標題：HW1-report
姓名：葉怡君
學號：110062529

Implementation

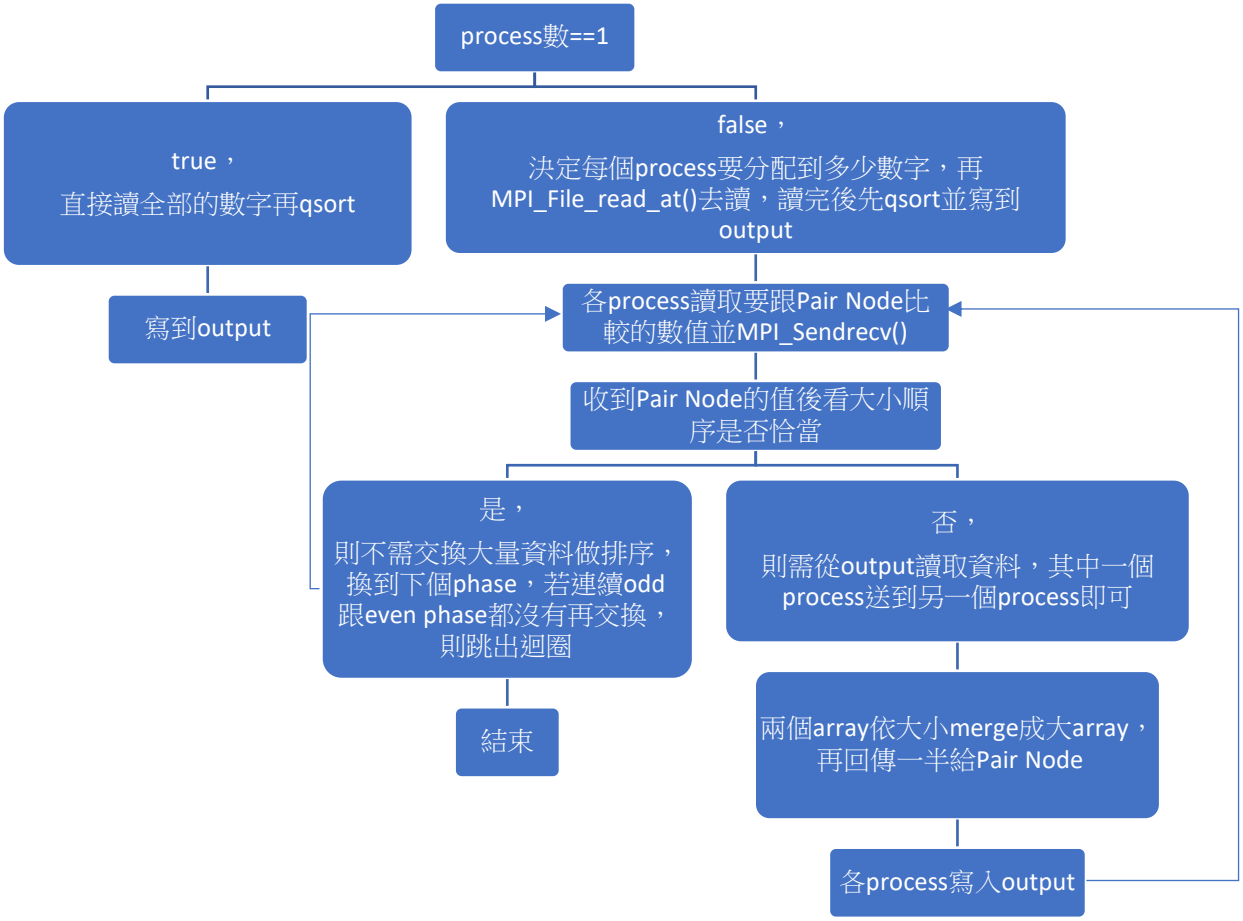


Fig 1. 流程圖

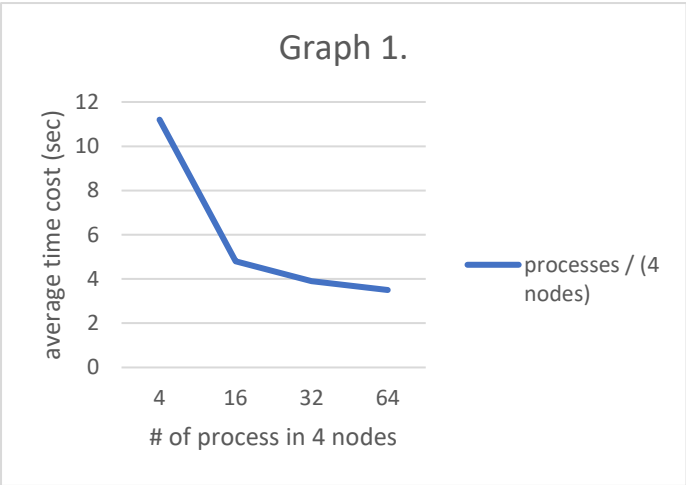
如果只有一個 process，就直接 read 到 array 做 qsort 再 output 到 file。

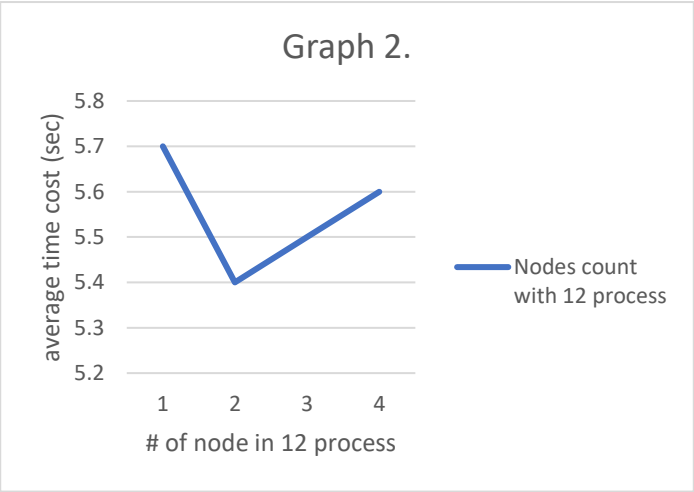
多個 processes 時，會先將數字均分到每個 process，用 rank 是否小於餘數來決定是否要多給 1 個數，也會記錄此 process 的前一個跟後一個 process 所分配到的數字個數。由於想要節省使用記憶體，所以每個 process 從 file 讀到資料後，先各自做 qsort 並寫到 output 後會先 free 掉 memory。接著進入 odd-even sort，各個 process 根據所在 phase 取得要跟 Pair Node 比較的數，再根據順序是否恰當決定兩 process 要不要從 output 讀取大量數字與 Pair Node 交換和 sort，若有需要交換，則兩 process 會從 output 讀取資料，其中一個 process 傳送到另一個 process 依大小順序 merge，因為兩 partition 已先排序過，所以 time complexity 為 O(n+m)，最後再把一半的數字回傳給 Pair Node，再各自寫入 output。

至於如何判斷已 sort 完，我用了 2 個變數(odd_c、even_c)紀錄，若各經過一次 odd 跟 even phase 但都不用交換的話即可結束。

Experiment & Analysis

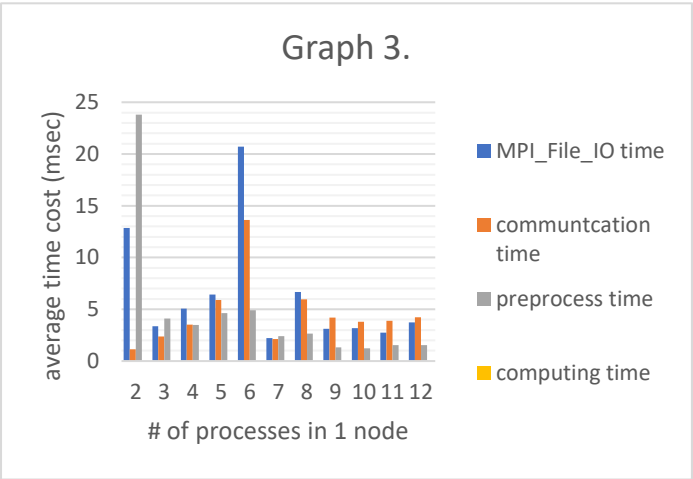
- Graph 1 & 2 以程式全部跑完所花費的時間當作 time cost，並取平均值來作圖。
  Input：33.in。有 536869888 筆資料

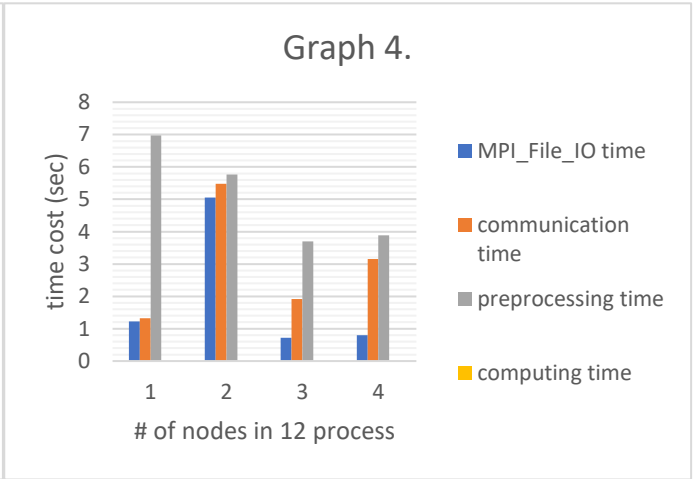




Discussion

從 Graph 1.可以看出，對大量資料來說 process 越多可以讓整個程式跑的越快，因為將大量資料分成很多 partition 去處理，花費的時間會減少許多。但正常來說應該 Node 數越多，所花的時間越少，但我的程式 scaling 不好，並沒有達到理想的結果。





- 因為每個 process 花的每種時間會有所不同，因此我取總和再除以各 process 數量來做 Graph 3.，Graph 4.因 process 數都相同，所以取總和而已，左邊圖表的 Input：09.in，有 99999 筆資料。右邊圖表的 Input：33.in，有 536869888 筆資料。

名詞定義：

MPI_File_IO time：包含 MPI_File_read、MPI_File_write

Communication time：包含 MPI_Recv、MPI_Send、MPI_Allreduce…等

Preprocessing time：從 MPI_Init 到進入 odd-even sort 迴圈之前，其中也有包含一些讀寫 file 及一開始對那個 partition 做 qsort，時間較長。

Computing time：將收到的 array 與自己的 array 合併排序的時間，因為兩 partition 已先排序過，故 time complexity 為 O(m+n)，所以沒花多少時間。

Discussion

在 Graph 3.中， preprocessing time 有因 processes 的數量變多而降低，我猜測是因為需要 qsort 的數量變少了，但在 communication 跟 MPI_File_IO 所花的時間，並沒有隨 process 數增加而減少，但不太確定 Graph 3.中 6 個 processes 時花的時間為何突然飆高。

Conclusion

我一開始寫的版本，只能通過 29 筆測資，無法處理大量的資料，因為要 allocate 的 memory 太

多，因此改變寫法，希望每個 process 使用少量的 memory 且用完就 free，避免 segmentation fault，也因此在 MPI_File_IO 花了很多時間，在 preprocessing time 因為做了 MPI_File_read、qsort 以及 MPI_File_write，所以花費的時間也很長，所以跟其他同學所花的時間有一大段落差。雖然 40 個測資都有通過就表示有達到平行化，但仍有可以 improve 的地方，但目前只想到這種寫法，希望未來可以參考其他同學的寫法，教學相長。