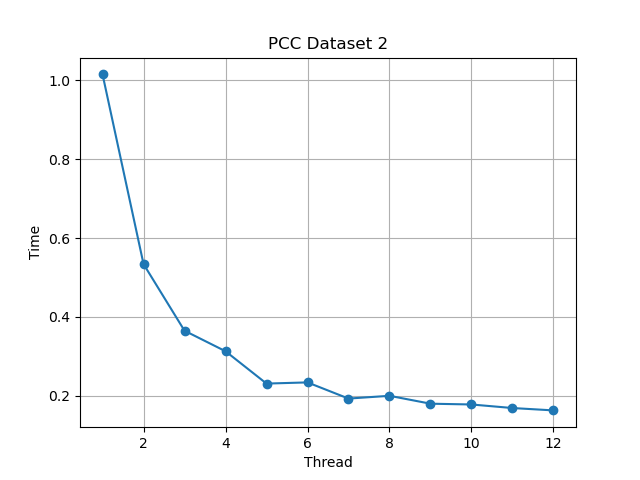
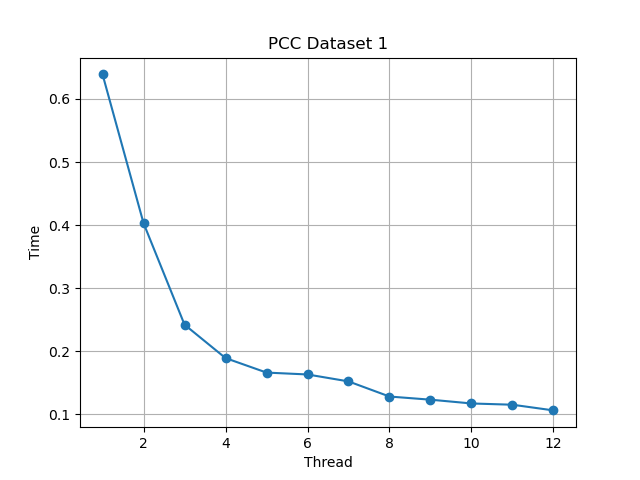
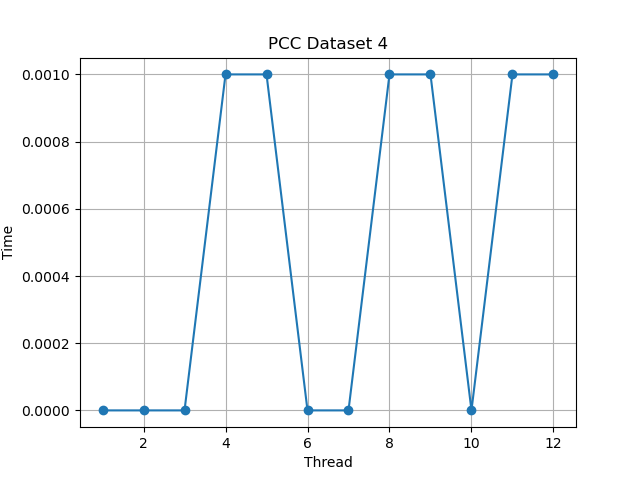
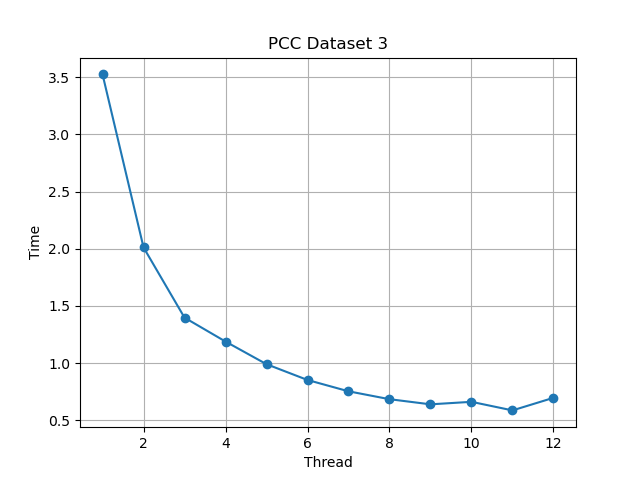
2024 EMCSS-Pthread

A1095551 廖怡誠

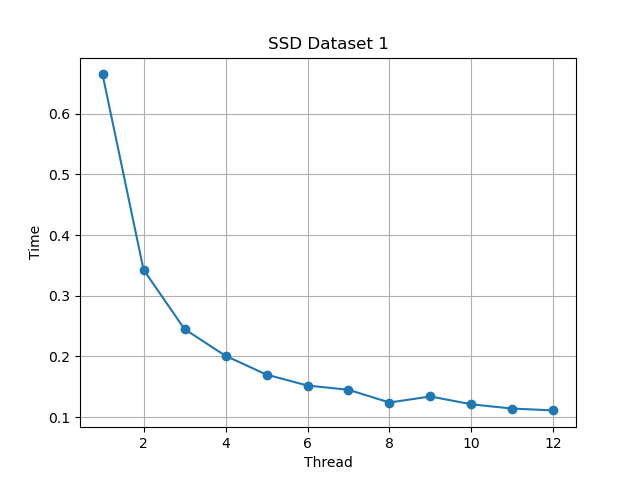
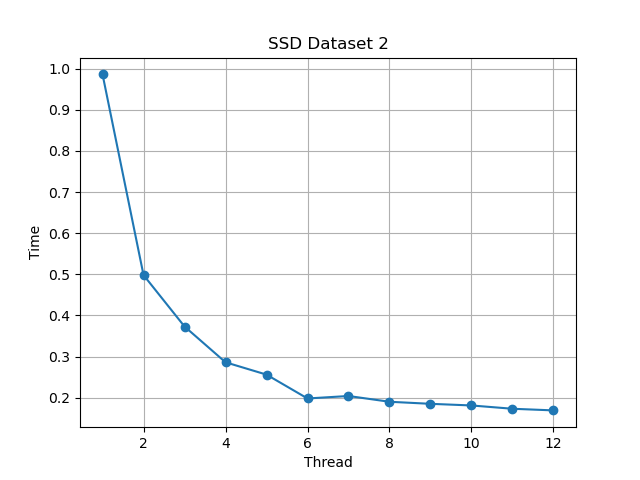
* 實驗結果:

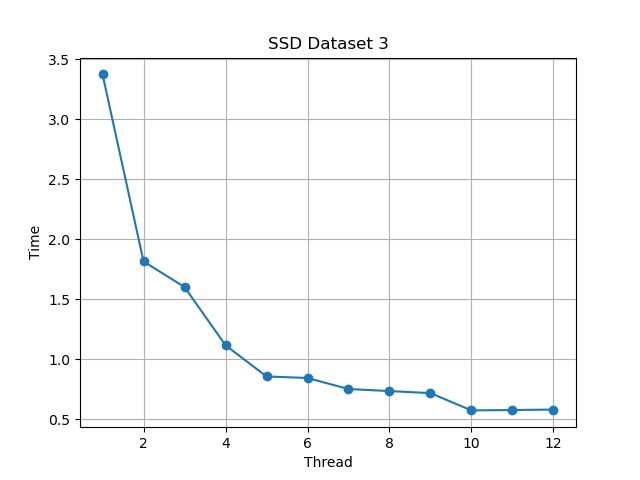
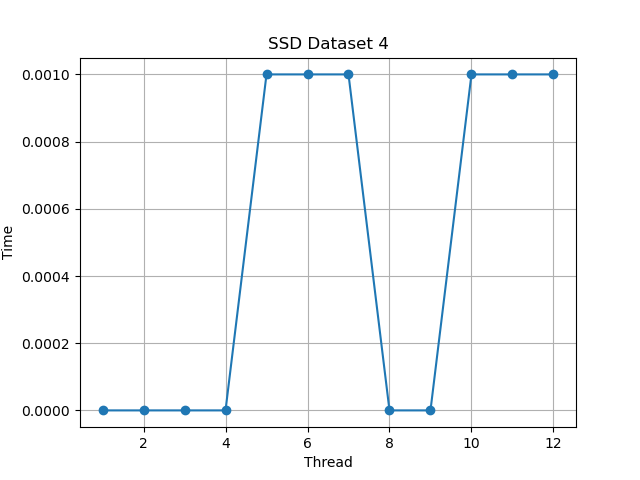
使用Prhread分別以1個thread到12個thread測試PCC(圖一到圖四)與SSD(圖五到圖八)在Dataset1到Dataset4的運作時間。

 **圖一、PCC Dataset1 圖二、PCC Dataset2**

 **圖三、PCC Dataset3 圖四、PCC Dataset4**

在除了PCC Dataset4的圖表中能觀察在使用2個Thread的運行時間比使用1個Thread的運行時間少耗費一半的時間，以Dataset2來說，1個Thread約為1.2秒，而2個Thread約為0.5秒，使用越多的Thread同步運算能減少的運行時間有顯著的減少，但由於硬體上的限制，使用11、12個Thread使用時，效能上已經沒有太大的變化。

而在Dataset 4中，由於矩陣的大小太小，因此運行時間上均為0.001秒左右，不太能看出明顯的差異，也可以證明Thread的同步運行應用需要在資料量較大的情況下使用，效果才能有明顯的改善，如矩陣大小最大的Dataset3能夠從3.5秒縮短5倍至0.7秒。

**圖五、SSD Dataset1 圖六、SSD Dataset2**

**圖七、SSD Dataset3 圖八、SSD Dataset4**

在SSD的計算中，運算的結果與PCC差異不大，但由於SSD本身的運算量小於PCC的運算量，因此若將兩者的圖表進行比較，SSD的整體運算成本會比PCC快上一些。

* 實作細節

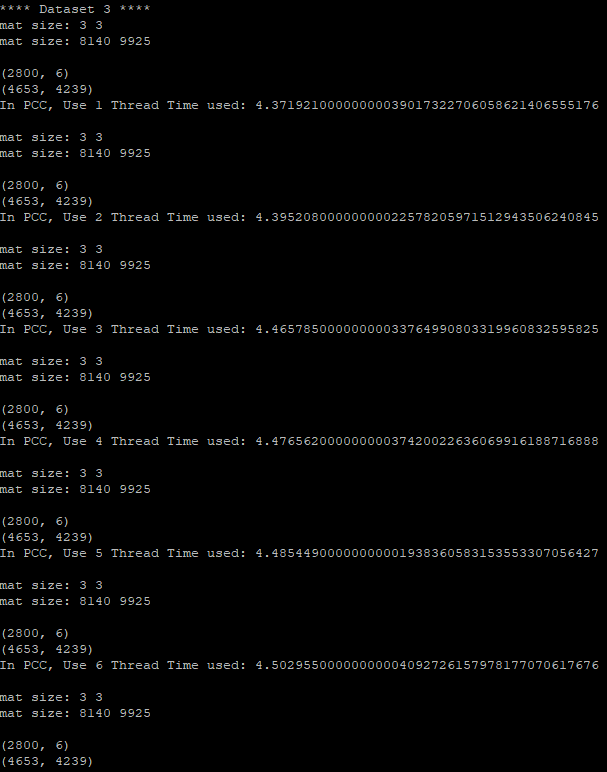
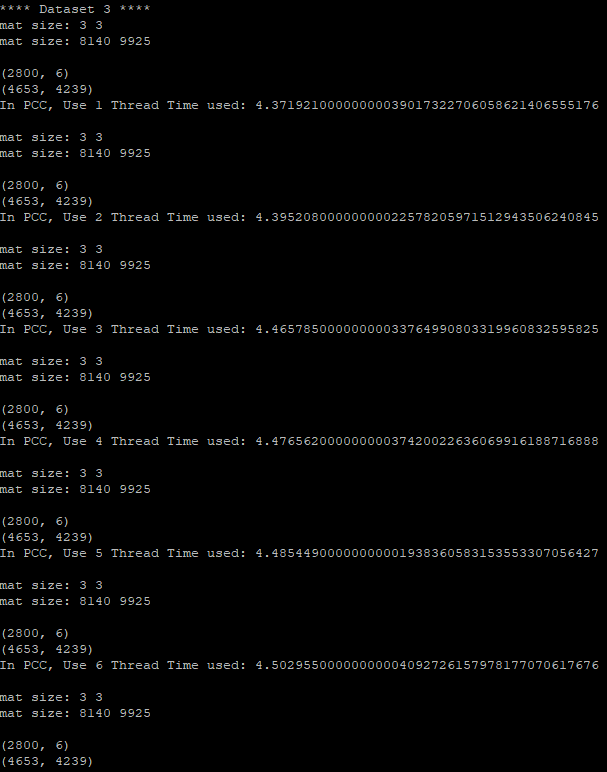
在這次的作業中，與CUDA的作業差別頗大，在CUDA中能夠使用Shared memory與數量較多的Thread來達到更快的運行速度，而在Pthread中，硬體限制在能夠使用的核心數少，能運用的Thread也較少。

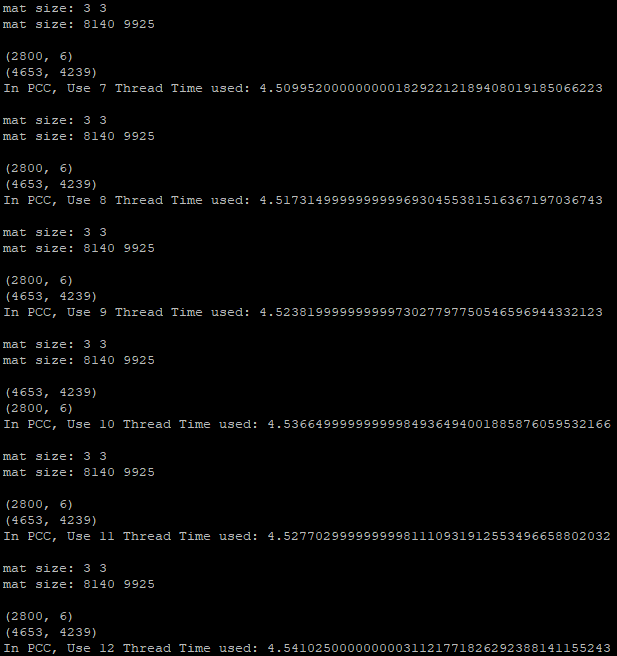
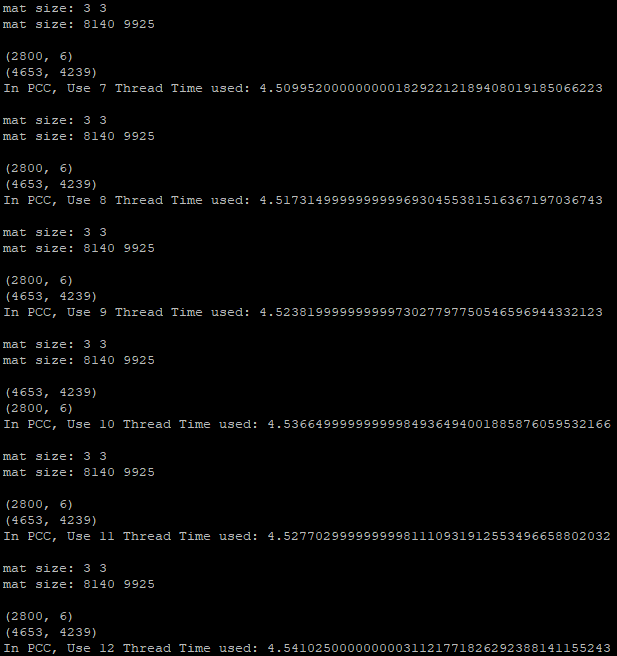
不過，在這次的設計中，我參考CUDA中以thread id分段計算矩陣的技術，有12個Thread則能夠將矩陣劃分成12個區塊進行運算，而由於PCC與SSD的計算中，並不會有變更矩陣數值的問題，所以不會使用到Mutex或是Lock技術。

但以我設計的方法來說，除了thread id之外，其他的參數如Source矩陣、Target矩陣和矩陣大小都不會變更，可是在pthread\_create( )使用上會是以指標型態將參數傳入到不同的Thread中，因此就需要在每次建立時都配置一個新的傳入值，所需要的記憶體空間隨著Thread的數量上升，消耗也會隨之變高，因此這個方法應該可以有更好的改善方案。

* 作業遇到的困難

本次作業中，遇到最大的問題是原先使用教授提供的主機進行運算，在1個Thread到12個Thread的運行時間差異不大(下圖所示)，原先的猜測是計算的優化不足導致成效不好，因此也考慮是否因為以區段劃分矩陣，造成Thread在記憶體的存取花費時間較長，想要改成連續記憶存取，又或是Thread之間其實沒有同步運行。後來將原先的程式碼轉到本機電腦運行，1個Thread到12個Thread的運行時間就能夠有明顯的差異。





* 心得

在剛開始寫Pthread時，因為對其中的函式使用不太熟悉，導致使用多個pthread\_create( )時，出現期望的結果沒有出來，程式就已經結束的情況，後來才明白在建立thread之後，都需要加上pthread\_join( )等待指定Thread執行完成再接續後面的程式段落，否則就可能出現Thread程式還沒完成就結束程式的問題。

而在實作的設計中，原先想要嘗試Blocking技術或是類似Shared Memory方法，但在Pthread的應用上不大，因為每個thread之間都是獨立的暫存器，所以沒辦法使用Shared Memory，最有效的方法應該是減少重複的記憶體存取，減少cache miss次數，所以使用Blocking應該能有效提高效率、減少運算時間。