

OS-HW1-Process & Thread



資訊三甲

10927143

王胤迦

開發環境: Window 10 / MinGW 8.1.0

Ubuntu 20.04 /

實作方法和流程:

方法一:用fstream讀檔案，用vector存起來，之後呼叫BubbleSort，將資料排序，排序完用ofstream將資料輸出成文字檔，同時會計算CPU Time和當前時間。

方法二:根據K值決定要分成多少份，利用檔案比數除以K得到每份有多少筆資料，之後用迴圈把每份資料做BubbleSort，再來就是Merge的部分，我是採用兩兩合併之後在兩兩合併，一直持續到剩下一個vector陣列為止，合併完成後如方法一寫檔。

方法三:切完檔案後建立共享記憶體區，將要BubbleSort的部分共享之後再每個process中利用共享的記憶體去做Sort，會等到所有子process都排序好才進行後續的Merge，一樣將要Merge的兩個陣列和產出的陣列空間共享起來，直到合併完為止，將資料寫檔，結束任方法三。

方法四:建立Thread，用vector把每個thread存起來，之後用for迴圈直到每個thread都join之後，才做Merge的部分，同multiprocessing，由於要合併的關係，因此每個thread都要等前一個thread合併完成才能繼續下一個thread的動作，因此要等前一個thread join後才能執行下去，一樣合併完了之後寫檔。

另外Linux編譯thread的指令要額外連接函式庫: -pthread

(g++ main.cpp -o main.out -pthread -std=c++11)

探討結果與原因:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K=5,10 | N=1W | N=10W | N=50W | N=100W |
| 方法一 | 977 | 100143 | 2539140 | 10572500 |
| 方法二 | 197, 99 | 19554, 9812 | 491799,244917 | 2048680,1103070 |
| 方法三 |  |  |  |  |
| 方法四 | 49, 18 | 4506, 1381 | 133739, 40134 | 556063, 156316 |

(單位:ms)

|  |  |  |
| --- | --- | --- |
| N=1W,10W,50W,100W | K=5 | K=10 |
| 方法一 | 977,100143,2539140,  10572500 | 977,100143,2539140,  10572500 |
| 方法二 | 197,19554,491799,2048680 | 99,9812,244917,1103070 |
| 方法三 |  |  |
| 方法四 | 49,4506,133739,556063 | 18,1381,40134,156316 |

(單位:ms)

由表一可以看到，在固定檔案數量時，除了方法一，其餘的每個切越多K速度越快，方法二中切兩倍時，速度也快兩倍，方法四中尤其明顯，當檔案數目一放大，那個速度的快慢是很大的。

從表二來看，方法四是最有效率的，並且同一個K值下，方法四比方法二快了4~5倍有，看來分thread的確是很有效率的方法。

至於為何有這個結果呢?方法二中，可以先從BubbleSort和MergeSort的時間複雜度下去看，前者是O(n^2)，後者為O(n)，單就這兩點可以看出在時間上差很多，再來拿方法二和方法四比，方法四中要分K個thread來去做運算，讓系統能在一個process內，用不同的thread來去平行運算，等於在同一個時間就能跑很多程式，提高了系統效率，但也增加了系統資源的使用。

方法三中，比方法四還複雜，因為C++在Multiprocessing中只能在Linux等unix系統去做，windows還不支援，所以要先裝虛擬機，裝好後又要去找相對應的函式庫，找到以後要去處理多個process之間如何共享記憶體，之後又要去思考如何共享vector的資料讓其他process能使用，尤其是合併的地方，必須等子process完成合併才能讓父process用合併完的陣列去合併，我雖然知道方法可惜在實作上，我尚未能想出可行的方法，若是早知道Python那麼簡單，import multiprocess as mp ，就能multiprocess了，我一定改用Python，可惜沒如果，我到期限前一天才知道Python有那麼好用的函式庫，而且在windows上有pool庫也可以達到類似的效果，HW2-Scheduling 我一定用Python下去做。