OS_HW1 結果報告 資訊三乙 10727224 李敏嘉

一、 開發環境

使用語言: python

IDE: Visual Studio Code

二、 實作方法和流程

Task1: 將 input 存入 list 後，直接進行 Bubble Sort，並計算執行時間，最後輸出結果。Bubble Sorth 參考演算法課本。

Task2: 將 input 存入 list 後，切成 K 份，建立 K 個 Threads，並將要做 Bubble Sort 的參數存入 inputs[](list)中，待 Bubble Sort 的 Threads 完成之後，再建立 N 個 Threads 做 Merge Sort。Merge Sort 是將剛完成的 Bubble Sort 的前二筆(Thread 完成之結果)取出，做 Merge Sort 後成為”一筆”資料並插回 list 的最後面。一直重複取出，做 Merge Sort，插回 list 最後面，直到 list 只剩下”一筆”資料後，輸出結果。

Task3: 將 input 存入 list 後，切成 K 份，建立 K 個 Process，並將要做 Bubble Sort 的參數存入 inputs[](list)中，待 Bubble Sort 的 Process 完成之後，再建立 N 個 Process 做 Merge Sort。其中使用 Pool 去管理 Process。Merge Sort 是將剛完成的 Bubble Sort 的前二筆(Process 完成之結果)取出，做 Merge Sort 後成為”一筆”資料並插回 list 的最後面。一直重複取出，做 Merge Sort，插回 list 最後面，直到 list 只剩下”一筆”資料後，輸出結果。

Task4: 將 input 存入 list 後，在一個 Process 內，將資料切成 K 份分別做 Bubble Sort，再用同一個 Process 做 Merge Sort，與 Task3 方法一樣，最後輸出結果。

三、 比較結果

K: 將資料切成 K 份

1w : 1 萬筆資料 (10w: 10 萬筆資料…以此類推)

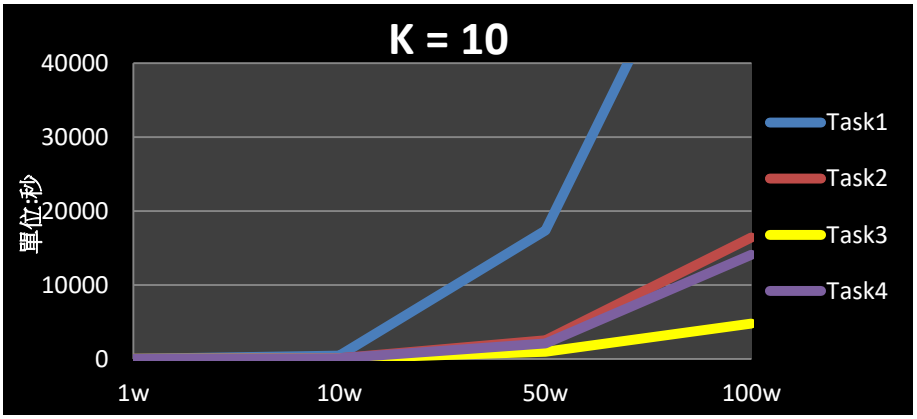
時間單位: 秒

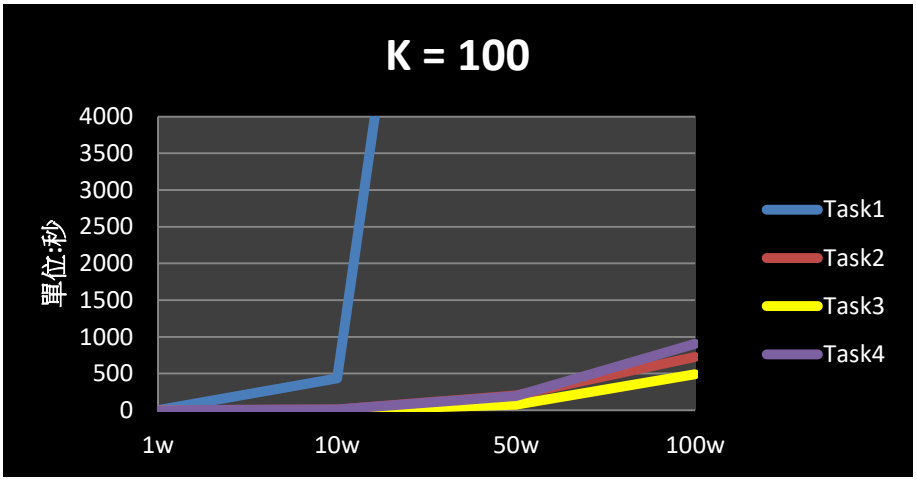
k = 10	1w	10w	50w	100w
Task1	3.867177	434.867	17364.51	73956.9
Task2	0.848672	77.93304	2533.043	16419.47
Task3	0.379543	28.60685	955.7442	4787.009
Task4	1.093429	76.16247	2070.674	14071.7

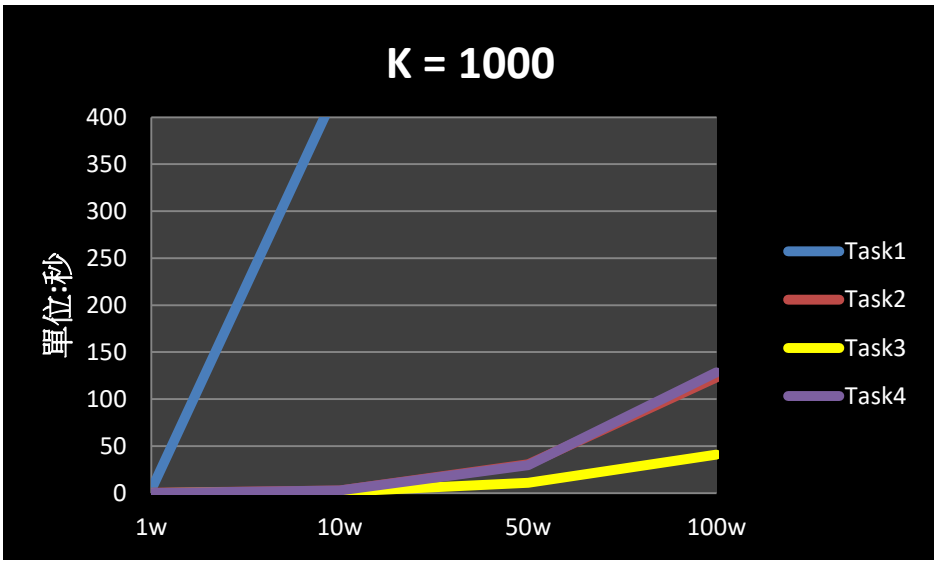
k = 100	1w	10w	50w	100w
Task1	3.867177	434.867	17364.51	73956.9
Task2	0.223356	8.723688	201.6048	725.7749
Task3	0.165179	3.038562	74.59228	490.2073
Task4	0.179074	8.527724	194.0098	902.3592

k = 1000	1w	10w	50w	100w
Task1	3.867177	434.867	17364.51	73956.9
Task2	0.322621	2.88331	30.73375	123.018
Task3	0.17749	1.317109	10.92781	41.0273
Task4	0.151762	2.683291	29.527	128.3458

以折線圖表示:







Task1 的執行速度最慢，Task3 的執行速度一直都是最快的，Task2 和 Task4 則差不多。

當 K 切越多份，則 Task2, Task3, Task4 執行速度會越快

四、 分析結果原因

Task1: 直接將資料做 Bubble Sort，當資料數量增加，執行時間也增加不少，Bubble Sort 的時間複雜度為:Θ(n^2)，每次 Bubble Sort 皆要從頭將資料比對過一遍才能確定一筆資料的位置，故執行時間會隨資料量的增加而越來越久。

Task2, Task3, Task4:Multi-Thread 共用一份 process 中的 address space，其中的 code, data, resource 皆給 K 個 Thread 共用，當一個記憶體區塊被一個子 Thread 使用的時候，其他的 thread 不能取動用，否則結果會錯誤，相較於 Multi-Process 來說，就會需要多一些時間。而 Multi-Process 中，每個 Process 皆有一份自己的資料，每個 Process 也需要一個自己的資源來工作，所以 Multi-process 比 Multi-thread 更消耗資源，但也因不需等待別的 Process 而更快。

當 K 的份數越來越大且資料量更大時，能看出建立 Multi-Thread 的效果優於一個 Process，有可能是因為課堂上所提的—開一個 Process 比開 Thread 的代價還要大，而且 Thread 在 context switch 的速度很快，所以 Task2 有可能會比 Task4 快。 不過 Multi-Thread 只達到了 Concurrent 的效果，但 Multi-Process 才能做到達到 Parallelism。所以 Task3 就會比 Task2 快。

參考資料:

Thread:

https://blog.gtwang.org/programming/python-threading-multithreaded-programming-tutorial/

Process:

http://python-learnnotebook.blogspot.com/2018/11/python-multiprocessing-process-and-pool.html

https://codychen.me/2019/12/%E5%A4%9A%E9%80%B2%E7%A8%8B/%E5%A4%9A%E5%9F%B7%E8%A1%8C%E7%B7%92-%E4%B8%A6%E7%99%BC/%E5%B9%B3%E8%A1%8C/

Concurrent&Parallelism

https://ithelp.ithome.com.tw/articles/10225707?sc=rss.iron