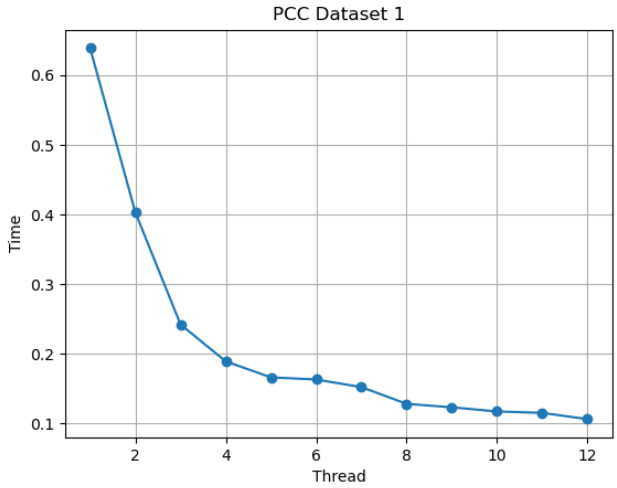
2024 EMCSS-Pthread

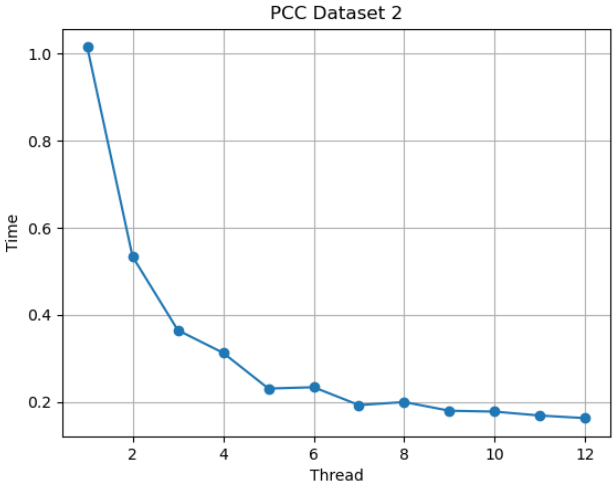
A1095551 廖怡誠

- 實驗結果:

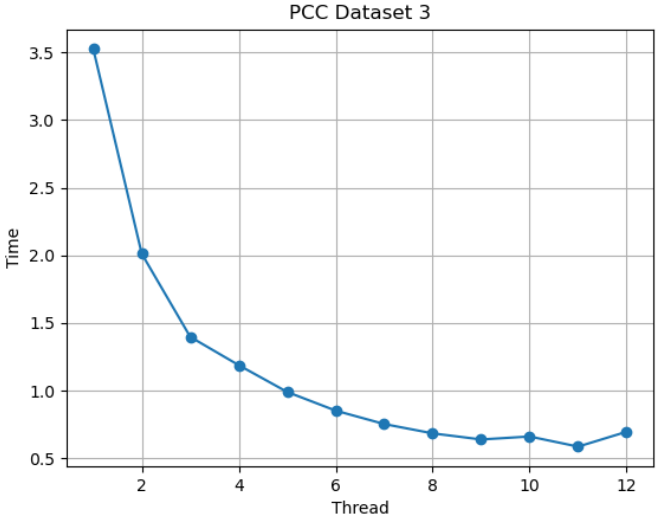
使用 Prhread 分別以 1 個 thread 到 12 個 thread 測試 PCC(圖一到圖四)與 SSD(圖五到圖八)在 Dataset1 到 Dataset4 的運作時間。



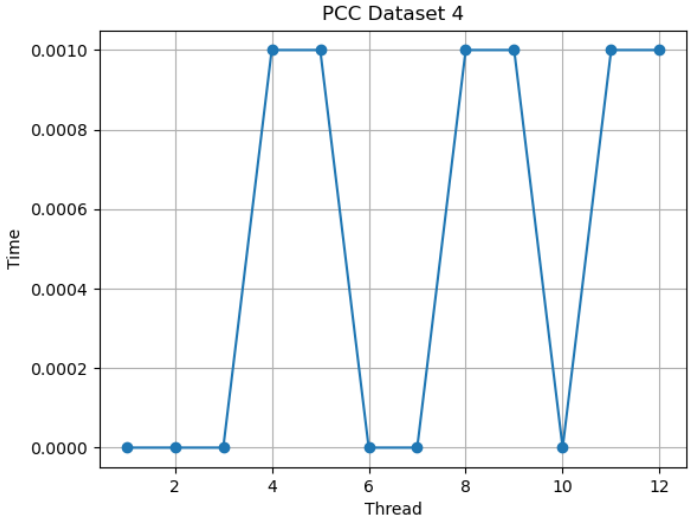
圖一、PCC Dataset1



圖二、PCC Dataset2



圖三、PCC Dataset3

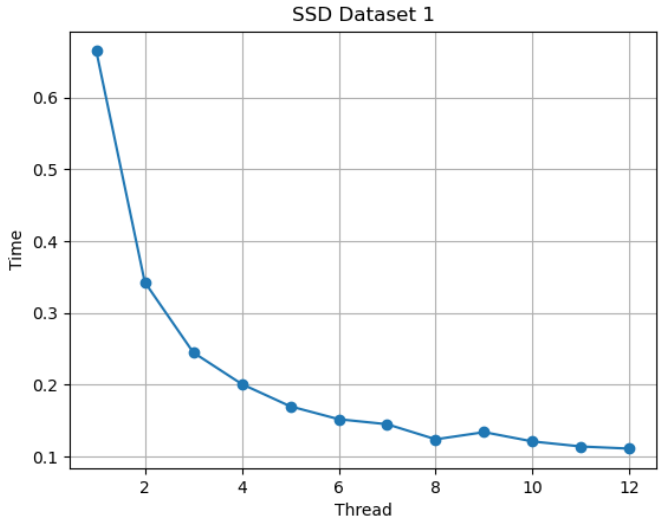


圖四、PCC Dataset4

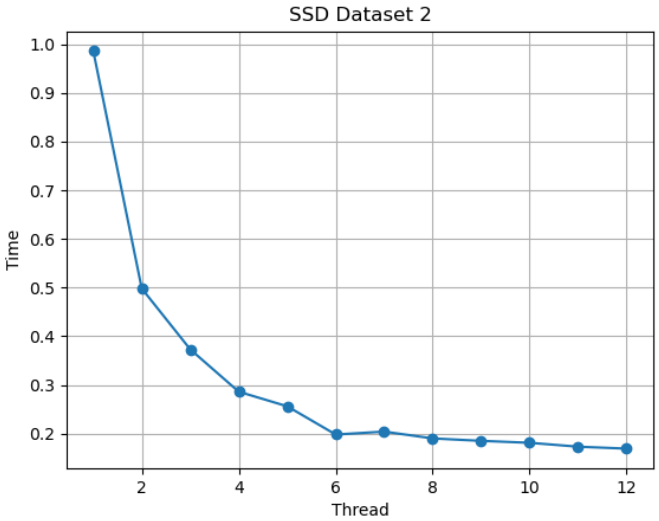
在除了 PCC Dataset4 的圖表中能觀察在使用 2 個 Thread 的運行時間比使用 1 個 Thread 的運行時間少耗費一半的時間，以 Dataset2 來說，1 個 Thread 約為 1.2 秒，而 2 個 Thread 約為 0.5 秒，使用越多的 Thread 同步運算能減少的運行時間有顯著的減少，但由於硬體上的限制，使用 11、12 個 Thread 使用時，效能上

已經沒有太大的變化。

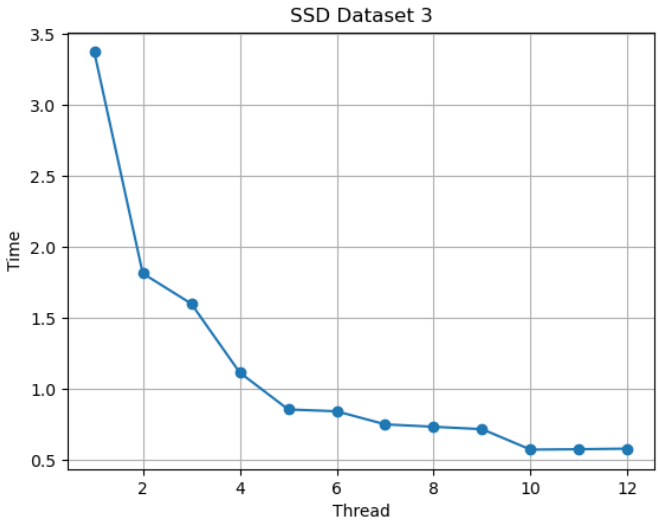
而在 Dataset 4 中，由於矩陣的大小太小，因此運行時間上均為 0.001 秒左右，不太能看出明顯的差異，也可以證明 Thread 的同步運行應用需要在資料量較大的情況下使用，效果才能有明顯的改善，如矩陣大小最大的 Dataset3 能夠從 3.5 秒縮短 5 倍至 0.7 秒。



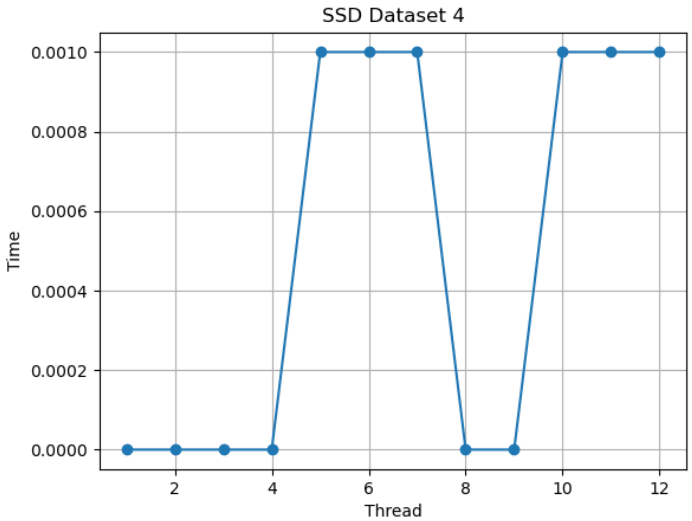
圖五、SSD Dataset1



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。

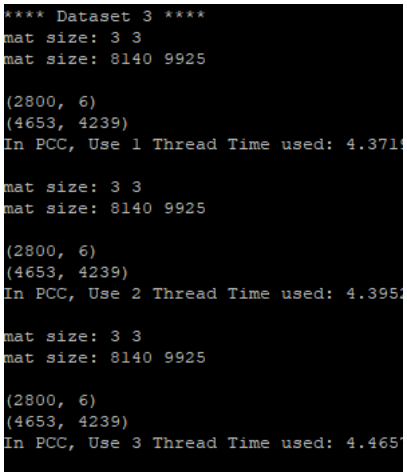
● 實作細節

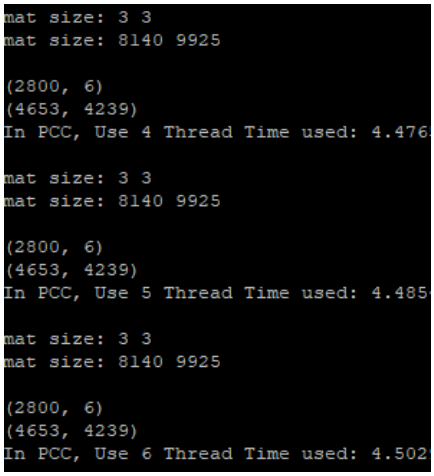
在這次的作業中，與 CUDA 的作業差別頗大，在 CUDA 中能夠使用 Shared memory 與數量較多的 Thread 來達到更快的運行速度，而在 Pthread 中，硬體限制在能夠使用的核心數少，能運用的 Thread 也較少。

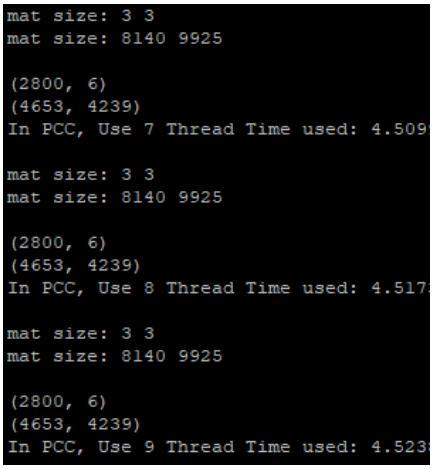
不過，在這次的設計中，我參考 CUDA 中以 thread id 分段計算矩陣的技術，有 12 個 Thread 則能夠將矩陣劃分成 12 個區塊進行運算，而由於 PCC 與 SSD 的計算中，並不會有變更矩陣數值的問題，所以不會使用到 Mutex 或是 Lock 技術。但以我設計的方法來說，除了 thread id 之外，其他的參數如 Source 矩陣、Target 矩陣和矩陣大小都不會變更，可是在 pthread_create( )使用上會是以指標型態將參數傳入到不同的 Thread 中，因此就需要在每次建立時都配置一個新的傳入值，所需要的記憶體空間隨著 Thread 的數量上升，消耗也會隨之變高，因此這個方法應該可以有更好的改善方案。

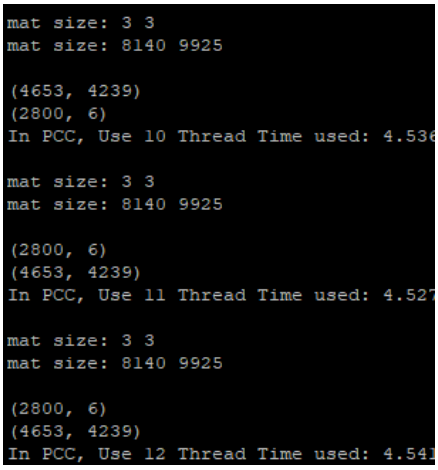
● 作業遇到的困難

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。