OS Homework1

資訊三乙 11027209 巫年巨

開發環境 : python , 版本3.10.7

實作方法與流程

首先講方法1就是普通的泡泡排序但是加上了bool值看這一輪有沒有排序，沒有的話代表 結束了，避免多餘的空次數做比較，優點是如果提早結束的話可以減少很多比較次數，缺點的話是worst-case會造成的話更多的比較。

第二題的題目把array切成k份，並把這些拿去分別做泡泡排序後，在當資料不為1時，一次抓兩筆資料做mergesort並合併為一筆資料，直到剩下最後一筆完成排序的資料。 由於這次的mergesort是拿bubble sort完的陣列去排序所以可以把mergesort寫成merge的形式把兩個array依照小到大合起來成一個array就好。

第三題大致步驟跟第二題很像，但是在切資料時我是每切一份生一個process去做那一份的sort，而為了切的時候資料是共享的，所以用了Python的multiprocess 庫裡面的Manager來做到share memory的效果，在process 們sort完後，主process會等到他們做完才繼續下面的步驟，之後把切完的資料拿去merge，這段我是用Pool來設定process的數量，我發現如果自己開process來，每次開現在有的份數/2的process去merge，直到只剩一筆資料為止，就能拿到答案，這題比較難的是share memory 的部分，因為Python 的share memory的資料型態與原本的資料型態不同用List舉例可以用+=的overload operator但是變成share的後他會變成ListProxy的型態，這時候對他拿去用opeator的型式的話，比如在Bubblesort中的swap的話，可能會造成速度緩慢甚至到無限迴圈的窘境，ListProxy最好只要用List有的function如append,pop,index等，雖然都是List，但這樣兩個是操作上彈性度很不同，由manager形成的ListProxy雖然也叫list的少了有些list能做的操作，不過換個方面想也是畢竟這是被共享的記憶體區段。

第四題是用thread 跟上面兩題差不多都是在切資料時就生對應數量的thread，由於thread是同一process拉出來的且有共用address space的優勢存在，所以不用煩惱共用memory的問題存在，所以整體下來方便許多，而thread跟process比有個優勢是在windows有限制同時只能有62，而thread沒有限制，在解這裡1百萬筆資料時用thread就是不錯的選擇，但thread也要注意等到做完，主process才能往下一步，必須先join在那邊等待才行，不然會有method跑完thread還在跑sort 的問題，這樣會使答案最終不正確。

探討結果和原因

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K = { 5, 17} | N = 1萬 | N = 10萬 | N = 50萬 | N = 100萬 |
| 方法1 | 4.1,4.1 | 433,433 | 11833,11833 | 55369,55369 |
| 方法2 | 0.78,0.22 | 76,23 | 2612,635 | 12412,2420 |
| 方法3 | 0.75,0.8 | 20,4 | 1512,97 | 9537,494 |
| 方法4 | 0.86,0.21 | 85,25 | 2358,610 | 11325,2110 |

表1：實驗記錄表格 (不同N的比較時間)(單位：s)

從上表可看出資料量N越來越大的時候，各方法的時間也越來越大，從方法1來看資料量上升了10倍，而執行的時間就差了100多倍左右，而方法234不同於1是多了切資料和merge這段，但是可以很明顯的看出來執行時間快了很多，方法3和4是多了耗cpu資源所以比2更快。

|  |  |  |  |
| --- | --- | --- | --- |
| N = {1,10,50,100 }萬 | K = 1 | K = 13 | K = 50 |
| 方法1 | 4.1,433,11833,55369 | 4.1,433,11833,55369 | 4.1,433,11833,55369 |
| 方法2 | 3.9,425,11351,52518 | 0.29,34,764,2946 | 0.15,8.6,299,680 |
| 方法3 | 4,421,13211,62172 | 0.59,5.51,147,952 | 0.53,4.1,42,167 |
| 方法4 | 4.2,418,12258,54563 | 0.28,31,724,2761 | 0.1,8,273,610 |

表2：實驗記錄表格(不同k的比較時間) (單位：s)

由於方法1不用切資料，所以k值影響對1來說不影響，而方法234在當資料量K越來越大的時候，可看出執行的速度會越來越快，從方法2來看資料量是50萬的時候K=50比K=13的執行時間少了很多，而方法34速度比2可以很明顯的看出來執行時間快了很多，3和4是因為耗了更多的cpu資源去建Thread和process所以比2更快。

將資料切成：1, 5, 13, 17,50 份 (K) 資料筆數：1, 10, 50, 100 萬 (N)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K = {1,5,13,17,50} | N = 1萬 | N = 10萬 | N = 50萬 | N = 100萬 |
| 方法1  (k不影響) | 4.1,4.1, 4.1,4.1,  4.1 | 433,433,  433,433,433 | 11833,11833,  11833,11833,  11833 | 55369,55369,  55369,55369,  55369 |
| 方法2 | 3.9, 0.78, 0.29,0.22 ,0.15 | 425,76,  34,23,  8.6 | 11351,2612,  764,635,  216 | 52158,12412,  2946,2420,  680 |
| 方法3 | 4, 0.75, 0.59,0.8,  0.53 | 421,20,  6.8,6.09,  4.1 | 13211,1512,  147,97,  42 | 62172,9537,  952,494,  167 |
| 方法4 | 4.2,0.86,  0.28,0.21,0.09 | 418,85,  31,25,  8.1 | 12258,2358,  724,610  201 | 54653,10325,  2761,2110,  610 |

表3：實驗記錄表格 (單位：s)

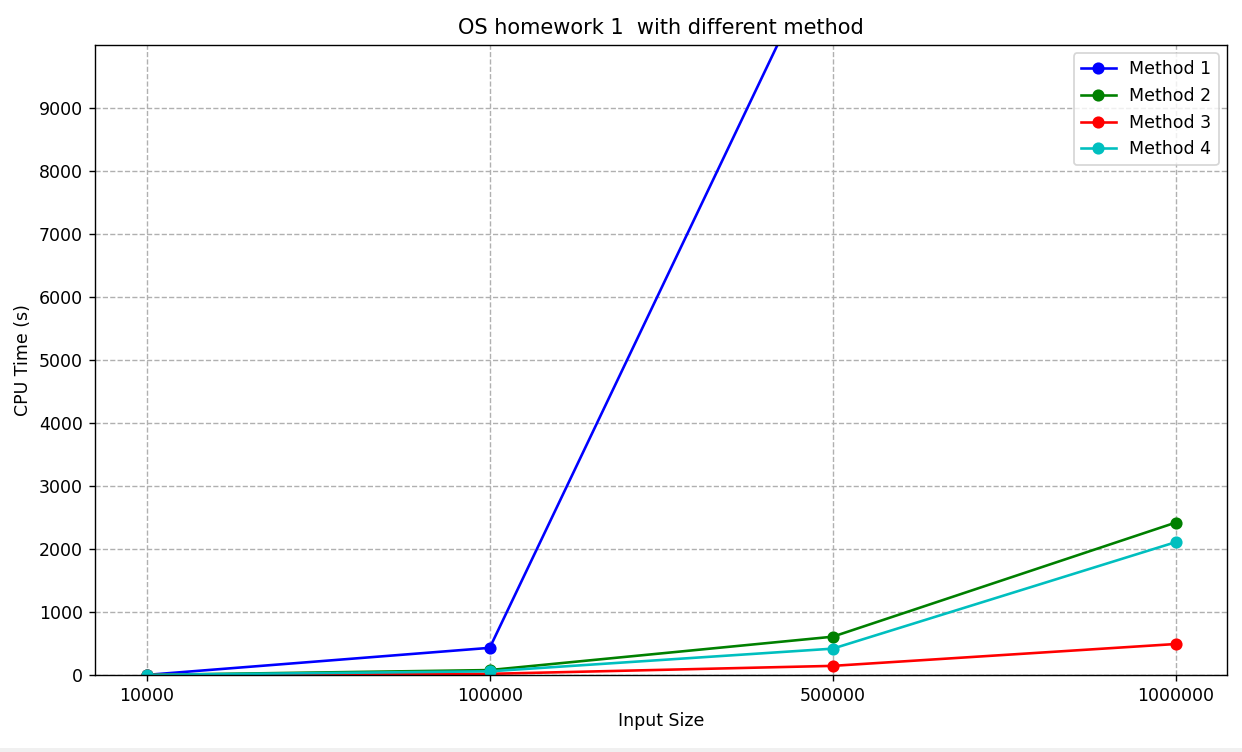


圖1：用k=1,k=5,k=13,k=17對應方法1234產生的圖

圖和表我們可以得出以下結論隨資料量n上升每次要跑的時間也會越多, BubbleSort用演算法看要O(n^2)，所以一萬筆資料跑的次數（迴圈）比一百萬筆資料差距就會到（100^2 = 10000），跑的時間上差距也會差到10000倍左右，用表格的數據去看，看到只有1萬筆資料時電腦跑大概約4秒左右，而到了100萬筆時跑的時間要到50000秒左右，所以整體看執行效率是相當的差。

方法234，前面在做泡泡排序切成k個array在去sort假設把10000切成10份，那就只要跑10個1000筆的bubble sort(比較的if次數計算1000\*1000\*10)，跟直接跑10000比(比較的if次數計算10000\*10000)，少了10倍的比較，雖然方法2還要跑merge但是用比較來說也不會有原本的那麼多， 第三和四由於是開多個process或是thread來跑與方法二相同的流程，在同樣的合理範圍k值下，理論上應該要比方法二快，因為開多個process和thread是叫CPU 去耗費資源，來分工處理不同任務。

至於這次作業中我比較有疑惑的是任務4的部分，在我原本的猜想是開多個thread照理講要跟3差不多快，但是結果卻和2差不多，我去網路查資料發現，Python的interpreter有個機制叫做GIL(global interpreter lock )，為了保護執行thread的安全性， 防止同時和修改共用的資料(如上課教的race condition)，所以同時只會有一個thread能run，運作是拿到GIL的thread可以run跑完後會釋放GIL給別的thread，所以跟方法2差不多。