作業系統 hw1 文件

資訊四甲
10827108
鄞志良

一、開發環境

在主機上使用虛擬環境開發

- 主機配備: Intel® Core™ I5-8265U CPU @ 1.60GHz, 8G RAM
- 虛擬機: 5 核心處理器、10G 記憶體
- 作業系統: Linux 64bit (Ubuntu 16.04)
- 程式語言: C++11

二、實作方法與流程

主程式流程:

1. 取得使用者輸入(檔名、切割份數、方法數)，皆已防呆處理
2. 根據使用者方法數，進入對應函式排序
3. 計時開始
4. 執行排序
5. 計時結束
6. 寫檔
7. 回到第一步輸入使用者資料

Bubble sort:

1. 外部迴圈選定最後一個元素開始排序
2. 內部迴圈和相鄰的元素做比較，較小的往前移，直到外部迴圈跑完

Merge sort:

1. 回傳兩陣列元素至函式中
2. 比較兩陣列排頭元素，較小的存入大陣列中，直到兩陣列元素讀完為止
3. 若剩餘陣列還未跑完，把剩餘的陣列元素存入大陣列中

方法一(直接 bubble sort):

呼叫 Bubble sort 函式直接排全部元素

方法二(一個 process 對 k 份資料 Bubble sort 完 Merge sort):

1. 將資料平均切成 k 份資料，存入二微陣列中
2. 對這些 k 份資料傳入 Bubble sort 函式做排序
3. 將這些 k 份資料倆兩傳入 Merge sort 做排序，一直到合併剩最後一份

方法三(多個 process 對 k 份資料 Bubble sort 完 Merge sort):

1. 將資料平均切成 k 份資料，存入二微陣列中
2. 建立儲存全部資料的共享記憶體(供多處理元之間使用)
3. 建立 k 個 process，每個 process 將一份資料傳入 Bubble sort 函式做排序
4. 建立 k-1 個 process，每個 process 將兩份資料傳入 Merge sort 做排序，一直到合併剩最後一份

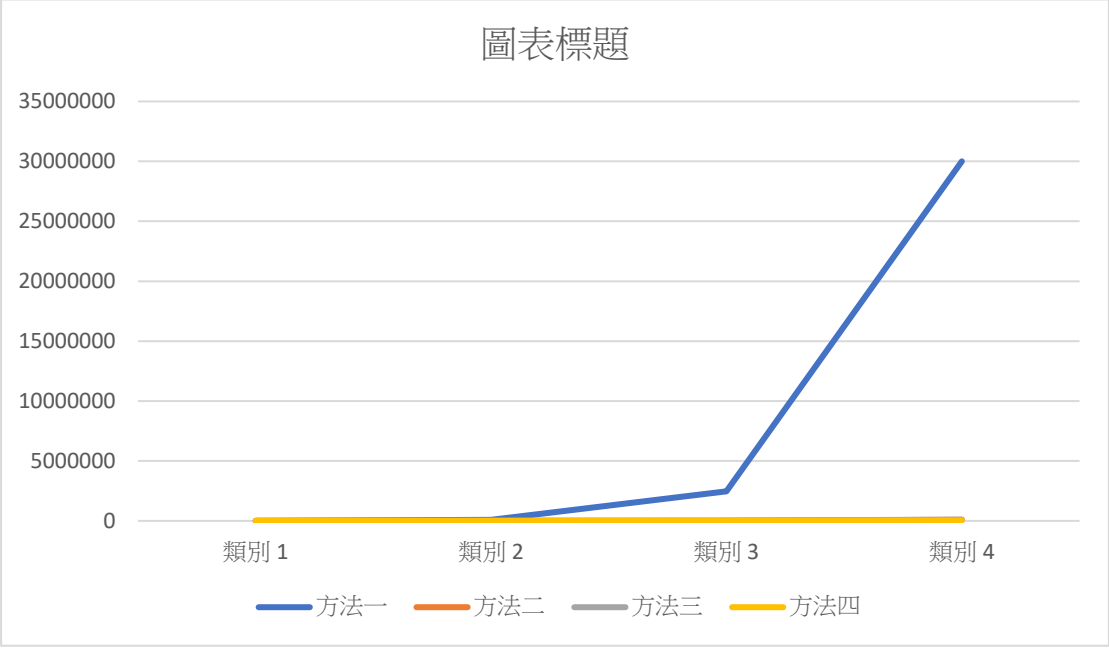
方法四(多個 thread 對 k 份資料 Bubble sort 完 Merge sort):

1. 將資料平均切成 k 份資料，存入二微陣列中
2. 建立 k 個 thread，每個 thread 將一份資料傳入 Bubble sort 函式做排序
3. 建立 k-1 個 thread，每個 thread 將兩份資料傳入 Merge sort 做排序，一直到合併剩最後一份

四、分析結果和原因

實驗一(單位 ms)

K = 100	一萬筆	十萬筆	五十萬筆	一百萬筆
方法一	1090	98532	247491	非常大
方法二	28	1040	110566	97427
方法三	101	1182	24951	98949
方法四	37	282	5706	28298

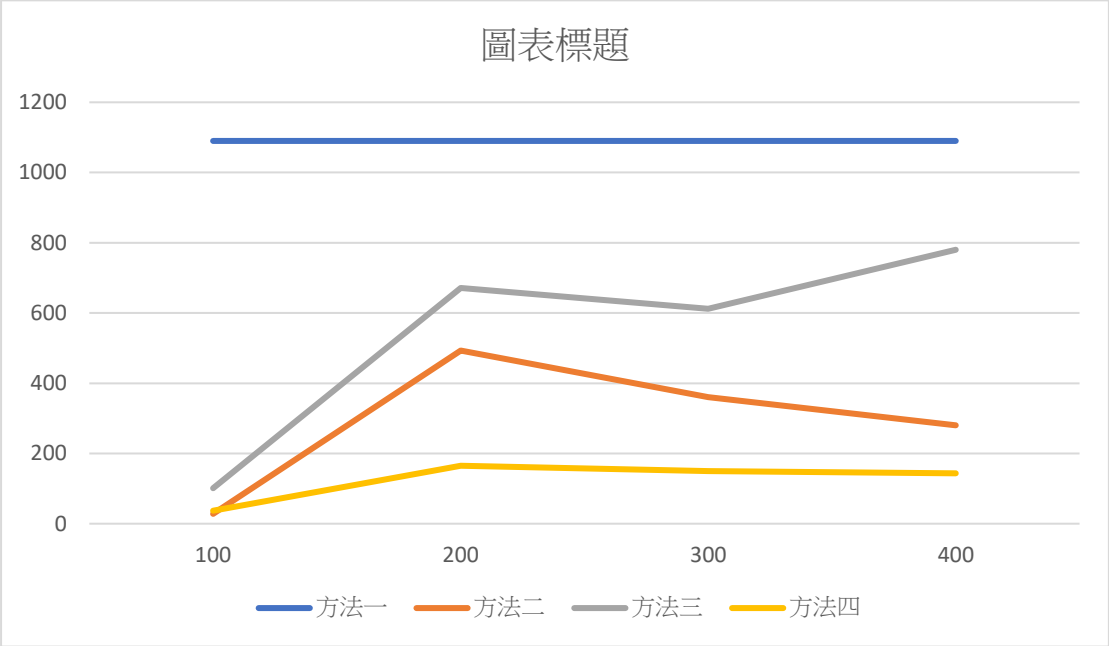


由圖表可以看出，執行效率最差的是方法一，其次為方法二、方法三，方法四效率為最佳。

由理論得知氣泡排序的執行效率比合併排序慢，方法一和方法二的差距就是最明顯的證明，實驗結果及理論符合，同時也觀察到方法三及方法四快於方法二，原因是使用了多處理元及多執行緒，導致排序工作同步並行處理，方法四又略快於方法三，會缺少這一小部份時間，可能是由於方法三在做 CPU 排程切換時，需要耗費較多時間去做 Context Switch，方法四 User Thread 在 CPU 中的排程僅是一個單位。

實驗二(單位 ms)

N= 50 萬	K=100	K=200	K=300	K=400
方法一	1090	1090	1090	1090
方法二	28	493	360	280
方法三	101	671	612	780
方法四	37	165	150	143



由圖表可以看出，k 與方法一無關，方法四效率最佳，且當 K 值愈大，效率愈佳。

根據實驗數據，當 k 越大，這意味著切成很多份，而導致一份資料內的元素下降進而影響需要動到氣泡排序的元素變小，且由上一個實驗的結果及理論得知氣泡排序的執行效率小於合併排序，在此實驗中能觀察出 k 越大，效率會加快，此原因來自 k 越大越像合併排序，但若太大也無法保證方法三和方法四的效率會比較好，系統可能要花時間去建立 Process 或 Thread 就為了對只有一個元素的陣列做氣泡排序，其效率自然可能不會是最佳。