角落。這些功能隨後可用於物件辨識。如從臉部影像提取特徵進行人臉識別，或從街道影像中提取特徵用於自動駕駛。結合以上應用，因此 Canny Edge Detection 在醫學圖像分析、機器人學、電腦視覺、工業檢驗相關研究領域廣泛使用。

Canny Edge Detection 的優點可以分為以下幾點

(1) Accurate edge localization : 可以精準的定位圖片中的邊緣，確保僅保留重要的邊緣
(2) Low error rate : 演算法中 double thresholding 與 edge tracking 的機制降低誤判邊緣的可能性，因此有較低錯誤率。
(3) Single response to edges : 圖片中每條邊緣在輸出僅由 single response 表示，避免重複邊緣偵測，使線條更為細緻。
(4) Robust to noise : 適用於受到各種噪聲影響的真實世界圖片影像。起始步驟中的 Smoothing 使 Canny Edge Detection 具有很高的抗噪性。

缺點則是

(1) Computationally expensive : Canny Edge Detection 的計算成本很高，尤其圖片解析度很高時。
(2) Sensitive to parameters : double thresholding 的選擇會很大程度影響邊緣偵測的輸出結果，對於不同圖片會有適合的 threshold。

Canny Edge Detection 的計算成本相當高，包含多個卷積、Gradient Computation、Nonmaxima Suppression、BFS 搜尋。雖然複雜，但計算間有獨立性，可使用平行處理來加速。透過將影像劃分為多個區域，使用 thread 平行處理每個區域，在多核心處理器上加速；或使用 GPU 矩陣乘法來加速卷積運算。藉由平行運算，可以在不影響正確性的情形下提升計算效率與 throughput，滿足需要處理大量資料的需求。

2 STATEMENT OF PROBLEM

在這個部分，我們會介紹 Canny Edge Detection 演算法，描述演算法中各步驟的運作，最後分析哪些部分是可以使用平行化來加速的。Canny Edge Detection 演算法分成以下幾個步驟：Smoothing、Gradient Computation、Nonmaxima Suppression、Double Thresholding 和 Hysteresis Edge Linking，接下來我們會針對每個步驟做詳細的說明。

2.1 Smoothing

對於 Smoothing，我們使用一個 3×3 的近似高斯濾波器 G 來對輸入的影像進行卷積，以達到平滑影像的效果。

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

令 $f(x, y)$ 為輸入影像，則經過 Smoothing 後的影像 $f_s(x, y)$ 可以表示為 f 與 G 的卷積，如下算式所示：

$$f_s(x, y) = f(x, y) * G(x, y)$$

2.2 Gradient Computation

對於 Gradient Computation，我們需要計算上一步中得到的 $f_s(x, y)$ 上每個點的梯度大小與方向，我們使用 Sobel filter 來計算梯度，Sobel filter 有兩種，分別是 S_x 與 S_y，分別代表用來計算水平與垂直方向的梯度的 filter，如下所示：

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

水平方向及垂直方向的梯度分別計算如下：

$$g_x(x, y) = f_s(x, y) * S_x(x, y)$$
$$g_y(x, y) = f_s(x, y) * S_y(x, y)$$

接著計算每個點上的梯度大小 $M(x, y)$，以及其梯度方向 $\alpha(x, y)$，其中 $M(x, y)$ 為梯度的 Euclidean vector norm，$\alpha(x, y)$ 為梯度的方向，計算如下列算式所示：

$$M(x, y) = ||\nabla f(x, y)|| = \sqrt{{g_x}^2(x, y) + {g_y}^2(x, y)}$$

$$\alpha(x, y) = \tan^{-1}(\frac{g_y(x, y)}{g_x(x, y)})$$

2.3 Nonmaxima Suppression

在計算完 $f_s(x, y)$ 上每個點的梯度大小與方向後，我們使用 Nonmaxima Suppression 對邊緣進行細化，Nonmaxima Suppression 後的圖片表示為 $f_N(x, y)$。對 $M(x, y)$ 的每個點，我們尋找一個最接近其梯度方向 $\alpha(x, y)$ 的方向 d_k (在我們的方法中我們分成 4 個方向，分別是水平、垂直、左上到右下、右上到左下)，並將 $M(x, y)$ 與兩鄰居的梯度大小 $M(x-\Delta x, y-\Delta y)$ 及 $M(x+\Delta x, y+\Delta y)$ (以下表示為 M_a, M_b) 做比較，其中 Δx 與 Δy 為 d_k 的水平與垂直方向的位移量，比較的結果有兩種情

況，如果 $M(x,y)$ 比兩鄰居都大，則 $f_N(x,y) = M(x,y)$，反之 $f_N(x,y) = 0$，如下算式所示：

$$f_N(x,y) = \begin{cases} M(x,y) & \text{if } M(x,y) \geq M_a \text{ and } M(x,y) \geq M_b \\ 0 & \text{otherwise} \end{cases}$$

$$M_a = M(x - \Delta x, y - \Delta y)$$
$$M_b = M(x + \Delta x, y + \Delta y)$$

2.4 Double Thresholding

在計算完 $f_N(x,y)$ 後，接著對 $f_N(x,y)$ 中所有點做 Double Thresholding，High threshold 與 Low threshold 的值表示為 T_h 與 T_l，我們將 $f_N(x,y)$ 中的點分成三類，分別是 strong pixel, weak pixel 和 non-relevant pixel，分別表示為為 edge、可能為 edge、不是 edge 的 pixel，分別定義如下：

- strong pixel: $M(x,y) \geq T_h$
- weak pixel: $T_h > M(x,y) \geq T_l$
- non-relevant pixel: $M(x,y) < T_l$

2.5 Hysteresis Edge Linking

在得到 $f_N(x,y)$ 中的 strong pixel 與 weak pixel 後，我們將所有 strong pixel 排入 queue，對 queue 中每個點的 8-connected component 做 BFS，將連通的 weak pixel 設為 strong pixel，並將其排入 queue，直到 queue 為空為止，最後得到的圖片即為 Canny Edge Detection 的結果。

2.6 Why we choose this problem?

這個演算法因為是應用在影像上的，且需要對圖片中每個像素做計算，需要大量的時間，但觀察演算法的這幾個步驟，可以發現許多步驟中都需要使用 for 迴圈遍歷整張圖片的所有像素，且在同一個步驟中，遍歷的過程中每個像素的計算都是獨立的，因此我們可以將這些步驟平行化，以加速演算法的執行，下一個章節我們將介紹我們計畫如何將這些步驟平行化。

3 PROPOSED APPROACHES

這個部份我們會詳細介紹如何將 CED 中的各個步驟利用 Pthread、OpenMP 和 CUDA 進行平行化，並在不影響邊緣偵測正確性的同時加速原本的 CED 演算法。

3.1 Smoothing

此步驟利用2.1所提到的濾波器對圖像進行卷積抑制原圖中的噪音；由於卷積運算是各像素獨立進行，因此可以利用 Pthread 和 CUDA 將各像素的運算交由 thread，使用 CPU 和 GPU 加速。另外我們也會使用 OpenMP 對原程式中的 for loop 進行 CPU 加速，並和 Pthread、CUDA 的結果進行比較。

3.2 Gradient Calculation

藉由 Sobel filter 計算圖像中各像素的梯度和向量角度時，因為是各像素獨立運算，所以運算之間不具有資料相依性；我們同樣利用 Pthread 和 CUDA 將不同像素運算分派給不同 thread，使用 CPU 和 GPU 加速；OpenMP 的部分也是對原程式中的 for loop 進行 CPU 加速。

3.3 Nonmaxima Suppression

為了限縮上述邊緣厚度至單一像素的大小，我們會依據各像素與其鄰居的梯度值和向量角度，判斷該像素是否為潛在的邊緣像素。由於此方法的運算是基於前一步驟的結果，因此運算之間並無資料相依性的問題。我們可以使用不同 thread 判斷不同像素是否為該向量方向的最大值，並結合 Pthread 和 CUDA，利用 CPU 和 GPU 加速完成平行化。由於此步驟在線性運算時會使用 for 迴圈進行運算，因此我們也會使用 OpenMP，利用 CPU 達到平行度。

3.4 Double Thresholding

此步驟藉由2.3所提到的兩個閾值，T_h 和 T_l，判斷 NMS 所得出的像素為 strong、weak 或 non-relevent pixel。由於此步驟為各像素間的獨立運算，因此我們一樣會使用 Pthread 和 CUDA 進行多線程的運算；迭代運算的部分可以使用 OpenMP 進行加速。

3.5 Hysteresis Edge Linking

為了尋找出最後的邊緣像素，我們先將 strong pixel 存在 queue 中，再利用 BFS 將連接 strong pixel 的 weak pixel 納入 CED 的最終邊緣結果。為了達成平行化，我們會使用 Pthread 將圖像分割給不同 thread 進行 BFS 運算，在 thread 清空各自的 queue 後，將該圖像分區的邊緣辨識結果寫回各 thread 之間共享的記憶體。不過這樣的做法會在圖像分割的邊界出現 weak pixel 沒有被標示為 strong pixel 的情形，因此需要在分割時讓各 thread 負責的區域重疊 1 像素的寬度，當全部的 thread 完成寫入後再針對重疊區域進行 BFS 運算。在使用 CUDA 進行此步驟的平行化時，我們依據 Luo 等人在 [3] 提出的方法，將圖像分割給不同的 thread block，並由 thread block 中的 thread 進行 BFS 運算；雖然同一 block 中的 thread 可以達成同步，但 thread block 之間無法進行同步，因此我們會使用論文中所提出的 multi-pass 方式重複進行數次 BFS 運算，避免相鄰區域的邊緣結果無法連結。

3.6 Parallelization Strategies

綜上所述，我們會將 CED 的平行化分為 For Loop Parallelism 和 Data Decomposition 兩部分，前四步驟為將各像素的運算交由 thread 進行加速，Hysteresis Edge Linking 的部分則是將圖像分割，讓 thread 對各分區進行運算。流程圖如圖1所示。

4 LANGUAGE SELECTION

我們的程式語言使用 C++。因為 C++ 是編譯型語言，在執行之前會被轉換為機器碼，使 C++ 較為快速和高效；且 C++ 提供控管記憶體的方式，給予一定靈活與設計空間；同時 C++ STL 提供多種 API 來支持平行化，包括 std::future、std::atomic 等。並且可使用其他框架來支持平行化，如 OpenMP、CUDA、OpenCL。

5 RELATED WORK

Gonzalez 等人在 [2] 中提及了 Canny 演算法的概念，並介紹了每個步驟中的過程並給出了一些實做細節上的選擇，比如 Smoothing 中 filter 的選擇，或是 Gradient Computation 中計算 Gradient 的方式，但沒有提及如何使用平行化加速演算法的執行，所以我們接下來將介紹其他將此演算法利用平行化加速的研究。

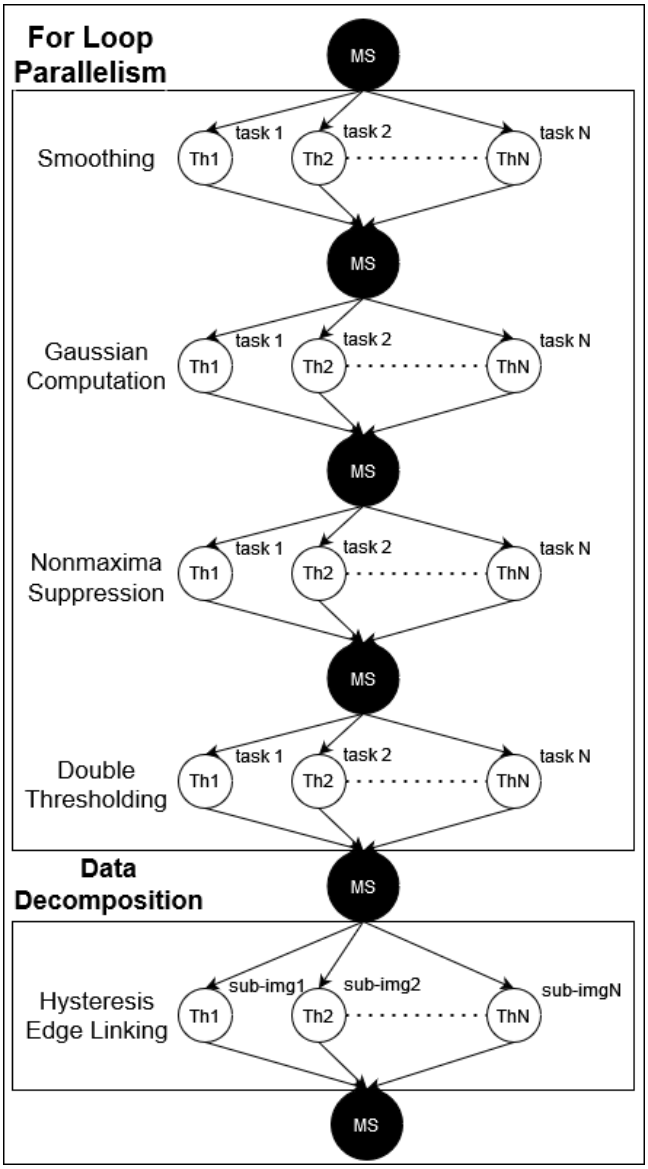

Figure 1: Parallel Strategies and Flow Chart of Canny Edge Detection Parallelism

Cheikh 等人在 [1] 中提出了兩種平行化的策略，分別為 loop-level parallelism 和 domain decomposition；loop-level parallelism 是將個步驟中可獨立運行的迴圈交由多個 thread 執行，但由於需要反覆進行 fork-join，因此會有額外的開銷，且會因為計算全域資料而失去資料局部性 (data locality)；domain decomposition 是將資料本身進行分割再交由 thread 完成各分區的各項步驟，雖然 domain decomposition 沒有前述 loop-level parallelism 的缺陷，但需要選擇適合的資料分割方式來降低平行化時的相依性來達到最佳效能。Cheikh 等人將兩種方式在多核 CPU 和 GPU 上進行實驗，發現 loop-level parallelism 在 GPU 上運行無 serial 的程式時可以達到很好的加速效果，而 domain decomposition 適合在 CPU 上針對大型資料的運算進行平行化；最後他們結合了兩種平行化策略，發現結合後的加速效果和擴展性相比於兩種平行化策略，在 CPU 和 GPU 的表現都更好，因此我們在針對 CED 的平行化策略上，也採取了結合 loop-level parallelism 和 domain decomposition 的方式，預期可以達到最好的成果。

Luo 等人在在 [3] 中嘗試以 CUDA 框架對 Canny Edge Detection 進行平行化，並且提到過往在 Canny Edge Detection 中 hysteresis labeling connected component part 的平行化處理較少，因為此步驟平行化需要 non-local memory，導致速度與效率顯著降低。論文中使用 GPU 在處理 pixel-wise operations 有較高效率的特性來加速 non-maximum suppression、hysteresis and connected components 等部分，並引入 Apron pixels 的機制來讓所有像素可以在卷積中操作。實驗結果顯示，與 CPU 版本的實作相比，隨著圖片大小而有 1.6X3.8X 不等的加速效果。另外 GPU 版本的 hysteresis 為了同步不同 thread block 間的運算使用了 multi-pass approach，導致 GPU 運算速度降低，佔據 75% 以上的運行時間。

6 STATEMENT OF EXPECTED RESULTS

針對 CED 的平行化，我們提出了數個理論並希望可以藉由實驗來觀察並驗證其正確性。

6.1 圖像大小 vs. 加速效果

我們會針對不同大小的圖像進行 CED 的平行化測試，由於 CED 的多個步驟都是針對各像素進行運算，可以推斷圖像大小等同於運算量的多寡，因此我們預期大張圖像的加速效果會優於小張圖像的加速效果。

6.2 平行化程式的擴展性

藉由不同的 thread 數量進行實驗，我們可以計算在運行多個 thread 的情況下是否可以達到預期的加速效果。因為 Hysteresis Edge Linking 步驟需要對 thread 進行同步，因此我們認為該步驟會在增加 thread 的同時增加同步所需要的開銷，從而限制加速的效果。

6.3 Pthread vs. OpenMP

雖然 Pthread 和 OpenMP 同樣使用 CPU 進行加速，但針對 thread 的行為部分，Pthread 是開發者自定義而 OpenMP 是取決於編譯器，因此我們認為 Pthread 的加速效果會優於 OpenMP。

6.4 Pthread vs. CUDA

CUDA 雖然使用 GPU 進行加速，但需要將資料在記憶體和 GPU 之間進行搬運，從而增加搬運的開銷；因此我們會利用小張圖像測試 Pthread 和 CUDA 的加速效果，觀察搬運開銷是否會在運算小張圖像時降低 CUDA 的加速效果。

6.5 Comparison of All Method

最後我們會利用 1920*1080 的圖像對不同加速方法進行測試，觀察並比較不同方法之間的加速效果。

7 TIMETABLE

下表為我們的進度排程

Action plan	Nov				Dec			
	W1	W2	W3	W4	W1	W2	W3	W4
Serial Program of CED								
Pthread and OpenMP Parallelism								
CUDA Parallelism								
Experiment & Evaluation								
Final Report and Presentation Preparation								

REFERENCES

[1] Taieb Lamine Ben Cheikh, Giovanni Beltrame, Gabriela Nicolescu, Farida Cheriet, and Sofiène Tahar. 2012. Parallelization strategies of the canny edge detector for multi-core CPUs and many-core GPUs. In *10th IEEE International NEWCAS Conference*. 49–52.

[2] R.C. Gonzalez and R.E. Woods. 2018. *Digital Image Processing*. Pearson.

[3] Yuancheng Luo and Ramani Duraiswami. 2008. Canny edge detection on NVIDIA CUDA. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 1–8.