Document

10727138。資訊三甲。游子諭

四種作法進行比較：

Method 1:

一個process做完全部的事情,而且程式內也沒有做像是Method4的分資料,而是全部交給BubbleSort()，所以速度一定是最慢的。

Method 2:

一個 process 中，將資料切成 k 份，並且每一份交給一個 thread 去做，而每個 thread 因為在同一個 process 中，可以同時對 process 內容中的資料進行修改，也因為同時的關係，整體效率也會比較好。

Method 3:

和 Method2 類似，但這邊是將每一份資料交給一個 process 去處理，但為了滿足同一時段只有 process 可以取用共同的資源,所以 process 之間比較不會有共同運作的狀況。

Method 4:

類似 Method 1 優化版,將一個 process 工作分發給 k 個 function 去處理，雖然沒有達到並行處理，但對於整體資料的排序節省了許多時間。

實作方法和流程：

讀檔以及處理：

我利用寫檔的 function，將檔案內的資料讀入進一維的 Vector(又稱 intVct)中，如果檔案不存在會回報錯誤，輸入 0 表示整個程式的結束。接著讀入切成幾分，並利用一開始的一維 Vector，將切完的資料再另外存到一個二維陣列(又稱 intVctVct)中，以方便 Method 2, 3, 4 使用。

Method 1:

因為第一項方法是將資料全部資料利用 BubbleSort 來進行排序, 所以建立一個全域 function，BubbleSort()，並予以呼叫來排序 intVct。

Method 2:

我利用 for 迴圈，來配合 intVctVct 中的資料份數，創造出相同數量的 thread 執行 BubbleSort()，並且執行 join()，以確保每個 thread 都確實執行完畢。至於 MergeSort 的部分我一樣創建一個全域 function，MergeSort()，方便我來使用，而因為 thread 是同時進行工作的緣故，因此我的 MergeSort()不

能直接改變 intVctVct 的資料結構，否則會天下大亂，而且每一次 Merge 都必須要等到同一層的結束，才能執行下一層的 Merge。

Method 3:

因為 process fork()的特性，parent process 有的東西，child process 都有一份 copy，因此我將每次 process 都執行一個 BubbleSort()，並執行 fork()，同時以 pid 來區分 parent 和 child，讓 child 繼續執行下一個 BubbleSort()，parent 執行 exit，避免 process 的數量因 fork()而成指數成長。

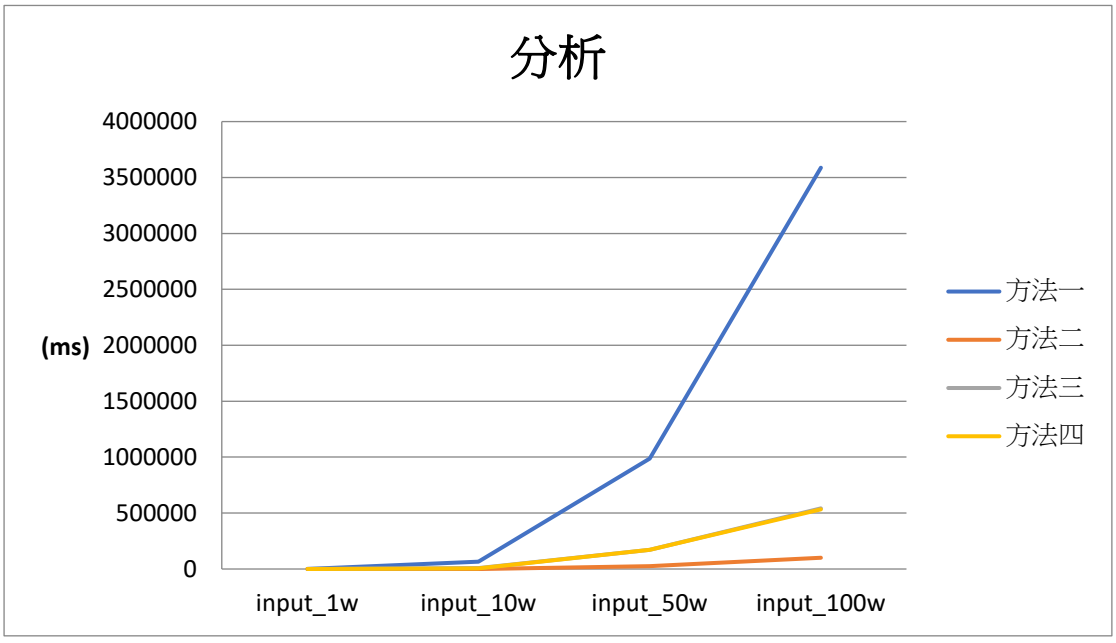
Method 4:

第四個方法有點像第一個的優化版，把工作分成數分後，讓每一份都交給一個 function 來處理，這可以大量減少 BubbleSort()所花的時間，因為 BubbleSort()的時間花費為 O(n^2)，而 MergeSort()也是同樣的道理，把資料分成數分後，分別給數個 MergeSort() function 來處理。

開發環境：

Windows 系統。CLion 上的 C/C++

分析結果和原因：



我這邊統一切的份數都是 10 份。基本上方法一、方法二和方法四都跟預想中差不多，根據前面**實作方法和流程**和**四種作法進行比較**，每個方法在概念上就有很大的效率差距，而當資料量越大時，這個差距就越明顯，至於方法三的部分主要是為了實現 process 的互斥關係，同時讓排序過的資料能夠順利傳下去給

child process，因此 process 之間是一個叫一個的模式。