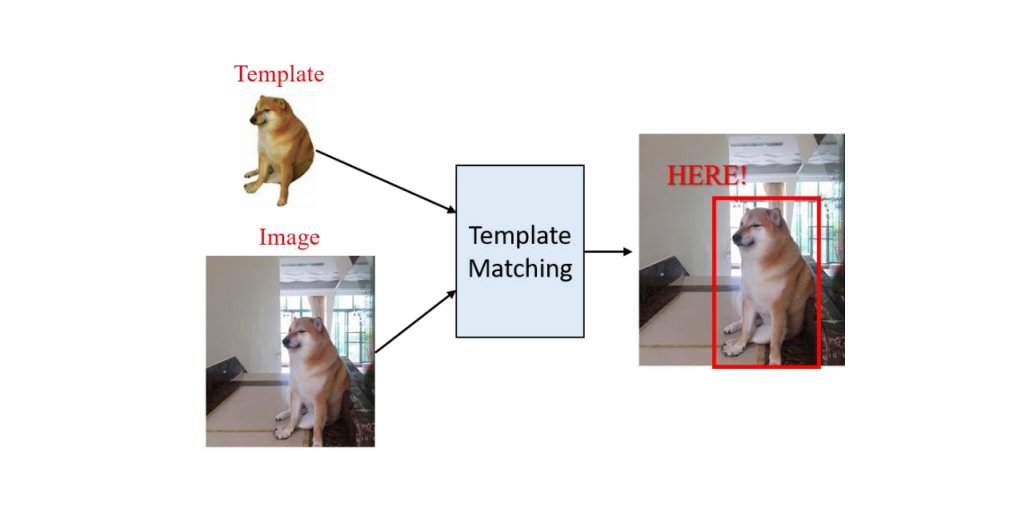
**Computer Architecture Final Project**

**312510239王則惟 312510144張理為**

**題目**

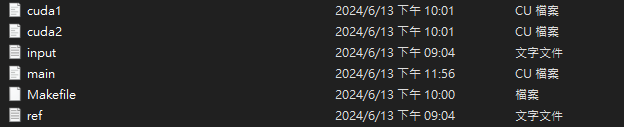
這次的期末專題，我們參考ICLAB的題目，題目是Template Matching with Image Processing，Template Matching可以被視為一種非常基本的物體檢測形式。它是一種高層次的機器視覺技術，能夠識別圖像中與預定模板匹配的部分，這意味著我們可以使用包含我們想要檢測的物體的“模板”來檢測輸入圖像中的物體。例如，計算圖像相似度以找到匹配部分的方法有很多種，比如Sum of Square Differences、Cross Correlation、Cross Coefficient等等。而這次的題目，我們分別使用8種operation，包括 Cross correlation、Max pooling、Right-Diagonal Flip 、Left-Diagonal Flip、Vertical flip、Horizontal flip、Zoom-in、Shortcut + Brightness Adjustment，並利用8個thread及16個thread來做平行化的處理。



**環境**

本次project的環境是使用助教的PPT所提供的virtual box，並匯入Linux虛擬機檔案來執行程式。

**檔案說明**

****

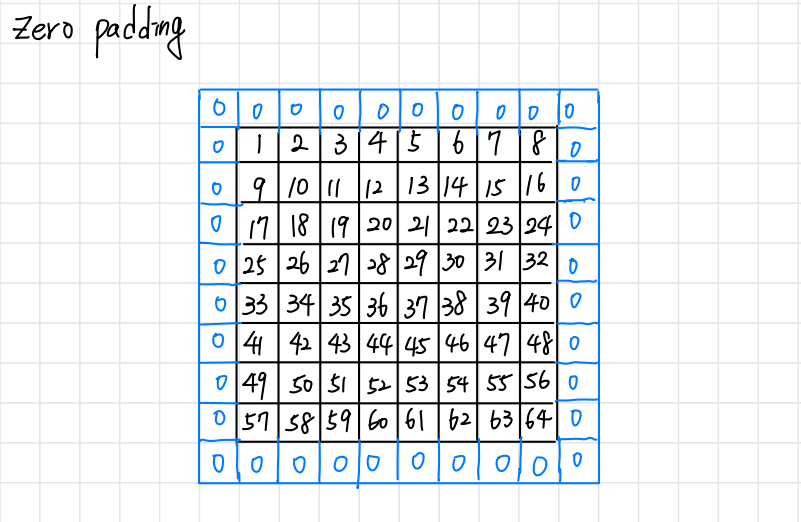
本次的project包含了6個檔案，其中main.cu會先透過吃檔案的方式讀取input.txt以及ref.txt來計算每一個operation的結果，並且呼叫cuda1.cu以及cuda2.cu中的function，然後驗證CPU計算的結果與GPU計算的結果是否相同，而cuda1.cu及cuda2.cu分別代表的是thread有8個或16個，最後再透過Makefile來compile，且執行程式時，最後會產生一個output.txt檔案，內容會是矩陣經過每種operation後的結果會是如何。

**演算法內容**

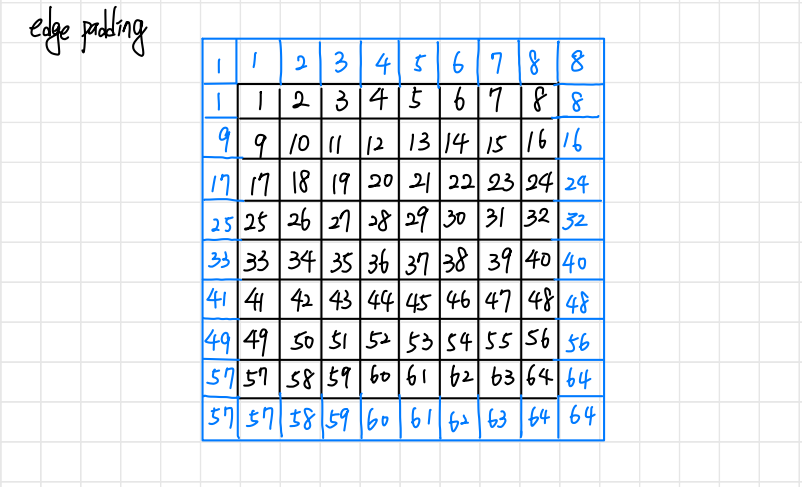
* **Operation 1 - Cross Correlation**

Cross correlation是透過先將image 做各種padding，再對ref image做convolution，而這次的題目我們分別使用了三種padding的方式，包含zero padding、edge padding，wrap padding，我們利用這三種padding的方式，將8x8的矩陣padding成一個10x10的矩陣，最後再透過對一個3x3的ref image做convolution得到我們想要的結果。

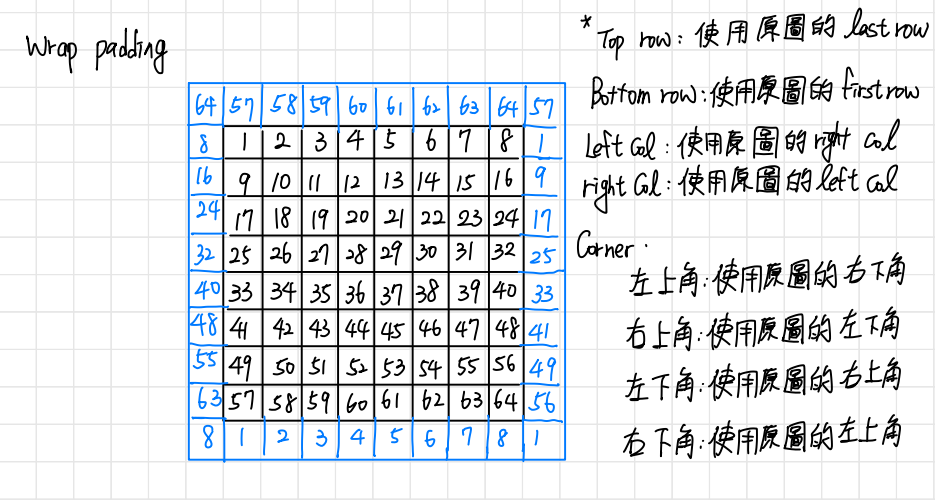
1. **Zero Padding**

****

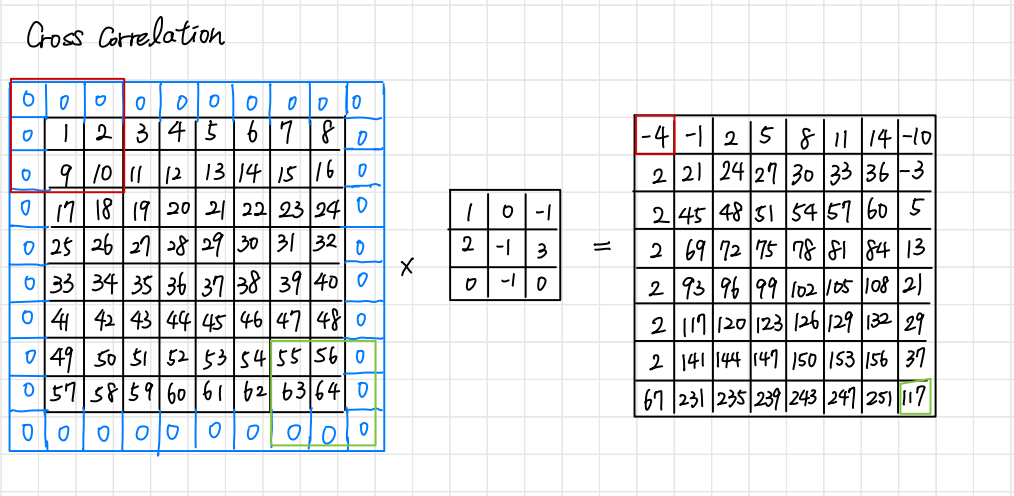
1. **Edge Padding**

****

1. **Wrap Padding**

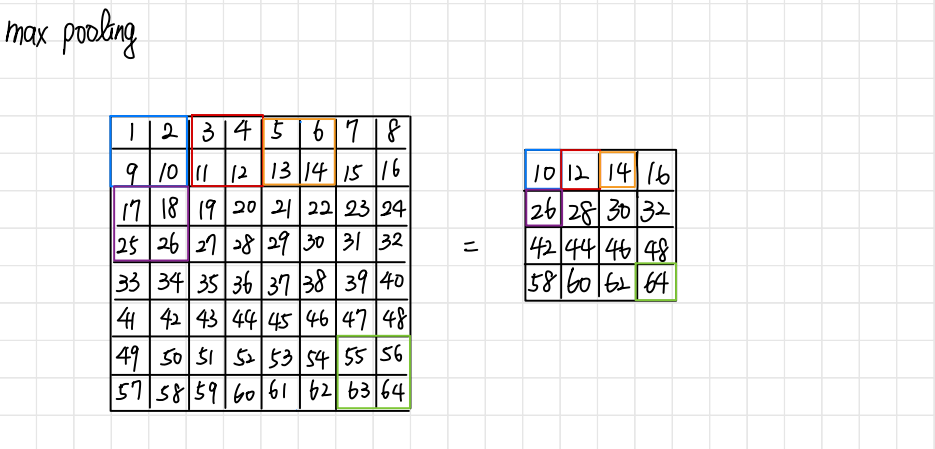
****

1. **Cross Correlation**

****

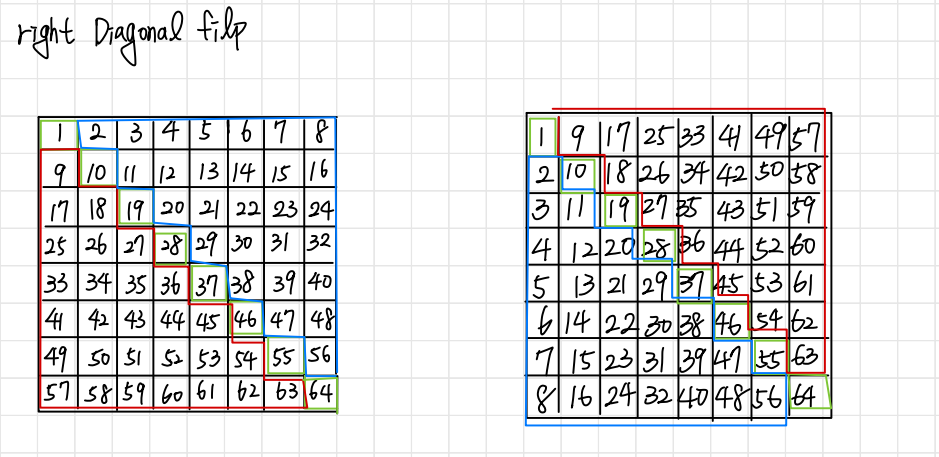
* **Operation 2 - Max Pooling**

Max Pooling是卷積神經網絡CNN中的一種下采樣（或降維）技術，用於減少特徵圖的尺寸，同時保留最重要的特徵。最大池化通常應用於卷積層之後的特徵圖，以減少計算量並防止過擬合。而這次題目我們對一個8x8的image做max pooling，結果會是一個4x4的矩陣，如下圖所示，我們會將矩陣四個一組，並取最大值來填入。



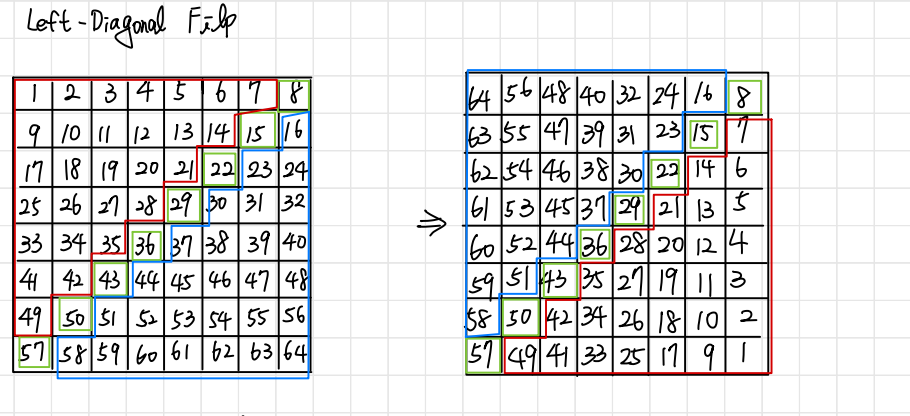
* **Operation 3 – Right Diagonal Flip**

右對角線翻轉(Right Diagonal Flip)，也稱為對角線對稱翻轉，是一種圖像變換操作。這種翻轉沿著圖像的主要對角線（從左上角到右下角）進行，使得圖像中的每個像素點 (i,j) 翻轉到 (j,i) 的位置。



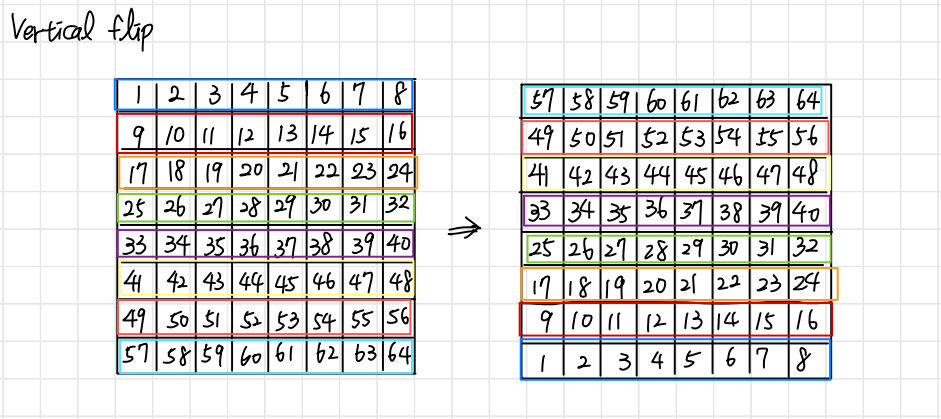
* **Operation 4 – Left Diagonal flip**

左對角線翻轉（Left Diagonal Flip），也稱為反對角線翻轉，是一種圖像變換操作。這種翻轉沿著圖像的次要對角線（從左下角到右上角）進行，使得圖像中的每個像素點 (i,j) 翻轉到 (n−1−j,n−1−i) 的位置，其中n是矩陣的維度。

****

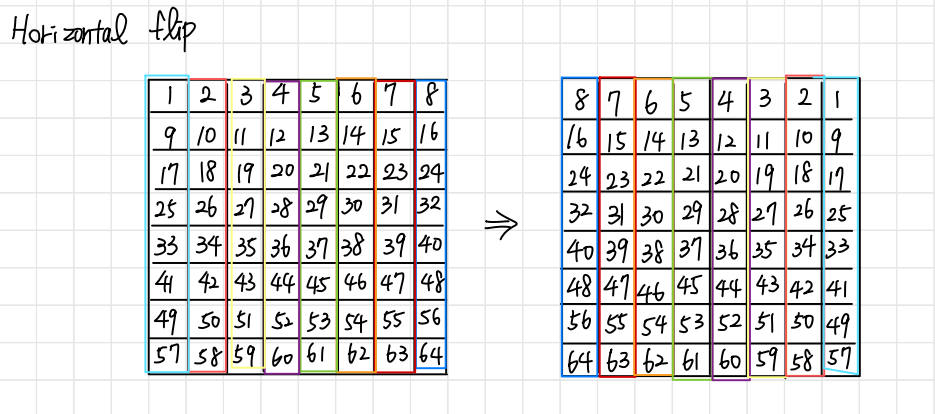
* **Operation 5 - Vertical Flip**

垂直翻轉（Vertical Flip）是圖像處理中的一種基本變換操作。這種操作將圖像沿垂直軸（從上到下）進行翻轉，使得圖像中的每個像素點 (i,j) 翻轉到 (n−1−i,j) 的位置，其中n是圖像的高度。

****

* **Operation 6 – Horizontal Flip**

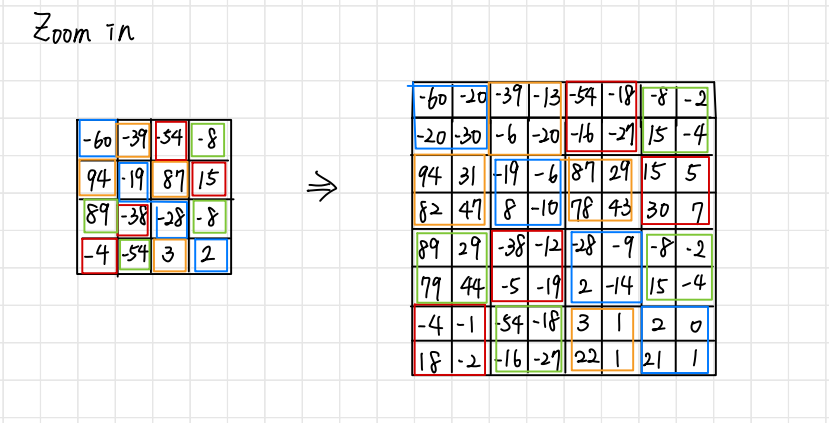
水平翻轉（Horizontal Flip）是圖像處理中的一種基本變換操作。這種操作將圖像沿水平軸（從左到右）進行翻轉，使得圖像中的每個像素點 (i,j) 翻轉到 (i,n−1−j) 的位置，其中n是圖像的寬度。

****

* **Operation 7 – Zoom in**

縮放（Zoom In）是圖像處理中的一種操作，用於放大圖像的一部分，使其顯示得更清晰和詳細。放大圖像通常涉及插值技術，以在圖像被放大後生成新的像素值。而本題使用的是一種基於linear equation的放大操作，該操作會將圖像放大為原來的兩倍，並且每個原始像素會對應放大圖像的左上角像素。給定alpha為0.5，adjust為20，並遵循以下規則：

這些規則將用於從原始圖像生成新的放大圖像。而本題的原始圖像大小為8x8，因此經過zoom-in後的結果圖像將會是原始圖像的兩倍大小，即為16x16，下圖的原始圖像以8x8為例。



* **Operation 8 – Shortcut & Brightness Adjustment**

Shortcut & Brightness Adjustment為一種經典的圖像處理操作，其中選擇圖像的中心並進行亮度調整。具體操作包括使用線性函數調整圖像亮度，並根據圖像大小進行縮放。具體內容如下：

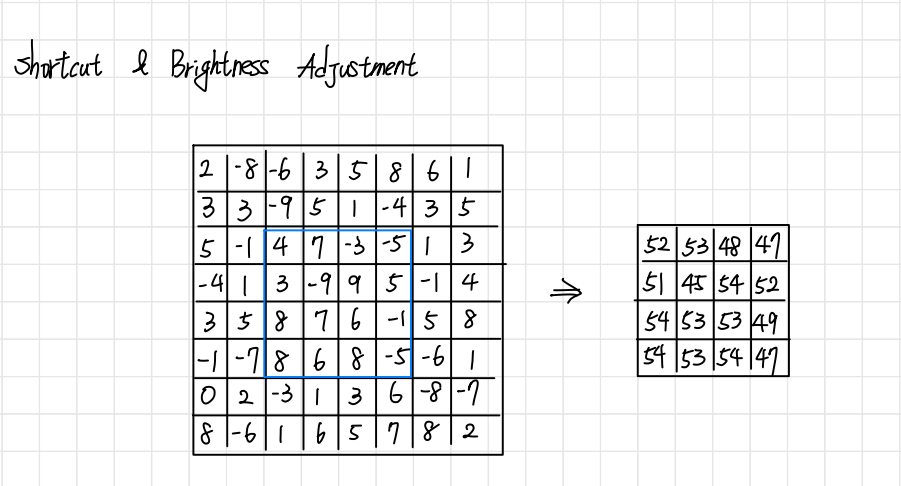
1. **亮度調整公式**：

* 使用線性函數來調整圖像的亮度：
* 其中，alpha設為 0.5，bias 設為 50。

1. **縮放規則**：

* 如果圖像的形狀為 8x8 或 16x16，則進行縮小操作，將圖像大小減半。
* 如果圖像的形狀為 4x4，則不進行縮放，保持原來大小，但依然進行亮度調整。

因本題所使用的image為8x8，因此縮放效果如下圖所示



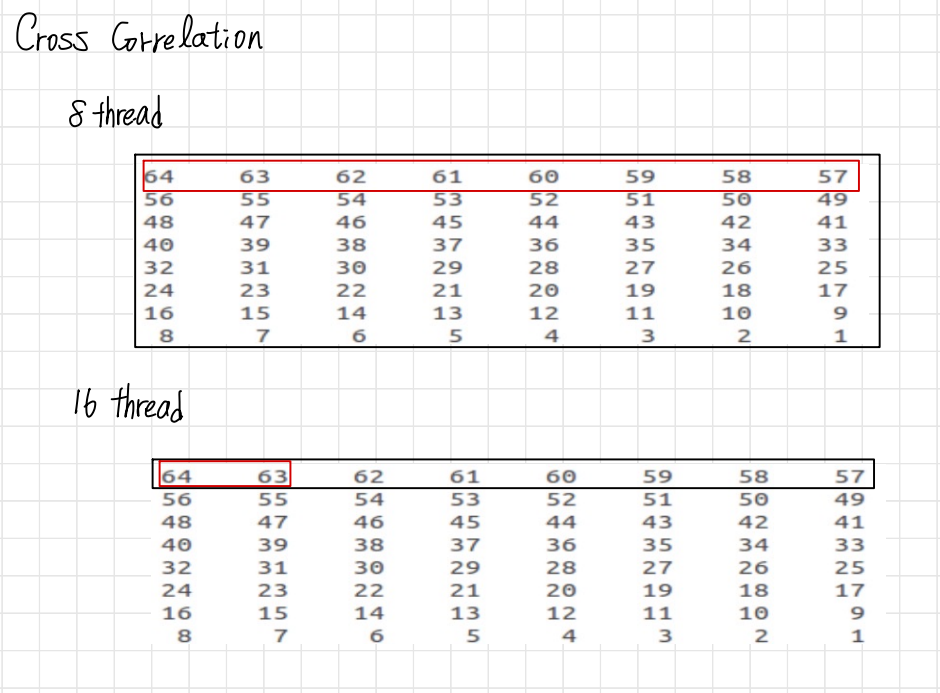
**切割方式**

本次project分別使用8個thread及16個thread來做平行化的處理，下方的圖為本次project的切割方式，其中紅色框框表示一個thread的運算範圍，黑色框框代表一個block的範圍。

* 1. **Cross Correlation**

8個thread的thread\_num=8，block\_num=1。

16個thread的thread\_num=2，block\_num=8。

****

* 1. **Max Pooling**

8個thread的thread\_num=2，block\_num=4。

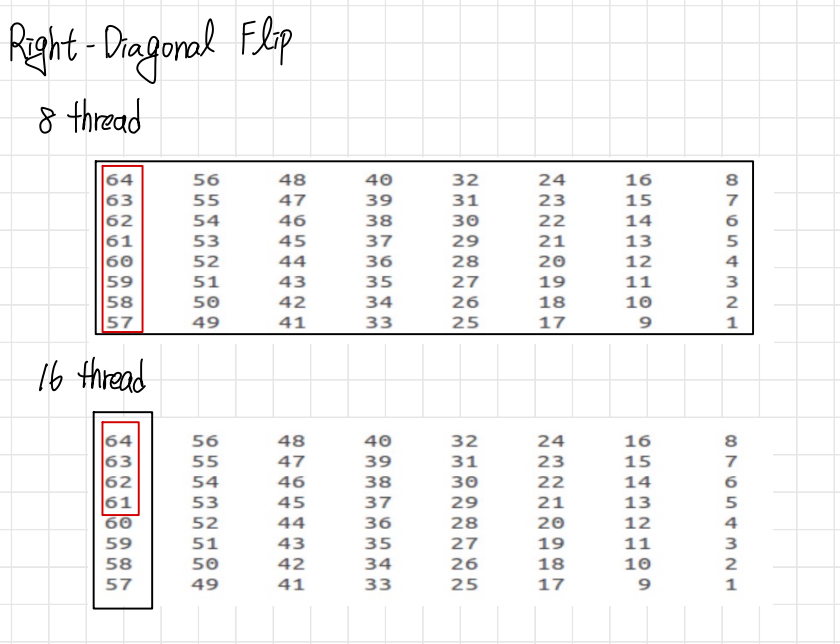
16個thread的thread\_num=4，block\_num=4。

****

* 1. **Right Diagonal Flip**

8個thread的thread\_num=8，block\_num=1。

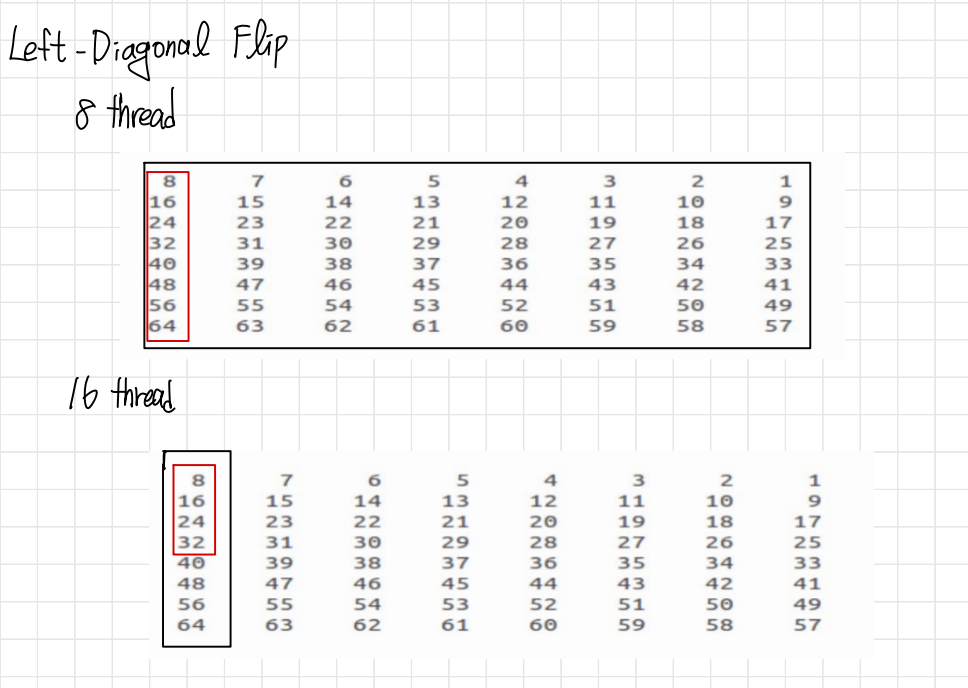
16個thread的thread\_num=2，block\_num=8。

****

* 1. **Left Diagonal Flip**

8個thread的thread\_num=8，block\_num=1。

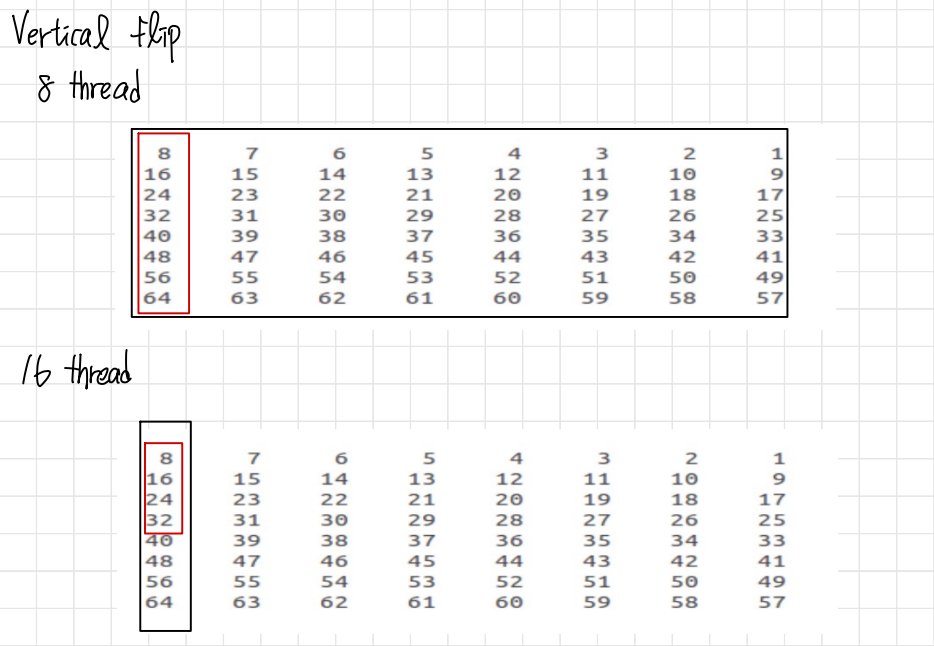
16個thread的thread\_num=2，block\_num=8。

****

* 1. **Vertical Flip**

8個thread的thread\_num=8，block\_num=1。

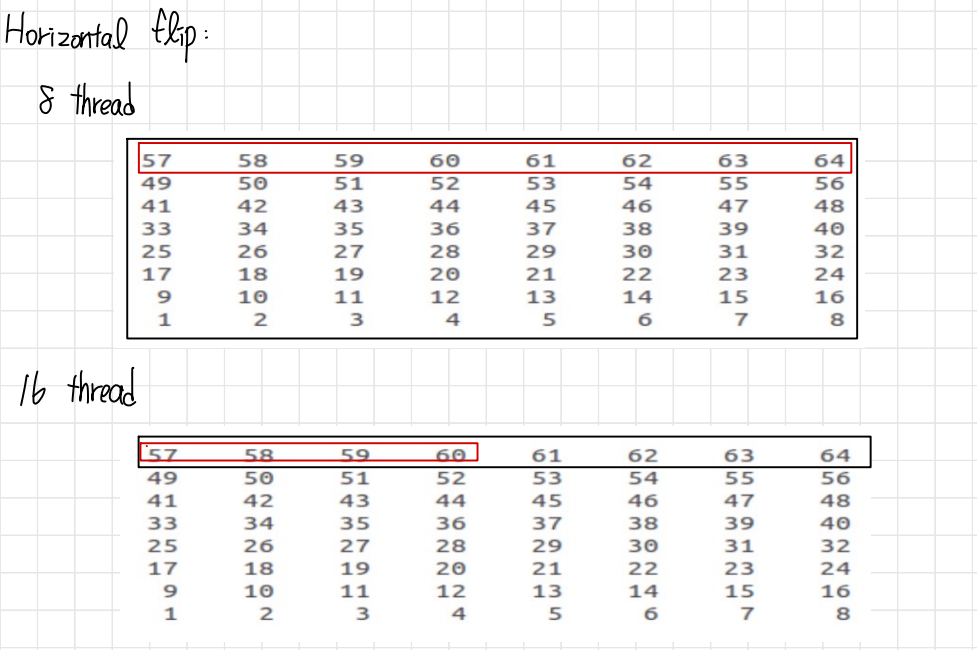
16個thread的thread\_num=2，block\_num=8。

****

* 1. **Horizontal Flip**

8個thread的thread\_num=8，block\_num=1。

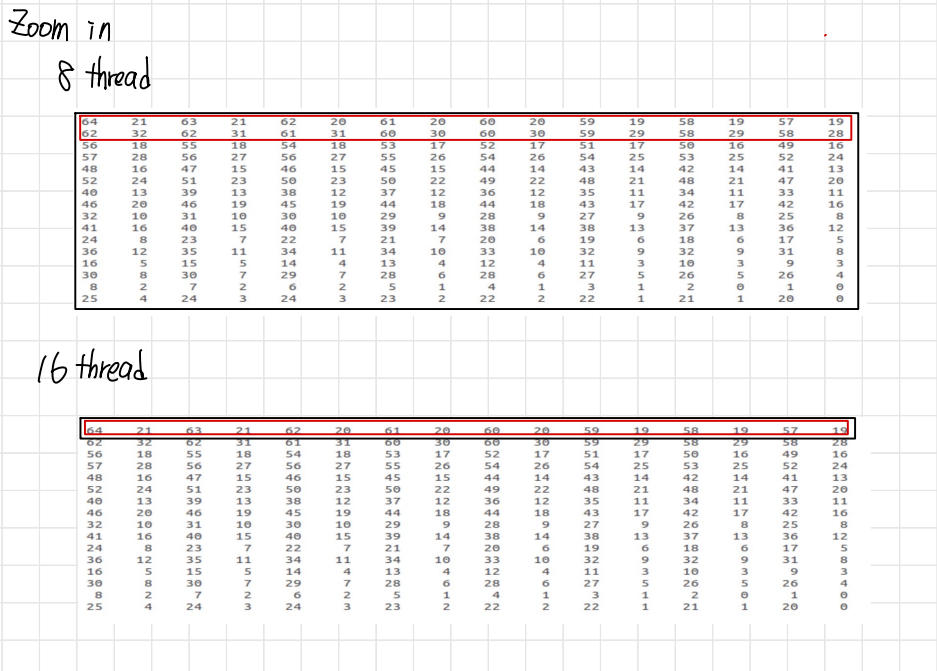
16個thread的thread\_num=2，block\_num=8。

****

* 1. **Zoom in**

8個thread的thread\_num=8，block\_num=1。

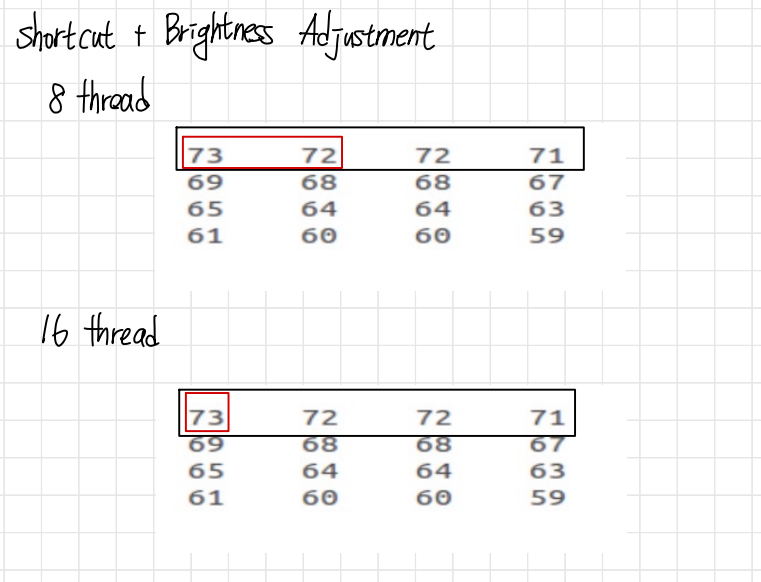
16個thread的thread\_num=1，block\_num=16。

****

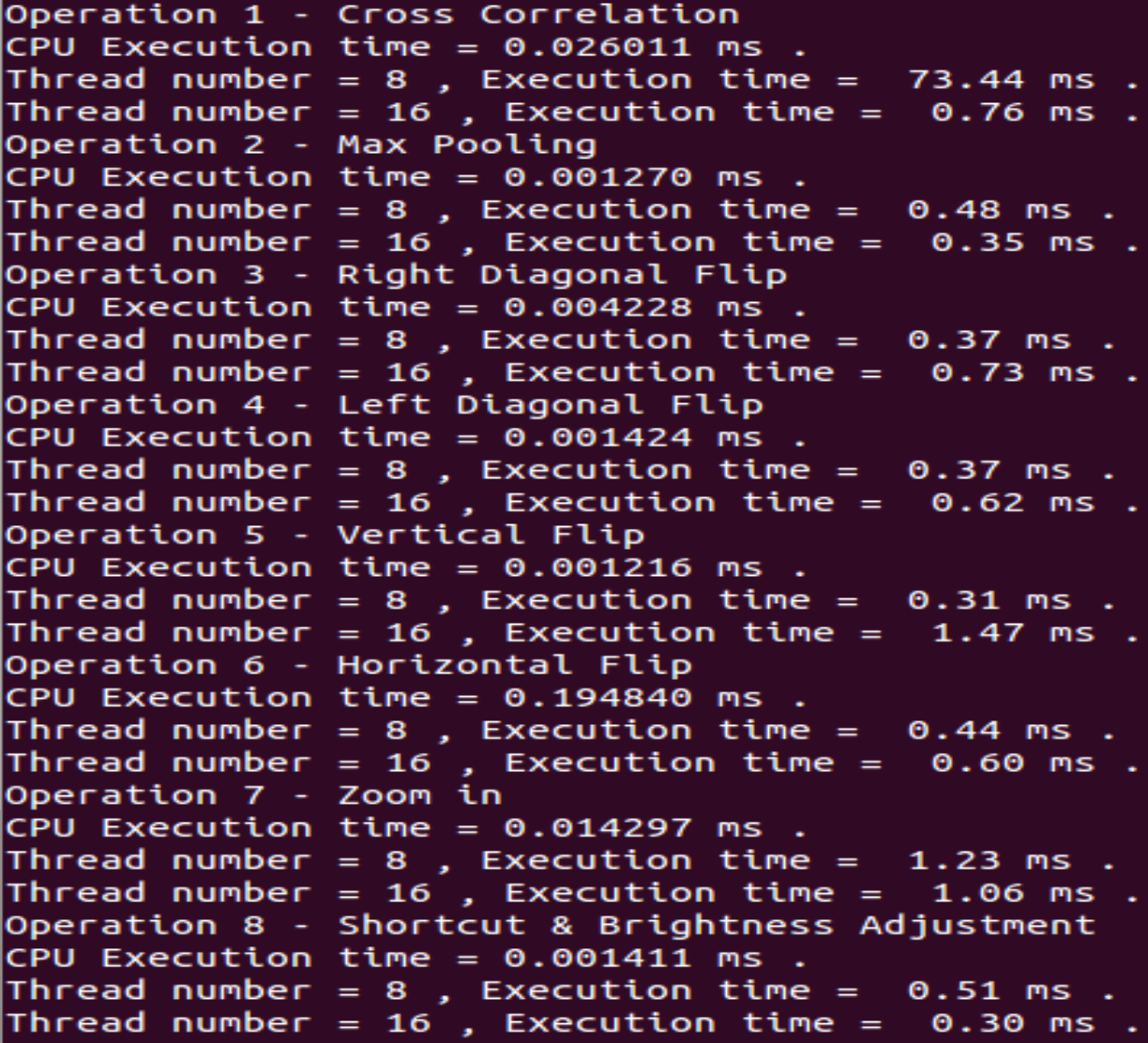
1. **Shortcut & Brightness Adjustment**

8個thread的thread\_num=2，block\_num=4。

16個thread的thread\_num=4，block\_num=4。

****

**模擬結果**

****

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Op1** | **Op2** | **Op3** | **Op4** | **Op5** | **Op6** | **Op7** | **Op8** |
| **8** | **73.44ms** | **0.48ms** | **0.37ms** | **0.37ms** | **0.31ms** | **0.44ms** | **1.23ms** | **0.51ms** |
| **16** | **0.76ms** | **0.35ms** | **0.73ms** | **0.62ms** | **1.47ms** | **0.6ms** | **1.06ms** | **0.3ms** |

**由上面表格我們可以看到，當我們使用thread\_num=8時，他在做一些較複雜的運算時的runtime明顯的比thread\_num=16時大很多，不過再做一些對角化或是一些flip的時候，兩者的rumtime卻無明顯差別，甚至thread\_num=16時還比thread\_num=8時大，我們認為這是因為這幾個operation因為沒有做甚麼複雜的運算，只是單純的移動image的內容，因此這些operation用來做平行化的處理無法得到我們預期的效果，不過當我們運算量變大時，例如cross correlation中要實作convolution，很明顯的我們的平行化處理就很成功，在max pooling時，因為要四個比大小，因此thread\_num=16時也較佳，而在zoom in以及brightness adjustment，因為要根據一些linear equation來計算結果，因此當thread\_num變大時也有得到相應的好處。**

**結論**

根據上述的實驗結果，我們可以得出以下結論：

* **平行化處理的有效性**：在進行較為複雜的運算時，增加執行緒數目可以顯著提高計算效率。例如，在cross correlation中實作卷積運算時，使用16個執行緒的運行時間顯著少於使用8個執行緒，這表明平行化處理在這些計算密集型操作中能夠充分發揮其優勢。
* **簡單操作的平行化限制**：對於一些簡單的圖像操作（如對角化和翻轉），平行化處理的優勢並不明顯，甚至在某些情況下，使用更多的執行緒反而會增加運行時間。這是因為這些操作主要涉及數據移動，並不涉及大量的計算，平行化處理在這些場景下無法顯著減少運行時間，反而可能因為多執行緒管理和調度的開銷而增加總運行時間。
* **適用場景的不同**：平行化處理在需要大量計算的場景中，也能夠取得不錯的效果。這些操作涉及基於線性方程的計算，增加執行緒數目能夠有效分攤計算負荷，從而縮短運行時間。

總結來說，平行化處理在計算密集型操作中能夠顯著提高運行效率，但在僅涉及簡單數據移動的操作中，其效果有限。選擇適當的執行緒數目應根據具體操作的計算特性來決定，以平衡平行化處理的優勢與執行緒管理的開銷。

**分工**

|  |  |
| --- | --- |
| 312510239 王則惟 | 312510144 張理為 |
| cuda1.cu  cuda2.cu  書面報告 | main.cu  Makefile  書面報告 |