Hw1

I. Implementation:

1. 假設 process 數目有 k 個，資料料有 n 個
   每個 process 處理 n/k 個資料
   如果 n/k 無法整除，則會再給前 n%k 個 process 一筆 data
2. MPI 根據 1. 得到的資料量去讀取對應得資料
3. 讀取完資料以後每個 process 會對自己所擁有的資料進行 sort
4. 執行 odd-even sort
   在每個 phase 當中，左邊的 process 會傳送所有 Data 給右邊的 process
   右邊的 process 會負責 merge 兩個 sort 完的 Data 再根據左邊 process 所擁有的 Data 給傳回一部份給左邊的 process

   如果 process 數目為奇數個，則在 even phase 最後一個 process 不必工作
   在 odd phase 第一個 process 不必工作

   如果 process 數目為偶數個，在 even phase 所有的 process 都要工作
   在 odd phase 第一個與最後一個 process 不必工作
5. 在每個 odd phase 檢查是否排序完成，如果排序完成就結束，如果排序沒有完成就繼續執行 step 4

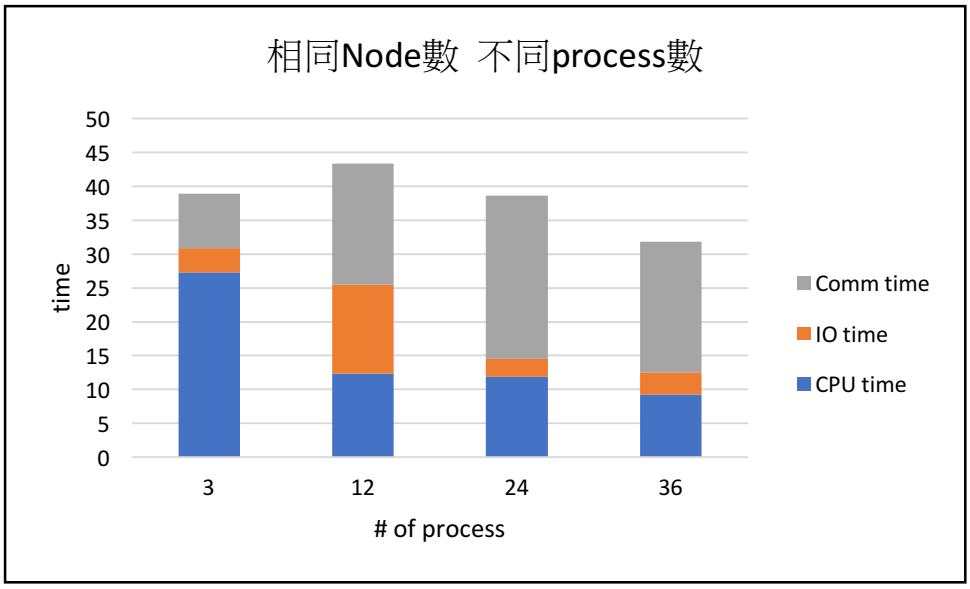
II. Experiment & Analysis

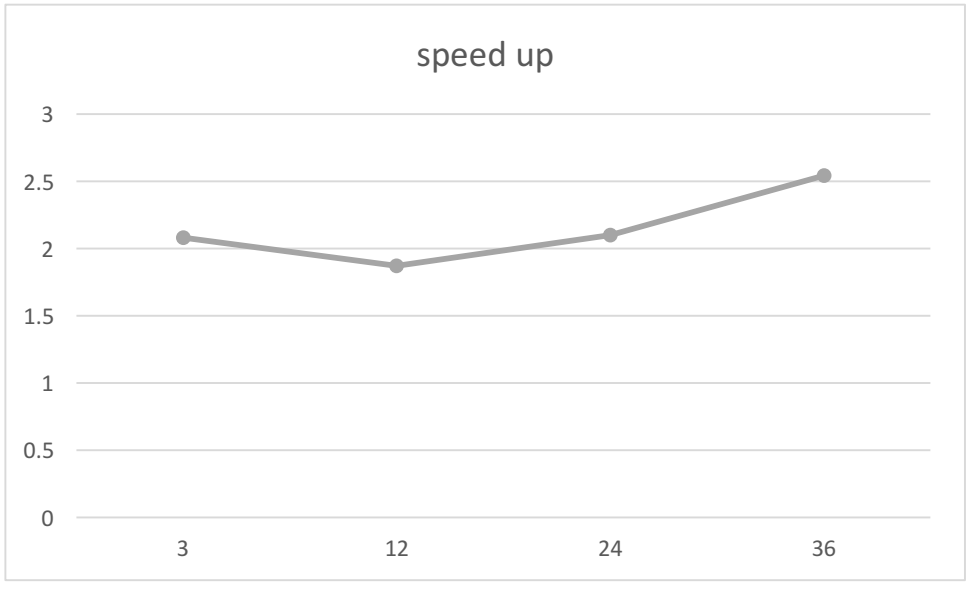
1. Methodology
   i. System Spec: Server
   ii. Performance Metrics
   利用 MPI_Wtime()取得時間
   IO time: 從 MPI_File_open 開始計算直到執行，MPI_File_close
   Comm time: MPI_Send 與 MPI_Recv 結束
   CPU time: 程式開始的時間扣除 IO time 及 Comm time

2. Plots: Speedup Factor & Time Profile

Data Size: 536870864

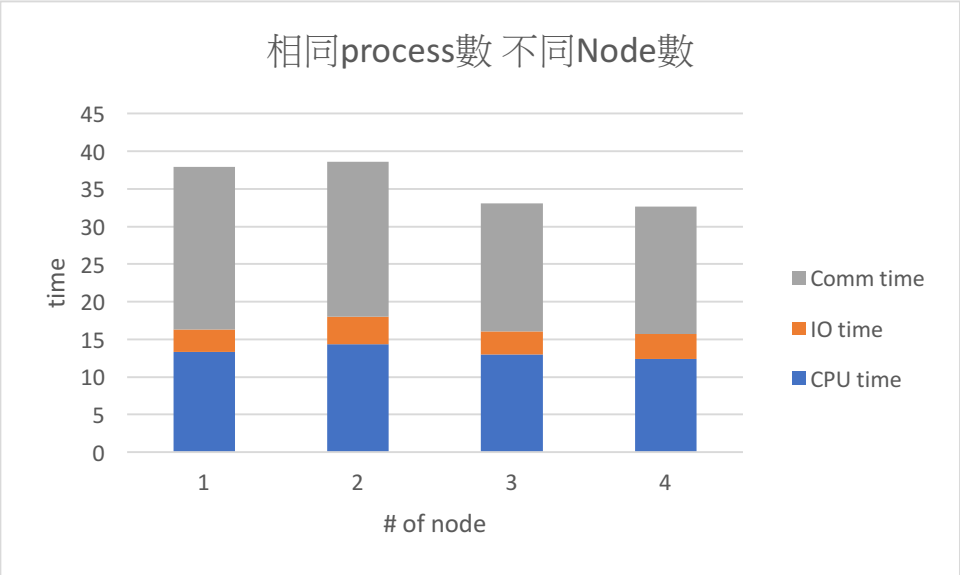
node 數: 3

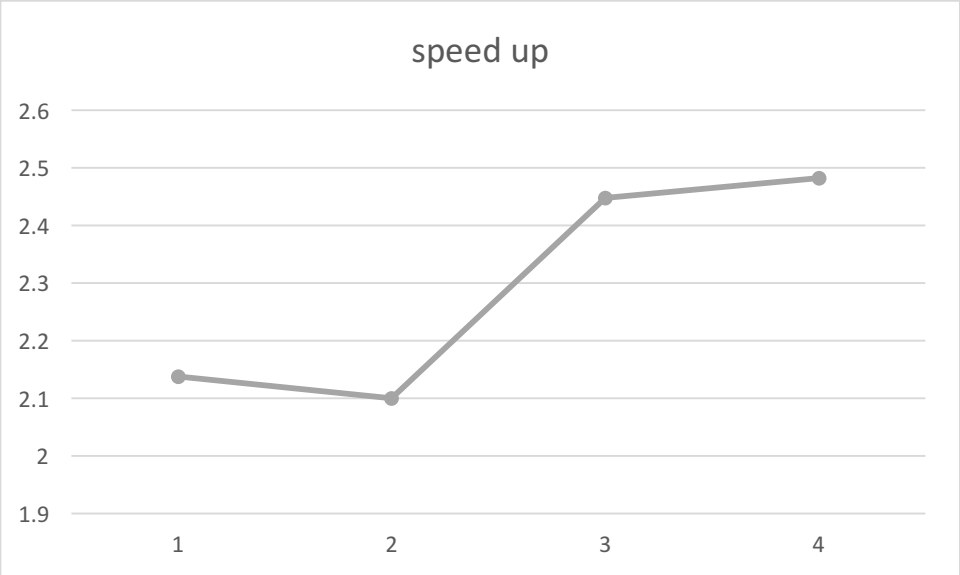




Data Size: 536870864

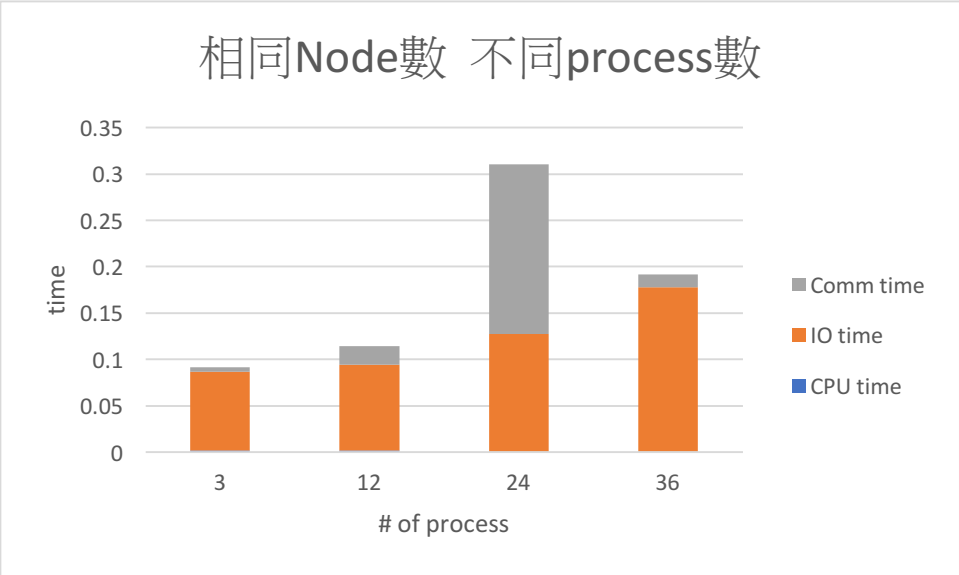
process 數: 12

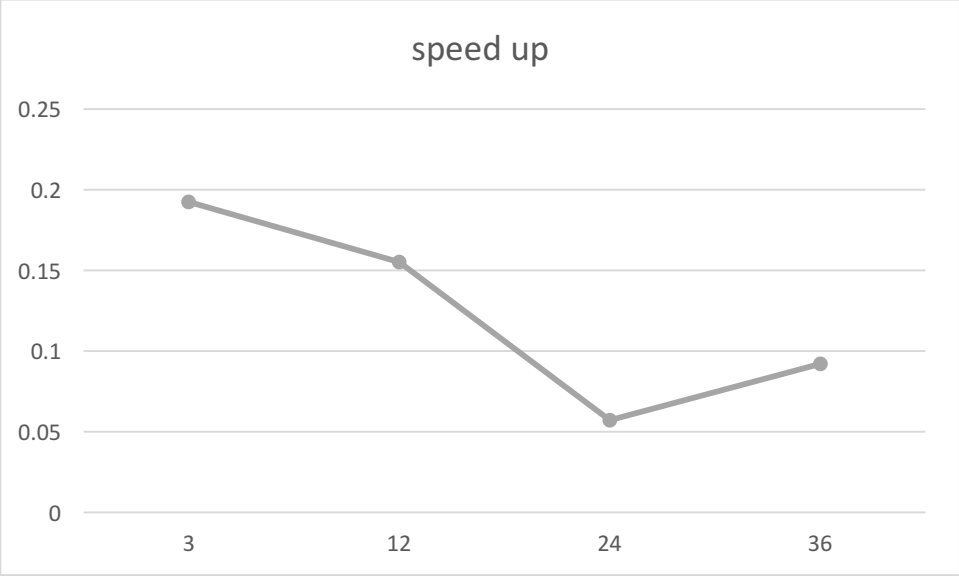




Data Size:63942

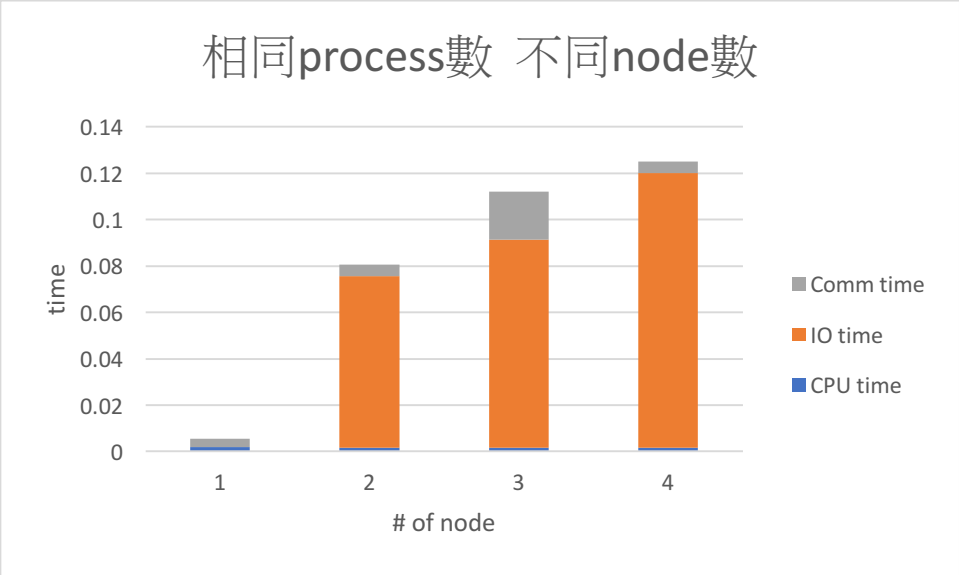
node 數: 3

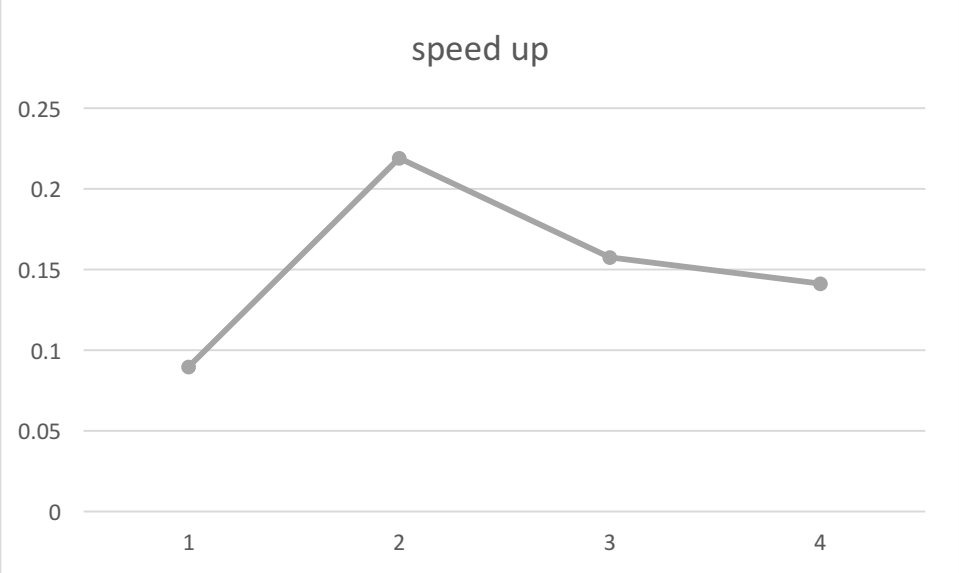




Data Size:63942

process 數: 12





3. Discussion:
   i. 更多的 process 會有更多的 comm overhead，選擇適當的 process 可以有效得降低 comm 的時間．
   ii. 每個 process 分配到的 data size 幾乎都一樣，所以能夠達成較好的 scalability，但是 comm 得部分因為都使用 block 的方式，所以很容易造成 comm overhead 過大的情況，這一部份或許可以改進．

4. Conclusion:
   i. 針對第一組結果，可以得知當 process 數量上升時，comm time 也會跟著上升，但是 CPU time 會相對應的下降，而 IO time 的時間大致上都差不多，12 個 process 數目的 IO time 可能是因為有其他的 user 導致 resource 分配不均．整體上而言，使用更多 process 程式的 speedUp 成自然對數的形式．
   ii. 相同數量的 process 分散在各個不同的 Node，他的執行速度會些微上升，可能因為使用到資源很多的 node．
   iii. 相對於 small data，big data 比較適合使用平行程式，因為 small data 的 comm overhead 會比 CPU time 多很多，所以程式的執行速度下降．